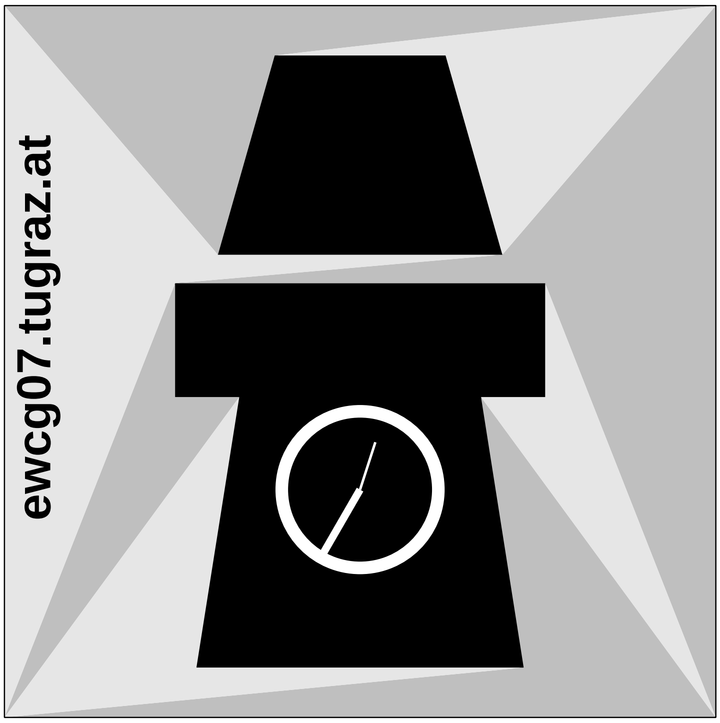
Collection of Abstracts

EWCG GRAZ



ewcg07.tugraz.at

EWCG'07 March 19–21 2007, Graz, Austria

supported by

OESTERREICHISCHE COMPUTER GESELLSCHAFT
AUSTRIAN COMPUTER SOCIETY

IBM

Stadt GRAZ Wissenschaft
Bürgermeister Siegfried Nagl

TU Graz

Das Land Steiermark
Wissenschaft

Das Land Steiermark
Landeshauptmann Franz Voves

Collection of Abstracts of the
23rd European Workshop on Computational Geometry
Technische Universität Graz, Austria
March 19–21, 2007

For information about obtaining copies of this volume contact

Oswin Aichholzer
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II
8010 Graz
Austria
`oaich@ist.tugraz.at`

ISBN: 978-3-902465-62-7

Verlag der Technischen Universität Graz
`www.ub.tugraz.at/Verlag`

Bibliographic information published by *Die Deutsche Bibliothek*. *Die Deutsche Bibliothek* lists this publication in the *Deutsche Nationalbibliographie*, detailed bibliographic data is available at `http://dnb.ddb.de`.

Editors:
Oswin Aichholzer and Thomas Hackl

Printed in Austria by:
Druck- und Kopierzentrum der Technischen Universität Graz

Preface

The **23rd European Workshop on Computational Geometry** (EWCG'07) was held at the University of Technology in Graz (Austria) on March 19th – 21st, 2007. More information about the workshop can be found at `http://ewcg07.tugraz.at`. This collection of extended abstracts contains the 60 scientific contributions as well as three invited talks presented at the workshop. The submission record of over 70 abstracts from more than 20 different countries, covering a wide range of topics, shows that Computational Geometry is a lively and still growing research field in Europe.
We would like to thank all the authors for submitting papers and presenting their results at the workshop. We are especially grateful to our keynote speakers Bernard Chazelle, Erik Demaine, and Herbert Edelsbrunner for accepting our invitation. Special thanks go to Bettina Speckmann for providing us with the framework of the conference web page and specific LaTeX-classes.
The collection of abstracts at hand will also be available electronically from the workshop's web portal at `http://ewcg07.tugraz.at/EuroCG2007Abstracts.pdf`. Following the tradition of the workshop, many contributions present ongoing research, and it is expected that most of them will appear in a more complete version in scientific journals. Selected papers from the workshop will be invited to a special issue of "Computational Geometry: Theory and Applications". We thank the co-editors in-chief, Kurt Mehlhorn and Jörg Rüdiger Sack, for their cooperation.
Finally, we are grateful to Graz University of Technology for providing the necessary infrastructure, and to our sponsors for their support: Stadt Graz - Wissenschaft, Stadt Graz - Bürgermeister Siegfried Nagl, Das Land Steiermark - Wissenschaft, Das Land Steiermark - Landeshauptmann Franz Voves, Österreichische Computer Gesellschaft, IBM - Geschäftsstelle Graz.

Oswin Aichholzer and Thomas Hackl (Editors)

Program Committee

Oswin Aichholzer
Franz Aurenhammer
Thomas Hackl
Hannes Krasser

Organizing Committee

Oswin Aichholzer
Thomas Hackl
Bernhard Kornberger
Petra Pichler
Birgit Vogtenhuber

1st EWCG 1983: Zürich, Switzerland	9th EWCG 1993: Hagen, Germany	17th EWCG 2001: Berlin, Germany
2nd EWCG 1984: Bern, Switzerland	10th EWCG 1994: Santander, Spain	18th EWCG 2002: Warsaw, Poland
3rd EWCG 1985: Karlsruhe, Germany	11th EWCG 1995: Linz, Austria	19th EWCG 2003: Bonn, Germany
4th EWCG 1988: Würzburg, Germany	12th EWCG 1996: Münster, Germany	20th EWCG 2004: Seville, Spain
5th EWCG 1989: Freiburg, Germany	13th EWCG 1997: Würzburg, Germany	21st EWCG 2005: Eindhoven, The Netherlands
6th EWCG 1990: Siegen, Germany	14th EWCG 1998: Barcelona, Spain	22nd EWCG 2006: Delphi, Greece
7th EWCG 1991: Bern, Switzerland	15th EWCG 1999: Antibes, France	23rd EWCG 2007: Graz, Austria
8th EWCG 1992: Utrecht, The Netherlands	16th EWCG 2000: Eilat, Israel	24th EWCG 2008: Nancy, France

More information about previous workshops can be found at `http://www.eurocg.org`.

Table of Contents

Wednesday, 21 March 2007

Introduction to persistent homology

Herbert Edelsbrunner

Arts and Sciences Professor of Computer Science and Mathematics
Duke University,
Computer Science Department
Box 90129, Durham, NC 27708, USA
edels@cs.duke.edu

Persistent homology is an algebraic tool for measuring topological features of shapes and functions. It casts the multi-scale organization principle we frequently observe in nature into a mathematical formalism. This talk will introduce the basic concepts, present a few applications, and survey extensions of the original concept motivated by applications.

Unfolding Lattice Polygons on Some Lattice Polyhedra*

Sheung-Hung Poon†

Abstract

We consider the problem of unfolding lattice polygons embedded on the surface of some classes of lattice polyhedra. We show that an unknotted lattice polygon embedded on a lattice orthotube or orthotree can be convexified in $O(n)$ moves and time, and a lattice polygon embedded on a lattice Tower of Hanoi or Manhattan Tower can be convexified in $O(n^2)$ moves and time.

1 Introduction

Graph reconfiguration problems have wide applications in contexts including robotics, molecular conformation, animation, wire bending, rigidity and knot theory. The motivation for reconfiguration problems of lattice graphs arises in applications in molecular biology and robotics. For instance, the bonding-lengths in molecules are often similar [8, 13, 14], as are the segments of some types of robot arms.

A *unit tree* (resp. *unit polygon*) is a tree (resp. polygon) containing only edges of unit length. An *orthogonal tree* (resp. *orthogonal polygon*) is a tree (resp. polygon) containing only edges parallel to coordinate-axes. A *lattice tree* (resp. *lattice polygon*) is a tree (resp. polygon) containing only edges from a square or cubic lattice. Note that a lattice tree or polygon is basically a unit orthogonal tree or polygon. A *lattice polyhedron* is a polyhedron whose surface is the union of lattice faces from a cubic lattice. A graph is *simple* if non-adjacent edges do not intersect. We consider the problem about the reconfiguration of a simple chain, polygon, or tree through a series of continuous motions such that the lengths of all graph edges are preserved and no edge crossings are allowed. A tree can be *straightened* if all its edges can be aligned along a common straight line such that each edge points "away" from a designated root node. In particular, a chain can be straightened if it can be stretched out to lie on a straight line. A polygon can be *convexfied* if it can be reconfigured to a convex polygon. We say a chain or tree is *locked* if it cannot be straightened. We say a polygon is *locked* if it cannot be convexified. We consider one *move* in the reconfiguration as a continuous monotonic change for the joint angle at some vertex, or a continuous axial rotation of one of its angle side around its another angle side in 3D, during which no edge crossings occur.

In four dimensions or higher, a polygonal tree can always be straightened, and a polygon can always be convexified [9]. In two dimensions, a polygonal chain can always be straightened and a polygon can always be convexified [11, 17, 6]. However, there are some trees in two dimensions that can lock [3, 10, 15]. In three dimensions, even a 5-chain can lock [4]. Alt et al. [2] showed that deciding the reconfigurability for trees in two dimensions and for chains in three dimensions is PSPACE-complete. However the problem of deciding straightenability for trees in two dimensions and for chains in three dimensions remains open. Due to the complexity of the problems in two and three dimensions, some special classes of trees and polygons have been considered. Cantarella and Johnston [7] showed that a unit 5-chain in three dimensions can always be straightened. Demaine et al. [12] even studied interlocked configurations of several short chains in three dimensions. In particular, they showed that two 3-chains cannot interlock, but three of them can. They also showed that a 3-chain and a 4-chain can interlock. Poon [15] showed that a unit tree of diameter 4 in two dimensions can always be straightened. In their paper, they posed a challenging open question whether a unit tree in either two or three dimensions can always be straightened.

Biedl et al. [4] proved that an open chain on the surface of a convex polyhedron can always be straightened. In this paper, we show that an unknotted lattice polygon embedded on a lattice orthotube, orthotree, Tower of Hanoi, and Manhattan Tower can always be convexified.

2 Preliminaries

A *near-lattice edge* is a unit-length edge within distance $\epsilon \ll 1$ from some lattice edge. The particular lattice edge is called the *core edge* of the corresponding near-lattice edge. A *core vertex* is a lattice vertex of some core edge. A *near-lattice tree* (resp. *near-lattice polygon*) is a tree (resp. polygon) that contains

*The author was supported by the Netherlands' Organization for Scientific Research (NWO) under project no. 612.065.307.

†Department of Mathematics & Computer Science, Technical University of Eindhoven, spoon@win.tue.nl

only near-lattice edges. Suppose P is a near-lattice tree or polygon. The *core* of P, denoted by $K(P)$, is the union of core edges for all edges in P. A *spring* in P is the set of edges in P converging to a common lattice edge. A spring with only one edge is called a *singleton.* A near-lattice edge or spring is called *embedded or lying on* a lattice polyhedron if its core is embedded on the lattice polyhedron.

3 Lattice Orthotube

A *lattice orthotube* is a lattice polyhedron made out of boxes that are glued face-to-face such that its face-to-face contact graph is a path or cycle. A lattice orthotube is called *open* if its face-to-face contact graph is a path; otherwise it is called *closed.* In an open lattice orthotube, the two blocks whose degrees in the face-to-face contact graph are one are called the *end blocks* of the given orthotube. An *end face* of an open orthotube is a face of its end block such that it is opposite to the face which is the intersection of the end block and the second last end block.

Remark that there are some orthogonal polygons embedded on some orthogonal polyhedra that can lock as shown in Figure 1(a), and there are some lattice polygons embedded on some closed lattice orthotubes can knot as shown in Figure 1(b). This motivates that we consider the lattice polygons embedded on open lattice orthotubes, and the *unknotted* lattice polygons embedded on closed lattice orthotubes.

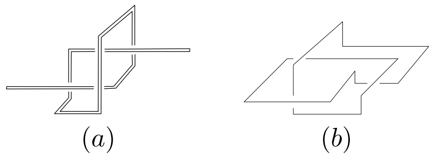


Figure 1: (a) A 3D locked orthogonal polygon. (b) A 3D knotted lattice polygon.

3.1 Open Lattice Orthotube

In this subsection, we will show that lattice polygons embedded on open lattice orthotubes can always be convexified.

Consider a near-lattice polygon embedded on open lattice orthotube. The end block of the orthotube is called *free* if its end face does not contain any edge from the core of the given embedded near-lattice polygon. It is clear that the free end blocks of an open orthotube do not help in our unfolding process and can be truncated away. We thus assume the end block of any orthotube mentioned below is not free. Our algorithm proceeds by folding up the polygon from the non-free end blocks of the orthotube successively. Suppose we are given a near-lattice polygon embedded on a lattice orthotube at the beginning of each folding step. We fold up the part of the given near-lattice polygon lying on the end block onto the springs of the second last end block using constant number of moves. After one folding step, again we obtain back a near-lattice polygon. We repeat this step until the remained orthotube contains only one lattice cell. Now it is clear that the near-lattice polygon embedded on one lattice cell can be unfolded to a convex polygon straightforwardly. We first need the following lemma on how to perform a folding step. Then we summarize our result in Theorem 2.

Lemma 1 *Given a near-lattice polygon P embedded on an open lattice orthotube $\mathcal{Q}$ such that both end blocks of $\mathcal{Q}$ are not free, and $\mathcal{Q}$ contains more than one lattice cells. Then the part of P lying on an end block of $\mathcal{Q}$ can be folded onto some springs on the second last end block so that the current end block becomes free.*

Proof. Suppose the end face of the orthotube $\mathcal{Q}$ is facing to the right. We divide into three cases depending on how many core edges of P lie on the end face.

Case 1: The end face contains one core edge. Then the end block can be folded up as shown in Figure 2.

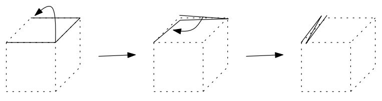


Figure 2: *Case 1* of folding up an end block.

Case 2: The end face contains two core edges. Then for the two subcases in Figure 3(a) or (b), the end block can be transformed into *Case 1* as shown in the figures; for the subcase in Figure 3(c), it can be treated as two occurrences of *Case 1.* Consequently, the resulting end block can be folded up by applying once or twice the operation of *Case 1.*

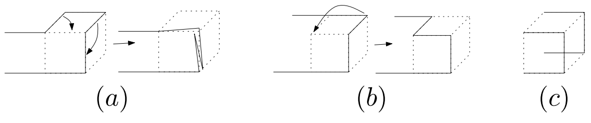


Figure 3: Case 2 of folding up an end block.

Case 3: The end face contains three core edges. Then the end block can be transformed into *Case 1* as shown in Figure 4.

Note that for any of the operations above, only the joint angles at the end vertices of the end edges of a constant number of related springs are changed. Thus folding up the end block takes a constant number of moves and time. □

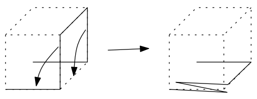


Figure 4: *Case 3* of folding up an end block.

Theorem 2 *A lattice polygon embedded on an open lattice orthotube can be convexified in $O(n)$ moves and time.*

3.2 Closed Lattice Orthotube

Given an unknotted lattice polygon embedded on a closed lattice orthotube. First it is clear that at any cross section, the intersection of the given lattice polygon and the cross-section cutting plane contains either zero, two or four corner vertices. If there is a point on lattice orthotube where the cross section does not intersect the given lattice polygon, then after cutting the closed orthotube open, we can use the algorithm for open lattice orthotube to unfold the polygon. Otherwise there is no such point where the cross section does not intersect the given lattice polygon. Then it is clear that there must exist some cross section at some lattice points where all the four corner vertices lie in the intersection of the given polygon and the current cross-section cutting plane, and the structure of its neighborhood on the polygon is in one of the cases in Figure 5.

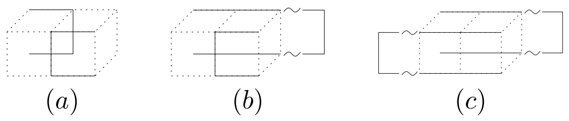


Figure 5: Cases for folding a closed lattice orthocube.

For case (a) in the figure, the closed orthocube can still be cut along the cross section to obtain an open lattice orthocube. In both cases (b) and (c), by successively applying the folding operation of *Case 1* to fold the end block of an open lattice orthocube, we can transform them to case (a) by eliminating the long U-turns. Therefore, we have the following theorem.

Theorem 3 *An unknotted lattice polygon embedded on a closed lattice orthotube can be convexified in $O(n)$ moves and time.*

4 Lattice Orthotree

A *lattice orthotree* is a lattice polyhedron made out of boxes that are glued face-to-face such that its face-to-face contact graph is a tree. In a lattice orthotree, those blocks whose degrees in the face-to-face contact graph are one are called the *end blocks* of the given orthotree. To convexify a lattice polygon embedded on a lattice orthotree, the algorithm runs in the same fashion as that for an open lattice orthotube. We fold up the polygon from the end blocks successively.

Theorem 4 *A lattice polygon embedded on a lattice orthotree can be convexified in $O(n)$ moves and time.*

5 Lattice Towers

Let Z_k be the plane $z = k$ for $k \geq 0$. A *Manhattan Tower* $\mathcal{Q}$ is an orthogonal polyhedron such that

1. $\mathcal{Q}$ lies in the halfspace $z \geq 0$ and its intersection with Z_0 is a simply connected orthogonal polygon;
2. For $j > k \geq 0$, $\mathcal{Q} \cap Z_j \subset \mathcal{Q} \cap Z_k$: the cross section at a higher level is nested in that at a lower level.

A *Tower of Hanoi* $\mathcal{Q}$ is a Manhattan Tower such that its intersection with Z_k for $k \geq 0$ is either empty or a simply connected orthogonal polygon.

5.1 Lattice Tower of Hanoi

Given a lattice polygon embedded on a lattice Tower of Hanoi. The overall intuition of the unfolding algorithm is to press level by level vertically downwards from the highest level. Let's first consider the detail for pressing the highest level L down to the second highest level L' under the condition that L' is not the lowest level. Notice that between L and L', there are vertically lattice polygon edges connecting them, which we call *legs*. And we also call the end vertex of the leg at L' the *foot* of the leg. To press level L to level L', we press the maximal polygon path on level L one by one onto the level L'. More precisely, each maximal polygon path α on level L has two legs connecting to level L'. We will collapse one leg and pull one edge of α towards one of the collapsed leg. On the other end, the other leg is pulled to replace the position of one end edge of α, and the end edge of α is pulled to replace the position second last end edge of α, and so on so forth. See the operation (a) in Figure 6. Remark that at the end of the pressing step, we

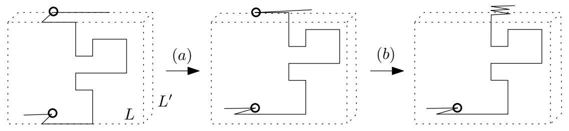


Figure 6: Pressing a path from level L down to L'.

don't really press α down to the level L'; but we keep a level higher but very close the level L' to prevent edge crossing with polygon edges in level L'. Later on, this treatment also gives us some convenience to recognize which springs are "inside" and which springs are "outside" for those springs with the same core edges. We then have the convention that the highest spring is the "most inside". After the pressing step, if the core edge of some spring has degree one at its one end, we need to collapse those dangling springs. See the operation (b) in Figure 6. It is clear that this pressing step takes at most $O(n)$ moves and time. Notice that each time we press a path down to one level lower, two vertical legs are collapsed. There can be at most $O(n)$ vertical legs. All the pressing operations take $O(n^2)$ moves and time.

After all the pressing steps, we obtain near-lattice polyhedron of height one. At this stage, using a generalized end-block collapsing similar to what we did for orthotubes and orthotrees, we can fold up the current near-lattice polygon to become a near-lattice unit square, which can then be convexified straightforwardly. This end-block collapsing process can be realized such that it takes $O(n^2)$ moves and time. However, its detail is eliminated in this abstract. Hence, we have the following theorem.

Theorem 5 *A lattice polygon embedded on a lattice Tower of Hanoi can be convexified in $O(n^2)$ moves and time.*

5.2 Lattice Manhattan Tower

Given a lattice polygon embedded on a lattice Manhattan Tower. The algorithm is the same as that for lattice Tower of Hanoi. The only difference is that when we press the highest level L to the second highest level L', we need to press several separate orthogonal polygonal regions on L instead of only one for lattice Tower of Hanoi. Thus we have the following theorem.

Theorem 6 *A lattice polygon embedded on a lattice Manhattan Tower can be convexified in $O(n^2)$ moves and time.*

6 Open Problems

We conjecture that a lattice polygon embedded on a general lattice polyhedron can always be convexified. The conjecture [16] that any unknotted lattice polygon in 3D can always be convexified is still open.

References

[1] J. Alberto-Calvo, D. Krizanc, P. Morin, M. Soss, G. Toussaint. Convexifying polygons with simple projections. *Information Processing Letters*, 80(2):81-86, 2001.

[2] H. Alt, C. Knauer, G. Rote, and S. Whitesides. The Complexity of (Un)folding. In *Proc. 19th ACM Symposium on Computational Geometry (SOCG)*, 164–170, 2003.

[3] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. A Note on Reconfiguring Tree Linkages: Trees can Lock. *Discrete Applied Mathematics*, volume 117, number 1-3, pages 293-297, 2002.

[4] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. Locked and Unlocked Polygonal Chains in Three Dimensions. *Discrete & Computational Geometry*, volume 26, number 3, pages 269-281, October 2001.

[5] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding Some Classes of Orthogonal Polyhedra. In *Proc. of 10th Canadian Conference on Computational Geometry (CCCG)*, 1998.

[6] J. Cantarella, E.D. Demaine, H. Iben, and J. O'Brien. An Energy-Driven Approach to Linkage Unfolding. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry (SoCG)*, 134–143, 2004.

[7] J. Cantarella and H. Johnston. Nontrivial embedding of polygonal intervals and unknots in 3-space. *J. Knot Theory Ramifications*, 7, 1027–1039, 1998.

[8] H.S. Chan and K.A. Dill. The protein folding problem. *Physics Today*, pages 24–32, February 1993.

[9] R. Cocan and J. O'Rourke. Polygonal Chains Cannot Lock in 4D. *Computational Geometry: Theory & Applications*, 20, 105–129, 2001.

[10] R. Connelly, E. Demaine, and G. Rote. Infinitesimally Locked Self-Touching Linkages with Applications to Locked Trees. In *Physical Knots: Knotting, Linking, and Folding of Geometric Objects in R^3*, edited by J. Calvo, K. Millett, and E. Rawdon (editors), 2002, pages 287–311, American Mathematical Society.

[11] R. Connelly, E.D. Demaine, and G. Rote. Straightening Polygonal Arcs and Convexifying Polygonal Cycles. *Discrete & Computational Geometry*, volume 30, number 2, 205–239, 2003.

[12] E. D. Demaine, S. Langerman, J. O'Rourke, and J. Snoeyink. Interlocked Open Linkages with Few Joints. In *Proc. 18th Annual ACM Symposium on Computational Geometry (SoCG)*, Barcelona, Spain, June 5-7, pages 189–198, 2002.

[13] K.A. Dill. Dominant forces in protein folding. *Biochemistry*, 29(31), 7133–7155, August 1990.

[14] B. Hayes. Prototeins. *American Scientist*, 86, 216–221, 1998.

[15] S.-H. Poon. On Straightening Low-Diameter Unit Trees. In *Proc. 13th International Symposium on Graph Drawing (GD)*, 519–521,2005.

[16] S.-H. Poon. On Unfolding Lattice Polygons/Trees and Diameter-4 Trees. In *Proc. 12th Annual International Computing and Combinatorics Conference (COCOON)*, 186–195, 2006.

[17] I. Streinu. A combinatorial approach for planar non-colliding robot arm motion planning. In *Proc. 41st ACM Annual Symposium on Foundations of Computer Science (FOCS)*, 443–453, 2000.

Morphing Polygonal Lines: A Step Towards Continuous Generalization

Damian Merrick* Martin Nöllenburg† Alexander Wolff‡ Marc Benkert†

Abstract

We study the problem of morphing between two polylines that represent a geographical feature generalized at two different scales. Some cartographical generalizations are not handled well by traditional morphing algorithms, e.g., when three consecutive bends in a river or road are generalized to two bends at a smaller scale. We attempt to handle such cases by modeling the problem as an optimal matching between characteristic parts of each polyline. A dynamic programming algorithm is presented that solves the matching problem in $O(nm)$ time, where n and m are the respective number of characteristic parts of the two polylines. We also show the results of applying this algorithm on real road data.

1 Introduction

Visualization of geographic information in the form of maps has been established for centuries. Depending on the scale of the map the level of detail of displayed objects must be adapted in a *generalization* process. Be it done manually or (semi-)automatically, generalization methods usually produce a map at a single target scale. This is a well-studied field, surveyed, for example, by Weibel et al. [12].

In current geographic information systems users can interactively zoom in and out of the map, ideally at arbitrary scales and with smooth, continuous changes. However, current approaches are often characterized by a fixed set of scales or by simply zooming graphically without modifying map objects. To overcome these deficiencies *continuous* generalization methods are needed.

This paper studies an algorithm for continuously generalizing linear features like rivers or roads between their representations at two scales. Instead of line-simplification methods with a single target scale, we consider morphing between a source and a target scale in a way that keeps the maps at intermediate scales meaningful. Of specific interest are morphings that can deal with a certain amount of exaggeration and schematization such as reducing the number but increasing the size of road serpentines at the smaller scale. Our method first partitions the input polyline into characteristic segments and then defines distances between these segments. Based on those distances we compute an optimum morphing of the polyline segments at the two input scales using dynamic programming. We have implemented a prototype of the algorithm and compare its output with that of a simple linear morph.

2 Related work

Cecconi and Galanda [3] study adaptive zooming for web applications with a focus on the technical implementation. While maps can be produced at arbitrary scales there is no smooth animation of the zooming. A set of continuous generalization operators is presented by van Kreveld [11], including two simple algorithms for morphing a polyline to a straight-line segment.

Existing algorithms for the geometric problem of finding an optimal intersection-free geodesic morphing between two simple, non-intersecting polylines [2] cannot be applied here because the two input polylines intersect in general. Given the correspondence between nodes of two plane graphs, Erten et al. [5] and Surazhsky and Gotsman [10] compute trajectories for an intersection-free morphing using compatible triangulations. In computer graphics, Cohen et al. [4] match point pairs sampled uniformly along two (or more) parametric freeform curves. They compute an optimal correspondence of the points w.r.t. a similarity measure based on the tangents of the curves. The algorithm is similar to ours in that it also uses dynamic programming to optimize the matching, but it does not take into account the characteristic points of geographic polylines. Samoilov and Elber [9] extend the method of Cohen et al. by eliminating possible self-intersections during the morphing.

3 Model and algorithm

In this paper, we consider the problem of morphing between two given polylines, each generalized at a different scale. Our algorithms to solve the problem can

*School of Information Technologies, University of Sydney and National ICT Australia (funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council), dmerrick@it.usyd.edu.au

†Faculty of Informatics, Karlsruhe University, Germany, {noelle,mbenkert}@iti.uka.de. Supported by grant WO 758/4-2 of the German Research Foundation (DFG).

‡Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, The Netherlands, http://www.win.tue.nl/∼awolff

be extended in a straightforward manner to finding a series of morphs across many scales, by solving each pair of polylines in the problem independently. The same approach can be applied to two networks with identical topology.

The problem of morphing between two polylines is two-fold. Firstly, a correspondence must be found between points on the two lines. Secondly, trajectories that connect pairs of corresponding points must be specified. Here we focus on the correspondence problem and assume straight-line trajectories.

In addressing the correspondence problem, our goal is to match parts of each polyline that have the same semantics, e.g. represent the same series of hairpin bends in a road at two levels of detail. We wish to do this in a way that allows the *mental map* to be retained as much as possible. We therefore want to minimize the movement of points from one polyline to another. To create a morph with these desired properties, we first detect characteristic points of a polyline (Section 3.1) and use these to find an optimum correspondence (Section 3.2).

Formally, we are given two polylines f and g in the plane $\mathbb{R}^2$. In the correspondence problem we need to find two continuous, monotone parameterizations $\alpha : [0,1] \to f$ and $\beta : [0,1] \to g$, such that $\alpha(0)$ and $\beta(0)$ map to the first points of f and g and $\alpha(1)$ and $\beta(1)$ map to the last points, respectively. These two parameterizations induce the correspondence between f and g: for each $u \in [0,1]$ the point $\alpha(u)$ is matched with $\beta(u)$.

3.1 Detection of characteristic points

In order to solve the correspondence problem, we first need to divide each polyline into subpolylines to be matched up. We do this by locating points on each line that are considered to be characteristic of the line; each of these characteristic points then defines the end of one subpolyline and the start of another.

Previous work on generalization notes the importance of inflection points, bend points, and start and end points in defining the character of a line [8]. To find such points, we process each of the vertices in a polyline in order, checking at each if the sign of curvature has changed (an inflection point) or if the vertex is a point of locally maximal curvature (a bend point). We also apply thresholding and Gaussian filtering techniques to minimize error on noisy or poorly sampled polylines, as detailed in Algorithm 1. Gaussian filtering is a method of smoothing curves often used to assist in analyzing noisy curves; Lowe [7] gives further details and an efficient algorithm.

Algorithm 1 requires $O(|f| + n')$ time and space, where $|f|$ is the number of vertices of the polyline f and n' is the number of sample points. All input parameters are user-defined. Their values influence the number of characteristic points that will be detected.

Algorithm 1 Characteristic point detection

```
Input: Polyline $f$, number of sample points $n'$, Gaussian smoothing factor $\sigma$, threshold angles $\theta_i, \theta_b$ and $\theta_c$.
Output: Set of characteristic points $C$.
1: Resample $f$ using $n'$ equally-spaced points to create a new polyline $f'$.
2: Apply a Gaussian filter (factor $\sigma$) to smooth $f'$.
3: Mark inflection vertices with inflection angle $\geq \theta_i$.
4: Mark bend vertices with bend angle between adjacent edges $\geq \theta_b$ and change in curvature $\geq \theta_c$ from last point of locally minimal curvature.
5: Mark first and last vertices.
6: Proceed through the smoothed polyline $f'$ and store the distance of each marked vertex from the start of $f'$ as a percentage of the length of $f'$.
7: Return set $C$ of points at the stored percentage distances along the original polyline $f$.
```

3.2 Finding an optimum correspondence

We detect the characteristic points of f and g independently of each other. Assume that there are $n+1$ such points on f and $m+1$ points on g, which divide the polylines into two sequences of subpolylines $(f_1, \ldots, f_n)$ and $(g_1, \ldots, g_m)$. Next, we approach the correspondence problem. Basically, there are five possibilities to match a subpolyline f_i:

(a) f_i is mapped to the last characteristic point g_j^{last} of a subpolyline g_j (i.e., f_i disappears),

(b) a subpolyline g_j is mapped to the last point f_i^{last} of f_i (i.e., g_j disappears),

(c) f_i is mapped to a subpolyline g_j,

(d) f_i is mapped to a merged polyline $g_{j\ldots(j+k)}$, and

(e) f_i is part of a merged polyline $f_{\ell\ldots i\ldots(\ell+k)}$ that is mapped to a subpolyline g_j.

Clearly, the linear order of the subpolylines along f and g has to be respected by the assignment.

Now assume that there is a morphing cost δ associated with the morph between two polylines. We suggest a morphing distance in the next section, but Algorithm 2 is independent of the concrete distance. It is based on dynamic programming and computes a minimum-cost correspondence. Algorithm 2 recursively fills an $n \times m$ table T, where the entry $T[i,j]$ stores the minimal cost of morphing $f_{1\ldots i}$ to $g_{1\ldots j}$. Consequently, we can obtain the optimum correspondence from $T[n,m]$.

The required storage space and running time of Algorithm 2 is $O(nm)$ provided that the *look-back parameter K* is constant. Otherwise the running time

Algorithm 2 Optimum correspondence

```
Input: Polylines $f = (f_1, \dots, f_n)$, $g = (g_1, \dots, g_m)$;
    distance matrix $\delta$.
Output: Optimum correspondence for $f$ and $g$.
1: $T[0,0] = 0$
2: $T[0,j] = T[0,j-1] + \delta(f_1^{\text{first}}, g_j),\ j = 1 \dots m$
3: $T[i,0] = T[i-1,0] + \delta(f_i, g_1^{\text{first}}),\ i = 1 \dots n$
4: for $i = 1$ to $n$ do
5:     for $j = 1$ to $m$ do
6:         $T[i,j] = \min \begin{cases} T[i-1,j] + \delta(f_i, g_j^{\text{last}}) \\ T[i,j-1] + \delta(f_i^{\text{last}}, g_j) \\ T[i-1,j-1] + \delta(f_i, g_j) \\ T[i-1,j-k] + \delta(f_i, g_{(j-k+1)\dots j}), \\ \quad k = 2, \dots, K \\ T[i-k,j-1] + \delta(f_{(i-k+1)\dots i}, g_j), \\ \quad k = 2, \dots, K \end{cases}$
7:         Store pointer to predecessor, i.e., to the table
           entry that yielded the minimum.
8: Generate optimum correspondence from $T[n,m]$
   using backtracking along pointers.
```

increases to $O(nm(n+m))$. The parameter K determines the maximum number of subpolyline segments that can be merged in order to match them with another segment in cases (d) and (e).

Distance measure. Algorithm 2 relies on a distance function δ that represents the morphing cost of a pair of polylines. Distance functions for polylines can be defined in many ways, e.g., *morphing width* [2] and *Fréchet distance* [1].

We define a new distance measure that takes into account how far *all* points move during the morphing by integrating over the trajectory lengths. Assume that two subpolylines f_i and g_j with uniform parameterizations α and β are given. Each point $\alpha(u)$ on f_i will move to $\beta(u)$ on g_j along the connecting segment of length $||\alpha(u) - \beta(u)||$. Then the morphing distance is defined as

$$\delta(f_i, g_j) = \int_0^1 ||\alpha(u) - \beta(u)|| du \qquad (1)$$

and can be computed in time linear in the complexity of f_i and g_j.

Optionally, we can add further terms to the base distance δ. Adding the length difference of f_i and g_j, or alternatively the length of the polyline $\gamma(u) := \alpha(u) - \beta(u)$ favors pairs of polylines that are roughly the same length or orientation. We can also multiply δ by the ratio of the subpolylines' length with the total length of the containing polylines f and g, to account for their relative visual importance.

Finally, we wish to avoid self-intersections in the morph. We do this locally by setting the effective morphing distance to ∞ if matching two subpolylines causes a self-intersection in the morph between them. However, in rare cases intersections between two non-corresponding subpolylines may still occur.

4 Results

We ran our implementations on a small set of French roads from the BD Carto® and the TOP100 series maps produced by the IGN Carto2001 project [6]. For each road, we used a polyline from BD Carto® at scale 1:50,000, and a generalized version at scale 1:100,000 from the Carto2001 TOP100 maps. Figures 1(a) and 1(b) show one example of a road in the dataset, at the two respective scales. The characteristic points that Algorithm 1 detected are marked by little squares. Currently, the parameters used to obtain these results were set by trial and error; so far we have no automatic process to pick reasonable values.

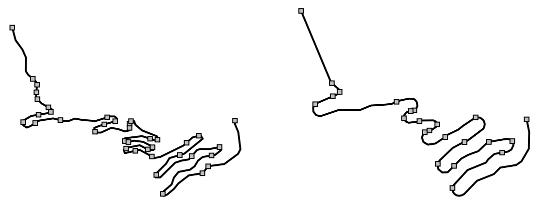

(a) Road 1 (1:50,000) (b) Road 1 (1:100,000)

Figure 1: Example roads at two scales with detected characteristic points marked.

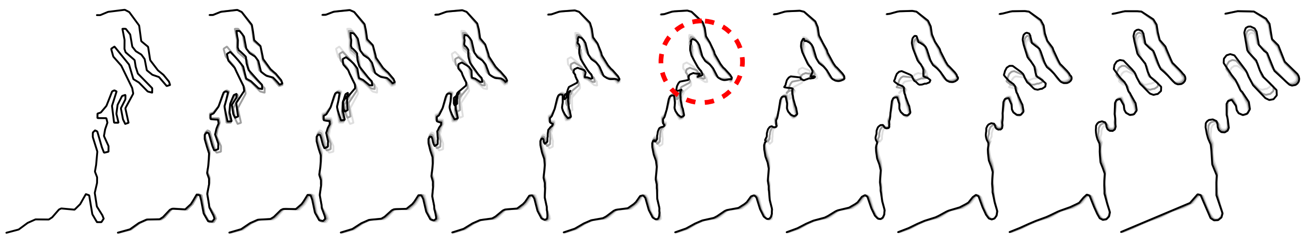
A sequence of snapshots[1] of the final morph, after applying Algorithm 2, is shown in Figure 2(b). A look-back parameter K of 5 was used. For the purpose of comparison, Figure 2(a) shows a simple linear morphing between the same polylines, where both polylines were uniformly parameterized to establish the correspondence between points. On a 3.0GHz Pentium 4 with 1GB RAM, the entire processing time was under 3 seconds.

The optimum-correspondence morphing shows some clear improvements over the naïve linear morphing. The linear morphing in Figure 2(a) shows one of the large serpentine sections at the top being flipped "inside-out" during the morph. In contrast, the optimum-correspondence morphing in Figure 2(b) simply expands the bends. It is evident that the total movement overall is much higher for the linear morphing than for the optimum matching morphing.

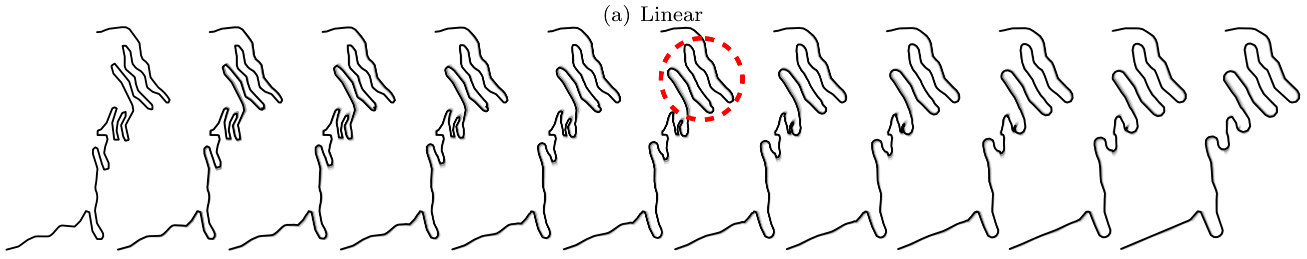
5 Concluding remarks

The algorithms in this paper should be improved in two ways. Ensuring that self-intersections do not

[1]The full animation and an additional example are available at http://i11www.iti.uni-karlsruhe.de/morphingmovies



(a) Linear



(b) OptCor

Figure 2: A comparison between simple linear morphing and the optimum-correspondence morphing (OptCor). In each snapshot, the previous two frames are drawn in successively lighter shades of grey. Areas of particular interest are marked with dashed circles.

occur during a morph could potentially be accomplished by utilizing the algorithm of Surazhsky and Gotsman [10] to compute non-linear trajectories for points. Also, the detection of appropriate characteristic points with little or no user interaction requires further investigation.

Acknowledgements The authors thank Sébastien Mustière for providing the Carto2001 data.

References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comp. Geom. & Appl.*, 5(1–2):75–91, 1995.

[2] S. Bespamyatnikh. An optimal morphing between polylines. *Int. J. Comp. Geom. & Appl.*, 12(3):217–228, 2002.

[3] A. Cecconi and M. Galanda. Adaptive zooming in Web cartography. *Computer Graphics Forum*, 21(4):787–799, 2002.

[4] S. Cohen, G. Elber, and R. Bar-Yehuda. Matching of freeform curves. *Computer-Aided Design*, 29(5):369–378, 1997.

[5] C. Erten, S. G. Kobourov, and C. Pitta. Intersection-free morphing of planar graphs. In *Proc. 11th Int. Sympos. Graph Drawing (GD'03)*, volume 2912 of *Lecture Notes in Computer Science*, pages 320–331. Springer Verlag, 2004.

[6] F. Lecordix, Y. Jahard, C. Lemarié, and E. Hauboin. The end of Carto 2001 project: Top100 based on bdcarto database. In *Proc. 8th ICA Workshop on Generalisation and Multiple Representation*, A Coruña, Spain, July 2005.

[7] D. Lowe. Organization of smooth image curves at multiple scales. *International Journal of Computer Vision*, 3(2):119–130, 1989.

[8] C. Plazanet, J.-G. Affholder, and E. Fritsch. The importance of geometric modeling in linear feature generalization. *Cartography and Geographic Information Systems*, 22(4):291–305, 1995.

[9] T. Samoilov and G. Elber. Self-intersection elimination in metamorphosis of two-dimensional curves. *The Visual Computer*, 14:415–428, 1998.

[10] V. Surazhsky and C. Gotsman. Morphing stick figures using optimized compatible triangulations. In *Proc. Ninth Pacific Conf. on Comp. Graph. and App. (PG'01)*, pages 40–49, 2001.

[11] M. van Kreveld. Smooth generalization for continuous zooming. In *Proc. 20th Int. Cartographic Conf. (ICC'01)*, pages 2180–2185, 2001.

[12] R. Weibel and G. Dutton. Generalising spatial data and dealing with multiple representations. In P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind, editors, *Geographical Information Systems – Principles and Technical Issues*, volume 1, chapter 10, pages 125–155. John Wiley & Sons, 1999.

Deflating The Pentagon

Erik D. Demaine* Martin L. Demaine* Diane L. Souvaine† Perouz Taslakian‡

Abstract

In this paper we consider deflations (inverse pocket flips) of quadrilaterals and pentagons. We characterize infinitely deflatable quadrilaterals by proving necessity of previously obtained sufficient conditions. Then we show that every pentagon can be deflated after finitely many deflations, and that any infinite deflation sequence of a pentagon results from deflating an induced quadrilateral on four of the vertices.

1 Introduction

A *deflation* of a simple planar polygon is the operation of reflecting a subchain of the polygon through the line connecting its endpoints such that (1) the line intersects the polygon only at those two polygon vertices, (2) the resulting polygon is simple (does not self-intersect), and (3) the reflected subchain lies inside the hull of the resulting polygon. A polygon is *deflated* if it does not admit any deflations, i.e., every pair of polygon vertices either defines a line intersecting the polygon elsewhere or results in a nonsimple polygon after reflection.

Deflation is the inverse operation of pocket flipping. Given a nonconvex simple planar polygon, a *pocket* is a maximal connected region exterior to the polygon and interior to its convex hull. Such a pocket is bounded by one edge of the convex hull of the polygon, called the *pocket lid*, and a subchain of the polygon, called the *pocket subchain*. A *pocket flip* (or simply *flip*) is the operation of reflecting the pocket subchain through the line extending the pocket lid. The result is a new, simple polygon of larger area with the same edge lengths as the original polygon. A convex polygon has no pocket and hence does not admit a flip.

In 1935, Erdős conjectured that every nonconvex polygon convexifies after a finite number of flips [3]. Four years later, Nagy [1] claimed a proof of Erdős's conjecture. Recently, Demaine et al. [2] uncovered a flaw in Nagy's argument, as well as other claimed proofs, but fortunately correct proofs remain.

In the same spirit of finite flips, Wegner conjectured in 1993 that any polygon becomes deflated after a finite number of deflations [6]. Eight years later, Fevens et al. [4] disproved Wegner's conjecture by demonstrating a family of quadrilaterals that admit an infinite number of deflations. They left an open problem of characterizing which polygons deflate infinitely.

In this paper, we show that the family of quadrilaterals described in [4] are the only polygons with four sides that admit infinitely many deflations, thus characterizing infinitely deflatable quadrilaterals. We also show that any such quadrilateral flattens in the limit. We use this characterization of infinitely deflating quadrilaterals to understand deflations of pentagons. Specifically, we show that every pentagon admitting an infinite number of deflations induces an infinitely deflatable quadrilateral on four of its vertices. Then we show our main result: every pentagon can be deflated after finitely many (well-chosen) deflations.

2 Definitions and Notations

Let $P = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ be a polygon together with a clockwise ordering of its vertices. Let $P^k = \langle v_0^k, v_1^k, \ldots, v_{n-1}^k \rangle$ denote the polygon after k arbitrary deflations, and P^* denote the limit of P^k, when it exists, having vertices v_i^*. Thus, the initial polygon $P = P^0$. The *turn angle* of a vertex v_i is the signed angle $\theta \in (-180°, 180°]$ between the two vectors $v_i - v_{i-1}$ and $v_i - v_{i+1}$. A vertex of a polygon is *flat* if the angle between its incident edges is 180°, i.e., forming a turn angle of 0°. A *flat polygon* is a polygon with all its vertices collinear. A *hairpin* vertex v_i is a vertex whose incident edges overlap each other, i.e., forming a turn angle of 180°. A polygon vertex is *sharpened* when its absolute turn angle decreases.

3 Deflation in General

In this section, we prove general properties about deflation in arbitrary simple polygons. Our first few lemmata are fairly straightforward, while the last lemma is quite intricate and central to our later arguments.

Lemma 1 *Deflation only sharpens angles.*

This result follows from an analogous result for pocket flips, which only flatten angles (see, e.g., [5]).

Corollary 2 *Any n-gon with no flat vertices will continue to have no flat vertices after deflation, even in an accumulation point.*

*MIT, {edemaine,mdemaine}@mit.edu
†Tufts University, dls@cs.tufts.edu
‡McGill University, perouz@cs.mcgill.ca

Lemma 3 *In any infinite deflation sequence $P^0, P^1, P^2, \ldots,$ the absolute turn angle $|\tau_i|$ at any vertex v_i has a (unique) limit $|\tau_i^*|$.*

Corollary 4 *In any infinite deflation sequence $P^0, P^1, P^2, \ldots,$ v_i^* is a hairpin vertex in some accumulation point P^* if and only if v_i^* is a hairpin vertex in all accumulation points P^*.*

Lemma 5 *Any n-gon with n odd and no flat vertices cannot flatten in an accumulation point of an infinite deflation sequence.*

Lemma 6 *For any infinite deflation sequence $P^0, P^1, P^2, \ldots,$ there is a subchain $v_i, v_{i+1}, \ldots, v_j$ (where $j - i \geq 2$) that is the pocket chain of infinitely many deflations.*

We conclude this section with a challenging lemma showing that infinitely deflating pockets flatten:

Lemma 7 *Assume $P = P^0$ has no flat vertices. If P^* is an accumulation point of the infinite deflation sequence $P^0, P^1, P^2, \ldots,$ and subchain $v_i, v_{i+1}, \ldots, v_j$ (where $j - i \geq 2$) is the pocket chain of infinitely many deflations, then $v_i^*, v_{i+1}^*, \ldots, v_j^*$ are collinear and $v_{i+1}^*, \ldots, v_{j-1}^*$ are hairpin vertices. Furthermore, if $v_{i+1}^*, \ldots, v_{j-1}^*$ extends beyond v_j^*, then v_j^* is a hairpin vertex; and if $v_{i+1}^*, \ldots, v_{j-1}^*$ extends beyond v_i^*, then v_i^* is a hairpin vertex. In particular, if $j - i = 2$, then either v_i^* or v_j^* is a hairpin vertex.*

Proof. Because $P^0 \supseteq P^1 \supseteq P^2 \supseteq \cdots$, we have $\text{hull}(P^0) \supseteq \text{hull}(P^1) \supseteq \text{hull}(P^2) \supseteq \cdots$, and in particular $\text{area}(\text{hull}(P^0)) \geq \text{area}(\text{hull}(P^1)) \geq \text{area}(\text{hull}(P^2)) \geq \cdots \geq 0$. Thus, $\sum_{t=1}^{\infty}[\text{area}(\text{hull}(P^t)) - \text{area}(\text{hull}(P^{t-1}))] \leq \text{area}(\text{hull}(P^0))$, so $\text{area}(\text{hull}(P^t)) - \text{area}(\text{hull}(P^{t-1})) \to 0$ as $t \to \infty$. Hence, for any $\varepsilon > 0$, there is a time T_ε such that, for all $t \geq T_\varepsilon$, $\text{area}(\text{hull}(P^t)) - \text{area}(\text{hull}(P^{t-1})) \leq \varepsilon$. As a consequence, for all $t \geq T_\varepsilon$, $\text{hull}(P^{t-1}) \subseteq \text{hull}(P^t) \oplus D_{\varepsilon/\ell}$ where $\oplus$ denotes Minkowski sum, D_x denotes a disk of radius x, and ℓ is the length of the longest edge in P, which is a lower bound on the perimeter of $\text{hull}(P^t)$.

Let $t_1, t_2, \ldots$ denote the infinite subsequence of deflations that use $v_i, v_{i+1}, \ldots, v_j$ as the pocket subchain, where P^{t_r} is the polygon immediately after the rth deflation of the pocket chain $v_i, v_{i+1}, \ldots, v_j$. Consider any vertex v_k with $i < k < j$. If $t_r \geq T_\varepsilon$, then $v_k^{t_r-1} \in \text{hull}(P^{t_r}) \oplus D_{\varepsilon/\ell}$. Also, $v_k^{t_r-1}$ is in the halfplane H_r exterior to the line of support of P^{t_r} through $v_i^{t_r}$ and $v_j^{t_r}$. Now, the region $(\text{hull}(P^{t_r}) \oplus D_{\varepsilon/\ell}) \cap H_r$ converges to a subset of the line $\ell_{i,j}^{t_r}$ through $v_i^{t_r}$ and $v_j^{t_r}$ as $\varepsilon \to 0$ while keeping $t_r \geq T_\varepsilon$. Thus, for any accumulation point P^*, v_k^* is collinear with v_i^* and v_j^*, for all $i < k < j$. In other words, $v_{i+1}^*, \ldots, v_{j-1}^*$ lie on the line $\ell_{i,j}^*$ through v_i^* and v_j^*. By Corollary 2, $v_{i+1}^*, \ldots, v_{j-1}^*$ are not flat, so they must be hairpins.

By Lemma 3, the absolute turn angle $|\tau_j|$ of vertex v_j has a limit $|\tau_j^*|$. If $|\tau_j^*| > 0$ (i.e., v_j^* is not a hairpin in all limit points P^*), then by Lemma 1, $|\tau_j^t| \geq |\tau_j^*| > 0$. For sufficiently large t, v_{j-1}^t approaches the line $\ell_{i,j}^t$. To form the absolute turn angle $|\tau_j^t| \geq |\tau_j^*| > 0$ at v_j, v_{j+1}^t must eventually be bounded away from the line $\ell_{i,j}^t$: after some time T, the minimum of the two angles between $v_j^t v_{j+1}^t$ and $\ell_{i,j}^t$ must be bounded below by some $\alpha > 0$. Now suppose that some $v_k^{t_r-1}$ were to extend beyond $v_j^{t_r-1}$ in the projection onto the line $\ell_{i,j}^{t_r-1}$ for some $t_r - 1 > T$. As illustrated in Figure 1, for the deflation of the chain $v_i^{t_r-1}, v_{i+1}^{t_r-1}, \ldots, v_j^{t_r-1}$ to not cause the next polygon P^{t_r} to self-intersect, the minimum of the two angles between $v_j^{t_r-1} v_k^{t_r-1}$ and $\ell_{i,j}^{t_r-1}$ must also be at least α. See Figure 1.

But this is impossible for sufficiently large t, because v_k^t accumulates on the line $\ell_{i,j}^t$. Hence, in fact, v_k^t must not extend beyond v_j^t in the $\ell_{i,j}^t$ projection for sufficiently large t. In other words, when v_j^* is not a hairpin, each v_k^* must not extend beyond v_j^* on the line $\ell_{i,j}^*$. A symmetric argument handles the case when v_i^* is not a hairpin.

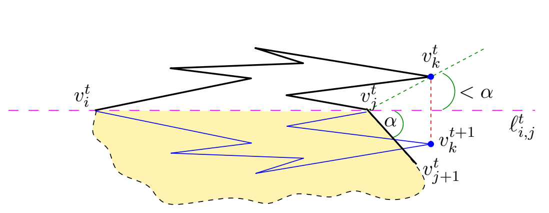
Finally, suppose that $j - i = 2$. In this case, because $v_{i+1}^* = v_{j-1}^*$ is a hairpin, it must extend beyond one of its neighbors, v_i^* or v_j^*. By the argument above, in the first case, v_i^* must be a hairpin, and in the second case, v_j^* must be a hairpin. Thus, as desired, either v_i^* or v_j^* must be a hairpin. □

4 Deflating Quadrilaterals

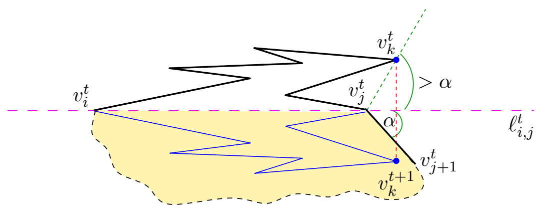
Lemma 8 *Any accumulation point of an infinite deflation sequence of a quadrilateral is flat and has no flat vertices.*

Proof. First we argue that all quadrilaterals $P^1, P^2, \ldots$ (excluding the initial quadrilateral P^0) have no flat vertices. Because deflations are the inverse of pocket flips, and pocket flips do not exist for convex polygons, deflation always results in a nonconvex polygon. Thus all quadrilaterals P^t with $t > 0$ must be nonconvex. Hence no P^t with $t > 0$ can have a flat vertex, because then it would lie along an edge of the triangle of the other three vertices, making the quadrilateral convex. By Corollary 2, there are also no flat vertices in any accumulation point P^*.

By Lemma 6, there is a subchain $v_i, v_{i+1}, \ldots, v_j$, where $j - i \geq 2$, that is the pocket chain of infinitely many deflations. In fact, $j - i$ must equal 2, because reflecting a longer (4-vertex) pocket chain would not change the polygon. Applying Lemma 7 to $P^1, P^2, \ldots$ (with no flat vertices), for any accumulation point P^*, v_{i+1}^* is a hairpin and either v_i^* or $v_j^* = v_{i+2}^*$ is a



(a) The angle between $v_k^t v_j^t$ and $\ell_{i,j}^t$ is less than α, hence in the next deflation step the chain $v_i^t \ldots v_j^t$ will intersect the polygon.



(b) The angle between $v_k^t v_j^t$ and $\ell_{i,j}^t$ is greater than α, so the polygon will not self-intersect in the next deflation step.

Figure 1: Because v_j^t is not a hairpin, the minimum angle α between $v_j^t v_{j+1}^t$ and $\ell_{i,j}^t$ is strictly positive. If any vertex v_k^t of the chain $v_i^t, v_{i+1}^t, \ldots, v_j^t$ extends beyond v_j^t, then the minimum angle between $v_k^t v_j^t$ and $\ell_{i,j}^t$ must be at least α for the polygon P^{t+1} not to self-intersect in the next deflation step. The dotted curve represents the rest of the polygon chain and the shaded area is the interior of the polygon below line $\ell_{i,j}^t$.

hairpin. Hairpin v_{i+1}^* implies that v_i^*, v_{i+1}^*, v_{i+2}^* are collinear, while hairpin v_i^* or v_{i+2}^* implies that the remaining vertex $v_{i+3}^* = v_{i-1}^*$ lie on that same line. Therefore, any accumulation point P^* is flat. □

Combining the flattening property of Lemma 8 with the previous necessary conditions of Fevens et al. [4], we obtain a complete characterization of infinitely deflating quadrilaterals:

Theorem 9 *A quadrilateral with side lengths $\ell_1, \ell_2, \ell_3, \ell_4$ is infinitely deflatable if and only if*

1. *opposite edges sum equally, i.e., $\ell_1 + \ell_3 = \ell_2 + \ell_4$; and*
2. *adjacent edges differ, i.e., $\ell_1 \neq \ell_2 \neq \ell_3 \neq \ell_4 \neq \ell_1$.*

Furthermore, every such infinitely deflatable quadrilateral deflates infinitely independent of the choice of deflation sequence.

Proof. Fevens et al. [4] proved that every quadrilateral satisfying the two conditions on its edge lengths are infinitely deflatable, no matter which deflation sequence we make. Thus the two conditions are sufficient for infinite deflation.

To see that the first condition is necessary, we use Lemma 8. Because deflation preserves edge lengths, so do accumulation points of an infinite deflation sequence, so the flat limit configuration from Lemma 8 is a flat configuration of the edge lengths $\ell_1, \ell_2, \ell_3, \ell_4$. By a suitable rotation, we may arrange that the flat configuration lies along the x axis. By Lemma 8, no vertex is flat, so every vertex must be a hairpin. Thus, during a traversal of the polygon boundary, the edges alternate between going left ℓ_i and going right ℓ_i. At the end of the traversal, we must end up where we started. Therefore, $\pm(\ell_1 - \ell_2 + \ell_3 - \ell_4) = 0$, i.e., $\ell_1 + \ell_3 = \ell_2 + \ell_4$.

To see that the second condition is necessary, suppose for contradiction that $\ell_1 = \ell_2$ (the other contrary cases are symmetric). By the first condition, $\ell_1 + \ell_3 = \ell_1 + \ell_4$, so $\ell_3 = \ell_4$. Thus, the polygon is a kite, having two pairs of adjacent equal sides. (Also, all four sides might be equal.) Every kite has a chord that is a line of reflectional symmetry. No kite can deflate along this line, because such a deflation would cause edges to overlap with their reflections. If a kite is convex, it may deflate along its other chord, but then it becomes nonconvex, so it can be deflated only along its line of reflectional symmetry. Therefore, a kite can be deflated at most once, so any infinitely deflatable quadrilateral must have $\ell_1 \neq \ell_2$ and symmetrically $\ell_1 \neq \ell_2 \neq \ell_3 \neq \ell_4 \neq \ell_1$. □

5 Deflating Pentagons

Theorem 10 *There is a pentagon with a flat vertex that deflates infinitely for all deflation sequences, exactly like the quadrilateral on the nonflat vertices.*

Proof. See Figure 2. We start with an infinitely deflating quadrilateral $\langle v_0, v_1, v_2, v_3, v_4 \rangle$ according to Theorem 9, and add a flat vertex v_5 along the edge $v_4 v_0$. As long as we never deflate along a line passing through the flat vertex v_4, the deflations act exactly like the quadrilateral, and thus continue infinitely no matter which deflation sequence we choose. To achieve this property, we set the length of segment $v_3 v_0$ to 1, with v_4 at the midpoint; we set the lengths of edges $v_0 v_1$ and $v_2 v_3$ to 2/3; and we set the length of edge $v_1 v_2$ to 1/3. Then we deflate the quadrilateral until the vertices are close to being hairpins that v_4 cannot see the nonadjacent convex vertex and the line through v_4 and the reflex vertex intersects the pentagon at another point. Thus no line of deflation passes through v_4, so we maintain infinite deflation as in the induced quadrilateral. □

Finally we show that any infinitely deflating pentagon induces an infinitely deflating quadrilateral.

Theorem 11 *Every pentagon with no flat vertices is finitely deflatable.*

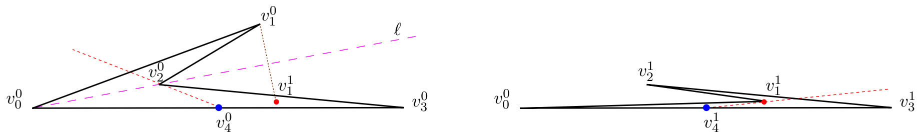


Figure 2: An infinitely deflatable pentagon that induces an infinitely deflatable quadrilateral (left) and its configuration after the first deflation (right).

Proof. Let P be a pentagon with no flat vertices, and assume for the sake of contradiction that P deflates infinitely. Consider any accumulation point P^* of an infinite deflation sequence $P^0, P^1, P^2, \ldots$. By Lemma 6, there is an infinitely deflating pocket chain, say $v_0, v_1, \ldots, v_j$, where $j \geq 2$. By Lemma 7, $v_1^*, \ldots, v_{j-1}^*$ are hairpin vertices. Because the pentagon has only five vertices, $j \leq 4$. In fact, $j \leq 3$: if $j = 4$, this pocket chain would encompass all five vertices, making P^* collinear, which contradicts Lemma 5. If $j = 3$, then v_1^* and v_2^* are hairpins. If $j = 2$, then by Lemma 7, either v_0^* or v_2^* must be a hairpin; suppose by symmetry that it is v_2^*. Thus, in this case, again v_1^* and v_2^* are hairpins. Hence, in all cases, v_1^* and v_2^* are hairpins, so $v_0^*, v_1^*, v_2^*, v_3^*$ are collinear, while by Lemma 5, v_4^* must be off this line.

By Lemma 7, an infinitely flipping pocket chain requires at least one hairpin vertex. Thus, the only possible infinitely flipping pocket chains are v_0, v_1, v_2; v_1, v_2, v_3; and v_0, v_1, v_2, v_3. Let T denote the time after which only these chains flip. Thus, after time T, v_0, v_3, v_4 stop moving, so in particular, v_4's angle and the length of the edge v_0v_3 take on their final values. Therefore, after time T, the vertices v_0, v_1, v_2, v_3 induce a quadrilateral that deflates infinitely.

Because $v_0^*, v_1^*, v_2^*, v_3^*$ are collinear and v_4^* is off this line, neither v_0^* nor v_3^* can be hairpins. By Lemma 7, v_1^* and v_2^* must lie along the segment $v_0^*v_3^*$. By Theorem 9, no two adjacent edges of the quadrilateral have the same length, so in fact v_1^* and v_2^* must be strictly interior to the segment $v_0^*v_3^*$. For sufficiently large t, $v_0^t, v_1^t, v_2^t, v_3^t$ are arbitrarily close to collinear, and v_1^t and v_2^t project strictly interior to the line segment $v_0^tv_3^t$. As a consequence, for sufficiently large t, we can deflate the chain $v_0^t, v_1^t, v_2^t, v_3^t$ along the line through v_0^t and v_3^t into the triangle $v_0^tv_3^tv_4^t$. But then the convex hull of P^{t+1} is $v_0^{t+1}v_3^{t+1}v_4^{t+1}$, which is fixed, so no further deflations are possible, resulting in a finite deflation sequence. $\square$

6 Larger Polygons and Open Problems

It is easy to construct n-gons with $n \geq 6$ that deflate infinitely, no matter which deflation sequence we choose. See Figure 3 for the idea of the construction. We can add any number of spikes to obtain n-gons with $n \geq 6$ and even. For n odd, we can shave off the tip of one of the spikes. Thus, $n = 5$ is the only value for which every n-gon with no flat vertices can be finitely deflated.

None of the infinitely deflating polygons of Figure 3 are particularly satisfying because their accumulation points are not flat. Are there any n-gons, $n > 4$, that have no flat vertices and always deflate infinitely to flat accumulation points? In an unpublished manuscript (2004), Fevens et al. show a family of infinitely deflating hexagons with no flat vertices that flatten in the limit.

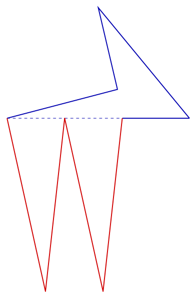


Figure 3: An infinitely deflating octagon constructed by adding long spikes to an infinitely deflating quadrilateral.

Does every infinite deflation sequence have a (unique) limit? Our proofs would likely simplify if we knew there was only one accumulation point.

Is there an efficient algorithm to determine whether a given polygon P has an infinite deflation sequence? What about detecting whether all deflation sequences are infinite? Even given a (succinctly encoded) infinite sequence of deflations, can we efficiently determine whether the sequence is valid, i.e., whether it avoids self-intersection?

References

[1] Béla de Sz. Nagy. Solution of problem 3763. *American Mathematical Monthly*, 46:176–177, 1939.

[2] Erik D. Demaine, Blaise Gassend, Joseph O'Rourke, and Godfried T. Toussaint. Polygons flip finitely: Flaws and a fix. In *Proc. of the 18th Canadian Conference in Comput. Geometry*, pp. 109–112, Aug 2006.

[3] Paul Erdős. Problem 3763. *American Mathematical Monthly*, 42:627, 1935.

[4] Thomas Fevens, Antonio Hernandez, Antonio Mesa, Patrick Morin, Michael Soss, and Godfried Toussaint. Simple polygons with an infinite sequence of deflations. *Contrib. to Algebra and Geom.*, 42(2):307–311, 2001.

[5] Godfried T. Toussaint. The Erdős-Nagy theorem and its ramifications. *Computational Geometry: Theory and Applications*, 31:219–236, 2005.

[6] Bernd Wagner. Partial inflations of closed polygons in the plane. *Contributions to Algebra and Geometry*, 34(1):77–85, 1993.

Wrapping the Mozartkugel

Erik D. Demaine* Martin L. Demaine* John Iacono† Stefan Langerman‡

Abstract

We study wrappings of the unit sphere by a piece of paper (or, perhaps more accurately, a piece of foil). Such wrappings differ from standard origami because they require infinitely many infinitesimally small "folds" in order to transform the flat sheet into a positive-curvature sphere. Our goal is to find shapes that have small area even when expanded to shapes that tile the plane. We characterize the smallest square that wraps the unit sphere, show that a 0.1% smaller equilateral triangle suffices, and find a 20% smaller shape that still tiles the plane.

Keywords: chocolate, marzipan, praline, nougat

1 Introduction

The Mozartkugel ("Mozart sphere") [9, 8] is a famous fine Austrian confectionery: a sphere with marzipan in its core, encased in nougat or praline cream, and coated with dark chocolate. It was invented in 1890 by Paul Fürst in Salzburg (where Wolfgang Amadeus Mozart was born), six years after he founded his confectionery company, Fürst. Fürst (the company) still to this day makes Mozartkugeln by hand, about 1.4 million per year, under the name "Original Salzburger Mozartkugel" [6]. At the 1905 Paris Exhibition, Paul Fürst received a gold medal for the Mozartkugel.

Many other companies now make similar Mozartkugeln, but Mirabell is the market leader with their "Echte (Genuine) Salzburger Mozartkugeln" [7]. Over 1.5 billion have been made, about 90 million per year, originally by hand but now by industrial methods, and Mirabell claims their product to be the only Mozartkugel that is perfectly spherical. They are also the only Mozartkugel to be taken into outer space, by the first Austrian astronaut Franz Viehböck as a gift to the Russian cosmonauts on the MIR space station. Despite industrial techniques, each Mozartkugel still takes about 2.5 hours to make.

Although most of a Mozartkugel is edible, each sphere is individually wrapped in a square of aluminum foil. To minimize the amount of this wasted, inedible material, it is natural to study the smallest piece of foil that can wrap a unit sphere. Because the pieces will be cut from a large sheet of foil, we would also like the unfolded shape to tile the plane.

We formalize this practical problem in the next section; the main difficulty is to allow a continuum of infinitesimal folds to curve the paper, a feature not normally modeled by mathematical origami. We then study wrappings by squares and equilateral triangles, and show that the latter leads to a small (0.1%) savings, which may prove significant on the many millions of Mozartkugel consumed each year. Even better, if we allow wrapping by arbitrary shapes that tile the plane, we show how to achieve a 20% savings. In addition to direct savings in material costs for Mozartkugel manufacturers, the reduced material usage also indirectly cuts down on CO_2 emissions, and therefore partially solves the global-warming problem and consequently the little-reported but equally important chocolate-melting problem.

2 Wrapping Problem

In standard mathematical origami [4, 5], a *piece of paper* is a two-dimensional manifold (usually flat), and a *folding* is an isometric mapping of this piece of paper into Euclidean 3-space. Here *isometric* means that distances are preserved, as measured by shortest paths on the piece of paper before and after mapping via the folding.

But there is no isometric folding of a square into a sphere: isometric folding preserves curvature. Therefore we define a new, less restrictive type of folding that allows changing curvature but still prevents stretching of the material. Namely, a *wrapping* is a continuous contractive mapping of a piece of paper into Euclidean 3-space. Here *contractive* means that every distance either decreases or stays the same, as measured by shortest paths on the piece of paper before and after mapping via the folding. This definition effectively assumes that the length contraction can be achieved by continuous infinitesimal pleating.

We can model one family of wrappings by expressing which distances are preserved isometrically. An optimal wrapping should be isometric along some path, for otherwise we could uniformly scale the entire wrapping and make a larger object. We call a path *stretched* if the wrapping is isometric along it. A *stretched wrapping* has the property that every point is covered by some stretched path. Such a wrapping can be specified by a set of stretched paths whose

*MIT, {edemaine,mdemaine}@mit.edu

†Polytechnic University, http://john.poly.edu/

‡Chercheur qualifié du FNRS, Université Libre de Bruxelles, stefan.langerman@ulb.ac.be

union covers the entire piece of paper. Although not all such specifications are valid—we need to check that all other paths are contractive—the specification does uniquely determine a wrapping. We specify all of our wrappings in this way, under the belief that stretched wrappings are generally the most efficient.

A special case of stretched wrapping is when the stretched paths consist of the shortest paths from one point x to every other point y. In this case, we are rolling geodesics in the piece of paper onto geodesics of the target surface. This situation corresponds to continuous unfoldings of smooth polyhedra as considered by Benbernou, Cahn, and O'Rourke [1]. Although perhaps the most natural kind of wrapping, this special case is too restrictive for our purposes, as it essentially forces the sphere to be wrapped by a disk of radius π, for those geodesics to reach around to the pole opposite x. We will show how to wrap with far less paper than this disk of area π^3.

Note that, if we start with an arbitrarily long and narrow rectangle, we can wrap the sphere using paper area arbitrarily close to the surface area 4π of the sphere [3]. This wrapping is not very practical, however; in particular, it makes it difficult to make a nondistorted logo on the surface of the sphere.

The only other known optimal wrapping result (where no contraction is necessary) is wrapping a unit cube with a square [2].

3 Petal Wrapping

Our wrappings are based on the following *k-petal wrapping*. On the sphere we first construct k stretched paths $p_1, p_2, \ldots, p_k$ from the south pole to the north pole, dividing the 2π angle around each pole into k equal parts of $2\pi/k$. To each path p_i we assign an "orange peel" with apex angles $2\pi/k$, centered on the path p_i and bounded by the Voronoi diagram of p_{i-1}, p_i, p_{i+1}. These orange peels partition the surface of the sphere into k equal pieces.

Then we construct a continuum of stretched paths to cover each orange peel. Specifically, for every point q along each path p_i, we construct two stretched paths emanating from q, proceeding along geodesics perpendicular to p_i in both directions, and stopping at the boundary of p_i's orange peel.

These stretched paths cover every point of the sphere (covering boundary points twice). It remains to find a suitable piece of paper that wraps according to these stretched paths. The main challenge is to unfold the half of an orange peel left of a path p_i. Then we can easily glue the two halves together along the (straight) unfolded path p_i, and finally join the resulting petals at the unfolded south pole.

To unroll half of a petal, we parameterize as shown in Figure 1. Here $B = \pi/k$ is the half-petal angle; c is a given amount that we traverse along the center path p_i starting at the south-pole endpoint; $A = \pi/2$ specifies that we turn perpendicular from that point; and b is the distance that we travel in that direction. Our goal is to determine b in terms of c.

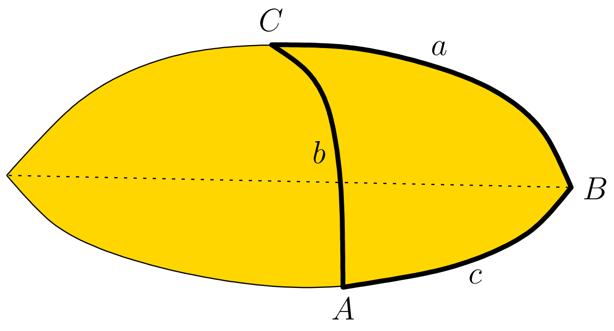


Figure 1: Half of a petal, labeled in preparation for spherical trigonometry.

By the spherical law of cosines,

$$\cos C = -\cos A \cos B + \sin A \sin B \cos c.$$

Now $\cos A = \cos(\pi/2) = 0$ and $\sin A = \sin(\pi/2) = 1$, so this equation simplifies to $\cos C = \sin B \cos c$. Hence, $\sin C = \sqrt{1 - \sin^2 B \cos^2 c}$. By the spherical law of sines,

$$\frac{\sin B}{\sin b} = \frac{\sin C}{\sin c}.$$

Substituting $\sin C = \sqrt{1 - \sin^2 B \cos^2 c}$, we obtain

$$\frac{\sin B}{\sin b} = \frac{\sqrt{1 - \sin^2 B \cos^2 c}}{\sin c},$$

i.e.,

$$\sin b = \frac{\sin B \sin c}{\sqrt{1 - \sin^2 B \cos^2 c}}.$$

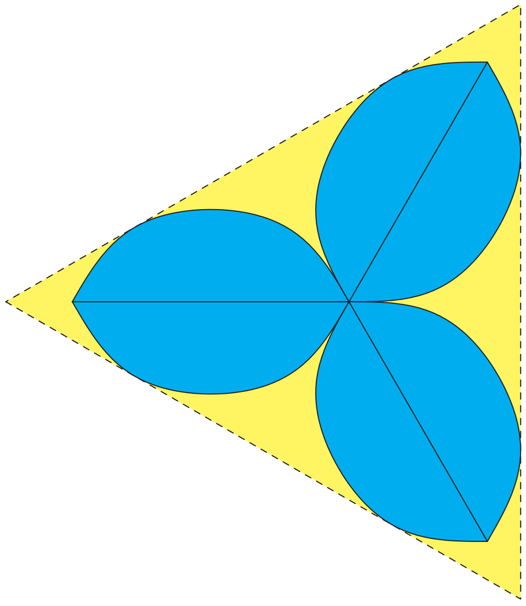
Taking arccos of both sides, we determine the value of b in terms of the parameter c and the known quantity $B = \pi/k$.

Figure 2 shows two examples of the resulting petal unfolding, with $k = 3$ and $k = 4$.

4 Square Wrapping

The angle at the tip of the petals can be computed by taking the derivative $\partial b/\partial c$ at $c = 0$. For $k = 4$, this derivative is 1 which implies a half angle of $\pi/4$. Because the petals are convex, the convex hull of the petal unfolding for $k = 4$ is exactly the square of diagonal 2π. No smaller square could wrap the unit sphere because the length of the path connecting the center of the square to the point mapped to the antipodal point must have length at least π. This square has area $2\pi^2$.

Note that the same area is attainable by a rectangle of dimensions $2\pi \times \pi$: draw one path p around the


(a) $k = 3$

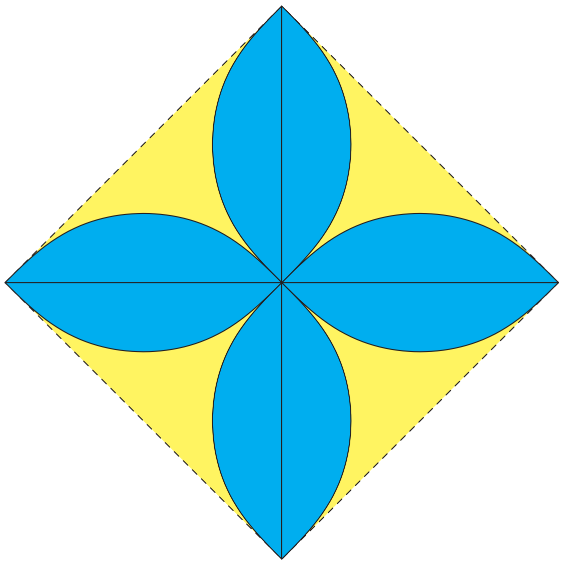

(b) $k = 4$

Figure 2: Petal unfoldings.

equator of the sphere and cover the sphere by a continuum of stretched paths perpendicular to p emanating from every point of p until the north and the south pole of the sphere. The same rectangle is also exactly a 2-petal unfolding. Interestingly, the area of this rectangle wrapping is also 2π. The Echte Salzburger Mozartkugel is wrapped by Mirabell using the same rectangle (expanded a bit to ensure overlap) but with a slightly different folding.

5 Triangle Wrapping

For $k = 3$, the angle at the tip of the petals can be computed similarly to obtain $2\pi/3$, which is natural as the three petals meet at the north pole, their angles summing to 2π. However, the convex hull of the 3-petal unfolding is not a triangle. We compute its smallest enclosing equilateral triangle. The supporting lines of the triangle will be each tangent to two of the petals. The tangent point on the petal can be computed by finding the point (c, b) on its boundary that maximizes the direction $(-\cos(\pi/3), \sin(\pi/3))$. Plugging this into the previous equations, we obtain

$$c = \arccos\left(\frac{\sqrt{57}}{6} - \frac{1}{2}\right) \approx 0.710086.$$

This implies that the supporting line is at a distance

$$\frac{\pi}{2} - \frac{1}{2}\arccos\left(\frac{\sqrt{57}}{6} - \frac{1}{2}\right) + \frac{\sqrt{3}}{2}\arcsin\left(\frac{\sqrt{\sqrt{57}-5}}{\sqrt{\sqrt{57}-3}}\right)$$
$$\approx 0.620190\pi$$

from the center. The area of the inscribing equilateral triangle is therefore $3h^2\tan(\pi/6) \approx 1.998626\,\pi^2$, about 0.1% less than the $2\pi^2$ area of the smallest wrapping square.

6 Tiling

Instead of expanding the petal unfoldings to tilable regular polygons, we can pack the petal unfoldings directly and expand them just to fill the extra space. Figure 3 shows an even better tiling resulting from the 3-petal unfolding. A quick computation shows that only about $1.6\,\pi^2$ area of paper is required for each wrapping, a substantial improvement.

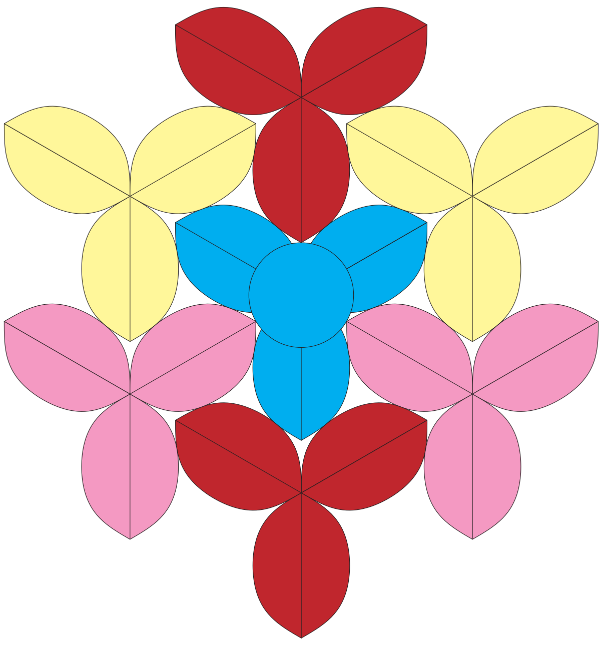
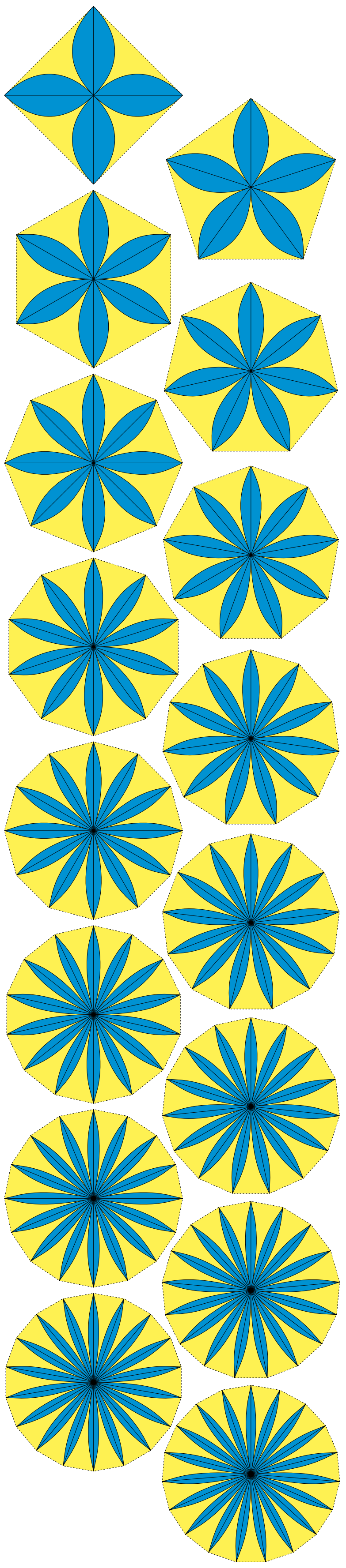


Figure 3: Packing the 3-petal unfolding.

7 Conclusion

This paper initiates a new research direction in the area of *computational confectionery.* We leave as open problems the study of wrapping other geometric confectioneries, or further improving our wrappings of the Mozartkugel. In particular, what is the optimal convex shape that can wrap a unit sphere? What is the optimal shape that also tiles the plane? What about smooth surfaces other than the sphere?



Acknowledgements

The authors thank Luc Devroye, Vida Dujmović, and Dania El-Khechen for helpful discussions, the anonymous referees for their comments, and the Café du Monde for their inspirational beignets.

References

[1] Nadia Benbernou, Patricia Cahn, and Joseph O'Rourke. Unfolding smooth prismatoids. In *14th Annual Fall Workshop on Computational Geometry*, Cambridge, MA, 2004. arXiv:cs.CG/0407063.

[2] Michael L. Catalano-Johnson and Daniel Loeb. Problem 10716: A cubical gift. *American Mathematical Monthly*, 108(1):81–82, January 2001. Posed in volume 106, 1999, page 167.

[3] Erik D. Demaine, Martin L. Demaine, and Joseph S. B. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. *Computational Geometry: Theory and Applications*, 16(1):3–21, 2000.

[4] Erik D. Demaine, Satyan L. Devadoss, Joseph S. B. Mitchell, and Joseph O'Rourke. Continuous foldability of polygonal paper. In *Proceedings of the 16th Canadian Conference on Computational Geometry*, pages 64–67, Montréal, Canada, August 2004.

[5] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, and Polyhedra.* Cambridge University Press, to appear.

[6] Fürst. Original salzburger mozartkugel. http://www.original-mozartkugel.com/.

[7] Mirabell. The brand. http://www.mozartkugel.at/.

[8] Die freie Enzyklopädie Wikipedia. Mozartkugel. http://de.wikipedia.org/wiki/Mozartkugel.

[9] The Free Encyclopedia Wikipedia. Mozartkugel. http://en.wikipedia.org/wiki/Mozartkugel.

Minimum-Dilation Tour is NP-hard

Panos Giannopoulos[*] Christian Knauer[†] Dániel Marx[*]

Abstract

We prove that computing a minimum-dilation (Euclidean) Hamilton circuit or path on a given set of points in the plane is NP-hard.

1 Introduction

Let P be a set of n points in $\mathbb{R}^2$ and G be a geometric network on P, i.e., an undirected graph $G(P, E)$ drawn with straight line edges on the plane, where the weight of an edge $pq \in E$ equals the Euclidean distance $|pq|$. The dilation $\delta_G(p,q)$ of a pair of points p, q in G is defined as $\delta_G(p,q) = d_G(p,q)/|pq|$, where $d_G(p,q)$ is the weight or length of a shortest path from p to q in G. The *vertex-to-vertex dilation* or *stretch factor* $\delta(G)$ of G is defined as

$$\delta(G) = \max_{p,q \in P,\ p \neq q} \delta_G(p,q).$$

For a real number $t \geq 1$, we say that G is a t-spanner for P if $\delta(G) \leq t$.

The cost of a network can be measured by the number of edges, the weight, the diameter, or the maximum degree. Constructing low-cost geometric networks of small dilation, as alternatives to the 'expensive' complete Euclidean graphs, is a problem that has been studied extensively; see the surveys by Eppstein [2] and Smid [7], and the forthcoming book by Narasimhan and Smid [6]. Among other interesting questions, one can ask for the minimum dilation that can be achieved by a network with a given number of edges and other additional properties, and how we compute such a network. We are interested in the complexity of the following problem:

Minimum-dilation tour (path): Given a set P of points in the plane, compute a minimum-dilation Euclidean Hamilton circuit (path) on P.

Related work. Klein and Kutz [5] have recently proved that computing a minimum-dilation geometric network on a point set in the plane, using not more than a given number of edges, is NP-hard, no matter whether edge crossings are allowed or not. Moreover, Cheong et al. [1] showed that the problem remains NP-hard even for the minimum-dilation spanning tree. On the other hand, Eppstein and Wortman [3] gave polynomial time algorithms for the minimum-dilation star problem.

Results. We prove that the minimum-dilation tour (and path) problem is (strongly) NP-hard. The problem requires the use of exactly n ($n-1$, for a path) edges and that every vertex have degree 2 (except for the start and end-point in the case of a path). Note that the proofs of the results by Klein and Kutz and Cheong et al., mentioned above, cannot handle our problem since the former creates graphs with more than one cycle, while the latter works only for trees with no restriction on the maximum degree. Also, both of these results use reductions from SET PARTITION, while we use a different approach and reduce from the HAMILTON CIRCUIT problem on grid graphs [4]. A collorary of our reduction is that the minimum-dilation tour (and path) problem does not admit an FPTAS.

2 Reduction

For a point $p \in \mathbb{R}^2$, we denote by $p(1), p(2)$ its x and y-coordinate respectively.

Let G^∞ be the infinite graph whose vertex set contains all points of the plane with integer coordinates and in which two vertices are connected if and only if the Euclidean distance between them is equal to 1. A *grid graph* is a finite, node-induced subgraph of G^∞. Note that a grid graph is completely specified by its vertex set. We reduce the Hamilton circuit problem in grid graphs, which is strongly NP-hard [4], to the decision version of the minimum-dilation tour problem. Our main result is the following:

Theorem 1 *Given a set P of points in the plane and a parameter $\delta > 1$, the problem of deciding whether there exists a Euclidean Hamilton circuit on P with dilation at most δ is NP-hard.*

Proof. Let G be a grid graph with vertex set V and $|V| = n$. Using V, we construct a point set P such that, for some δ, a Hamilton circuit on P with dilation at most δ exists if and only if G has a Hamilton circuit.

We assume that G has no degree-0 or 1 vertices, since, otherwise, there is no Hamilton circuit in G;

[*]Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, D-10099 Berlin, Germany, `{panos, dmarx}@informatik.hu-berlin.de`

[†]Institut für Informatik, Freie Universität Berlin, Takustraße 9, D-14195 Berlin, Germany, `Knauer@inf.fu-berlin.de`

this can be checked in polynomial time. Consider the smallest enclosing rectangle R of G, see Fig. 1. Since G is finite and $|V| = n$, R has finite dimensions and its height is at most n. Let $v \in V$ be the vertex that

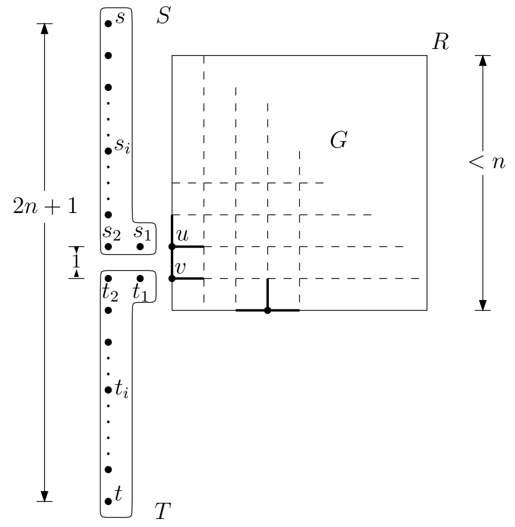


Figure 1: A grid graph G, its smallest enclosing rectangle R, and the point-sets ('handles') S and T.

is closest to the lower-left corner of R and lies on the left vertical edge of R. Then, v must be a degree-2 vertex and have a neighbor on the same edge of R; let u be this vertex. We append two point-sets, called 'handles', S and T to G as shown in Fig. 1. Each handle has one horizontal and one vertical part consisting of 2 and $n + 1$ points respectively, and the two parts have one point in common.

Let $s_1 = (u(1) - 1, u(2))$, $s_2 = (u(1) - 2, u(2))$, and $s_i = (u(1) - 2, u(2) + i - 3)$ with $i = 3, \dots, n + 3$. Similarly, let $t_1 = (v(1) - 1, v(2))$, $t_2 = (v(1) - 2, v(2))$, and $t_i = (v(1) - 2, v(2) - i + 3)$ with $i = 3, \dots, n + 3$. We set $S = \{s_1, s_2, s_i | i = 3, \dots, n + 3\}$ and $T = \{t_1, t_2, t_i | i = 3, \dots, n + 3\}$. Let $W = V \cup S \cup T$. We have that $|W| = 3n + 4$. Copies of W will be included later in P. Consider the points $s, t \in W$ with $s = s_{n+3}$ and $t = t_{n+3}$. We have that $|st| = 2n + 1$. We start with a simple lemma.

Lemma 2 *There exists a Hamilton s-t path on W with length $3n+3$ if an only if there exists a Hamilton circuit in G.*

We continue with the construction of point set P. First, we choose points on a rectangle R' of width α and height β, with $\alpha = (2n^2 + 1)n^6 + 2n^2n^3$ and $\beta = 2n^6 + 3n^3$. Let a, b, c, and d be the upper-right, upper-left, bottom-left, and bottom-right corner points of R' respectively. Consider a straight-line segment of length n^6. We choose a set B of points on the segment at regular intervals such that the distance between any two consecutive points is $n/2$. We have that $|B| = 2n^5 + 1$. We use B as a 'building' block: starting on the upper side of the rectangle, from a, we choose copies of B, simply referred to as *blocks*, at regular intervals of length n^3; see Fig. 2 (to avoid cluttering, the edges of the rectangle are not shown). Let K, L, M, and N be the sets of points on the right, upper, left, and lower side of the rectangle respectively. Sets K and M are unions of two vertical blocks each, while L and N are unions of $2n^2 + 1$ horizontal ones. The right and left-most point of an horizontal block are called the right and left *end-points* of the block. Similarly, the lower and upper-most point of a vertical block are called the lower and upper end-points of the block. Let $K = K_1 \cup K_2$, where K_1, K_2 is the upper and lower block respectively, as shown in Fig. 2. Also, let e be the lower end-point of K_1 and f be the upper end-point of K_2. In the empty interval, i.e., the gap, between K_1 and K_2, we place point set W such that its handles S and T lie on the right side of R'. Additionally, we require that $|es| = |ft| = (n^3 - |st|)/2 = (n^3 - 2n - 1)/2$. Since the height of the minimum enclosing rectangle R of V is at most n, the distance between any point of a block and any point of W is at least $(n^3 - 2n - 1)/2$ as well. A reflected copy of W, denoted by W', is placed between the two blocks (subsets) of M in a similar way.

Let $P = K \cup L \cup M \cup N \cup W \cup W'$ and

$$\delta = \frac{\alpha + \beta - (2n + 1) + 3n + 3}{\beta} = 1 + n^2 + g(n) + h(n),$$

where $g(n) = (n^3 - n^2)/(2n^3 + 3)$ and $h(n) = (n + 2)/(2n^3 + 3)n^3$. Note that $g(n), h(n) < 1$ for every $n \geq 1$. We have that $|P| = (2n^2 + 1)(2n^5 + 1) + 2(2n^5 + 1) = O(n^7)$.

Lemma 3 *If there is a Hamilton s-t path on W with length $3n + 3$, then there is a Hamilton circuit on P with dilation at most δ.*

Proof. Let H_W be a Hamilton s-t path on W with length $3n + 3$. We construct a Hamilton circuit H_P on P by simply connecting the points in K, L, M, N in the 'canonical' way along the sides of rectangle R', as shown in Fig. 2. First, every two consecutive points in each block are connected by an edge. Second, in L, N, the left end-point of each block is connected to the right end-point of its immediate neighbor block. Finally, the upper end-point of K_1 and the lower end-point of K_2 are connected to points a and d respectively, while e connects to s and f connects to t; the blocks of M are connected to b, c, and the point set W' in a similar way. We prove that $\delta(H_p) \leq \delta$. Let p and q be the 'middle' points of L and N respectively. That is, $p = (a + b)/2$ and $q = (c + d)/2$. Note that any path from p to q in H_P must go through either W or W'. By the symmetry of the construction of H_P, we have that

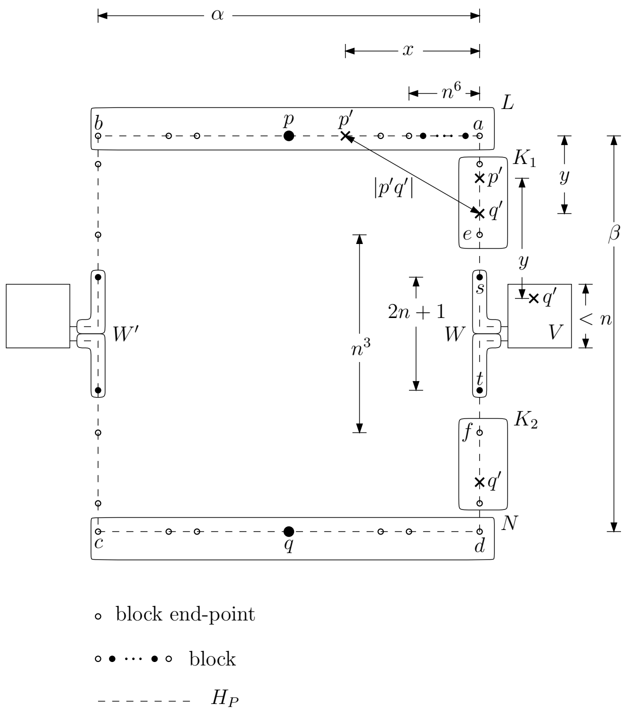


Figure 2: Point set P, Hamilton circuit H_P and example positions of points p' and q'.

$d_{H_P}(p,q) = \alpha+\beta-|st|+d_{H_W}(s,t) = \alpha+\beta+n+2$. It is easy to check that $\delta_{H_P}(p,q) = d_{H_P}(p,q)/|pq| = \delta$. We now prove that for any other pair of points $p', q' \in P$, $\delta_{H_P}(p',q') \leq \delta$. We distinguish the following cases, see Fig. 2:

(i) p', q' lie on opposite sides of R' or $p' \in W'$ and $q' \in K$ (symmetrically, $p' \in M$ and $q' \in W$), or $p' \in W'$ and $q' \in W$. In this case we have that $|p'q'| \geq |pq|$. Note that the total length of H_P is $2d_{H_P}(p,q)$, hence $d_{H_P}(p',q') \leq d_{H_P}(p,q)$. Thus, $\delta_{H_P}(p',q') = d_{H_P}(p',q')/|p'q'| \leq d_{H_P}(p,q)/|pq| = \delta$.

(ii) p', q' lie on non-opposite sides of R' and there is a path in H_P connecting them that visits no point in W (symmetrically, W'). If p', q' lie on the same side of R', then $d_{H_P}(p',q') = |p'q'|$, hence $\delta_{H_P}(p',q') = 1$. If p', q' lie on different, i.e., vertical to each other, sides, then $d_{H_P}(p',q') = |p'a| + |aq'| < 2|p'q'|$, hence $\delta_{H_P}(p',q') < 2$.

(iii) p', q' lie on non-opposite sides of R' and any path in H_P connecting them must visit a point in W (symmetrically, W'). First, $|p'q'| \geq |es| = (n^3 - |st|)/2 = (n^3 - 2n - 1)/2$. Let $x = |p'(1) - a(1)|$ and $y = |p'(2) - q'(2)|$. Then,

$$d_{H_P}(p',q') < x + y + d_{H_W}(s,t) < 2|p'q'| + 3n + 3.$$

Thus, $\delta_{H_P}(p',q') < 2 + (3n+3)/(n^3 - 2n - 1) < 3$, for any $n \geq 3$.

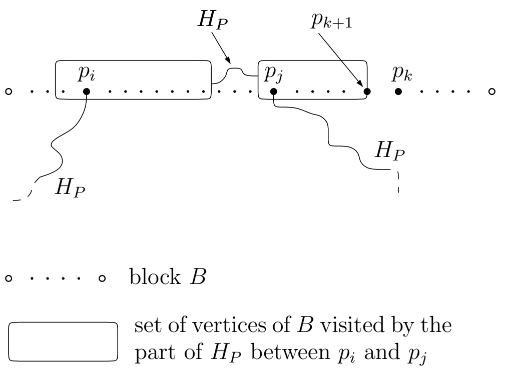
(iv) Finally, when $p', q' \in W$ or W', we have that $d_{H_P}(p',q') \leq d_{H_W}(s,t) = 3n+3$ and $|p'q'| \geq 1$, hence $\delta_{H_P}(p',q') \leq 3n+3 \leq \delta$, for any $n \geq 4$. $\square$

Conversely, we now prove the following.

Lemma 4 *If there is a Hamilton circuit on P with dilation at most δ, then there is a Hamilton s-t path on W with length $3n+3$.*

Proof. Let H_P be a Hamilton circuit on P with $\delta(H_P) \leq \delta$. Also, let $p = (a+b)/2$ and $q = (c+d)/2$. We prove that H_P must contain a path from p to q that is 'locally optimal' in the sense that firstly, it connects p to s and t to q in the 'canonical' way on the sides of R' and, secondly, it connects s to t via a Hamilton path on W with length $3n+3$. In particular, we show that δ is small enough to ensure that the following requirements be met:

(i) Once inside a block, H_P visits all the points of the block before leaving it. To see this, consider a block B and a point $p_i \in B$ for which there is an edge op_i with $o \in P \backslash B$ (such a point must exist since H_P must visit all the points of P); see Fig. 3. We trace H_P starting from o and entering B via op_i. Let us assume that H_P



H_P
p_{k+1}
p_i
p_j
p_k
H_P
H_P
block B
set of vertices of B visited by the part of H_P between p_i and p_j

Figure 3: Case (i) in the proof of Lemma 4.

leaves B before having visited all its points and let p_j be the last point visited and p_k be a point that is left out. Then, at least one neighbor point, say p_{k+1} of p_k in B is not connected to p_k via this part of H_P inside B. Also, H_P must visit p_k but only after it has visited at least one point outside B. Let o' be such a point connected to p_j with an edge p_jo'. Note that o' must be in some block other than B or in W or in W'; the same holds for o. Recall that the distance between any two blocks is at least n^3 and that the distance between any block and W or W' is at least $|es| = (n^3-2n-1)/2$. Hence, $|p_jo'| \geq (n^3-2n-1)/2$ and $|op_i| \geq (n^3-2n-1)/2$ as well. We have that $d_{H_P}(p_{k+1},p_k) > 2\min\{|p_jo'|, |op_i|\} \geq 2(n^3-2n-1)$ and, thus, $\delta_{H_P}(p_{k+1},p_k) > 2n^2-6 > \delta$, for any $n \geq 3$.

(ii) Once inside W (or W'), H_P visits all the points of W (or W') before leaving it. This can be seen by using arguments similar to the ones in case (i).

(iii) Any two blocks that are consecutive along the sides of R' must be 'connected' by an edge in H_P, as long as W or W' does not lie between the two blocks. To see this, consider a block B and a neighbor of it, B', and let p' and q' be the endpoints of B and B' respectively, with $|p'q'| = n^3$. Assume that H_P contains no edge connecting a point of B to a point of B'; see Fig 4. Then, any path from p' to q' in H_P must visit some other block, different from B and B', and, hence, $d_{H_P}(p', q') > n^6$, where n^6 is the diameter of any block. Thus, $\delta_{H_P}(p', q') > n^3 > \delta$, for any $n \geq 2$.

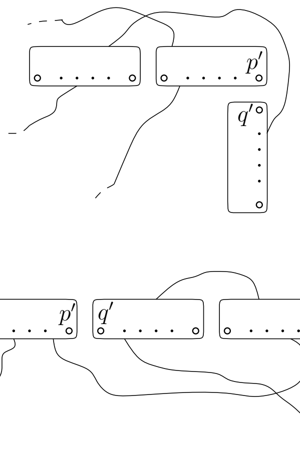


Figure 4: Examples of the case (iii) in the proof of Lemma 4.

(iv) Blocks K_1, K_2 must be connected to W by an edge in H_P; this holds also for the blocks in M and W' (similar to the case (iii)).

All the above requirements assert that H_P does not contain 'long' edges (jumps) between any two points of P that belong to different blocks or between a point of a block and a point of W or W'. Note that, since all points of a block lie on the same straight-line segment, the minimum-length Hamilton path on a block has length n^6; any detour increases this length by at least n. Also, s and t are the points of W that are closest to e and f respectively. Case (ii) also asserts that the part of H_P inside W (W') forms a Hamilton s-t path on W (W'); let l be its length. Consider now the pair p, q. By combining all the above, we have that

(1) $d_{H_P}(p,q) \geq |pa|+|ae|+|es|+l+|tf|+|fd|+|dq| = \alpha + \beta - (2n+1) + l$.

Since $\delta(H_P) \leq \delta$, we have that $\delta_{H_P}(p,q) \leq \delta$ as well. From the proof of Lemma 3, this implies that

(2) $d_{H_P}(p,q) \leq \alpha + \beta + n + 2$.

From (1), (2), we have that $l \leq 3n+3$. However, any Hamilton path on W has length at least $3n+3$, and the lemma follows. □

Note that all points in P have rational coordinates with numerators and denominators bounded by a polynomial in n. Also, the construction of P takes $O(|P|) = O(n^7)$ time. Combining Lemmata 2, 3, and 4, concludes the proof of the theorem. □

Any Hamilton path on W has length at least $3n+3$ and, from Lemma 2, this value is achieved if and only if there is a Hamilton circuit in G. On the other hand, if there is no Hamilton circuit in G, then any Hamilton path on W has length at least $(3n+3) - 1 + \sqrt{2}$: In this case, an edge with length at least the length of a diagonal of the grid must be used by the path. This observation implies that the minimum-dilation tour problem admits no-FPTAS.

Corollary 5 *The minimum-dilation tour problem does not admit an FPTAS.*

As it is easy to see, Theorem 1 and Corollary 5 hold also for the decision version of the minimum-dilation path problem by considering a point set that contains p and q, the points of P that lie on R' to the right side of p and q, and the points in W.

Corollary 6 *Given a set P of points in the plane and a parameter $\delta > 1$, the problem of deciding whether there exists a Euclidean Hamilton path on P with dilation at most δ is NP-hard. The minimum-dilation path problem does not admit an FPTAS.*

References

[1] O. Cheong, H. Haverkort, and M. Lee. Computing a minimum-dilation spanning tree is NP-hard. In *Proc. of Computing: the Australasian Theory Symposium (CATS)*, 2006. To appear.

[2] D. Eppstein. Spanning trees and spanners. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[3] D. Eppstein and K.A. Wortman. Minimum dilation stars. In *Proc. of the 21st ACM Symp. Comput. Geometry*, pages 321–326, 2005.

[4] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Computing*, 11(4):676–686, 1982.

[5] R. Klein and M. Kutz. Computing geometric minimum-dilation graphs is NP-hard. In *Proc. of the 14th Internat. Symp. on Graph Drawing*, 2006. To appear.

[6] G. Narasimhan and M. Smid. *Geometric Networks.* Cambridge University Press. To appear.

[7] M. Smid. Closest point problems in computational geometry. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science Publishers, 2000.

Existence of Simple Tours of Imprecise Points [*]

Maarten Löffler [†]

Abstract

Assume that an ordered set of imprecise points is given, where each point is specified by a region in which the point may lie. This set determines an imprecise polygon. We show that it is NP-complete to decide whether it is possible to place the points inside their regions in such a way that the resulting polygon is simple. Furthermore, it is NP-hard to minimize the length of a simple tour visiting the regions in order, when the connections between consecutive regions do not need to be straight line segments.

1 Introduction

Traditionally, geometric algorithms assume exact input. However, in practice there is often some imprecision in real-life input data, which may result in artifacts. For example, a polygon that describes the boundary of some region may have self-intersections.

If we know that the vertices of some polygon are imprecise, given by some region in which they lie, the question arises how to place the vertices inside their regions such that the resulting polygon has no self-intersections. We show here that it is NP-complete to determine whether such a placement is possible, and hence also to find one if it exists. This is a refinement of an earlier result, where we proved that it is NP-hard to find the minimum perimeter polygon that has no self-intersections [4].

In this paper, we study the following problem. We are given an *ordered* set $\mathcal{S} \subset \mathcal{P}(\mathbb{R}^2)$ of n connected regions in the plane. We are looking for a tour (closed curve) that visits all regions of $\mathcal{S}$ in the correct order. We call such a tour *simple* if it does not cross itself. We call such a tour *straight* if it is a polygon with a vertex in each region, and no other vertices. We are interested in the existence of a simple straight tour. Figure 1 shows an example of an ordered set of regions and some tours through them.

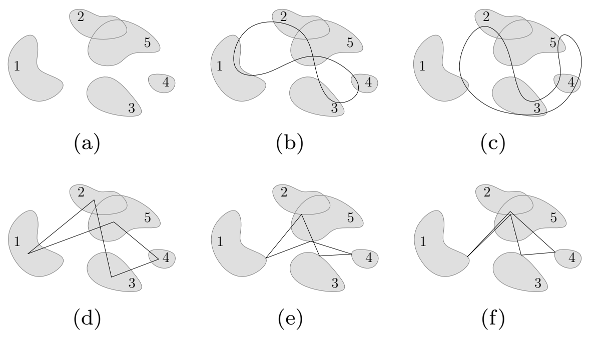


Figure 1: (a) Five regions and an order on them. (b) A tour passing through the regions in order. (c) A simple tour. (d) A straight tour. (e) The shortest tour. (f) The shortest simple tour.

For non-simple tours, it is easy to see that the shortest one is always straight. This problem has been studied by Dror *et al.* [2], and can be solved in near-linear time if the regions are convex polygons, while it is NP-hard for non-convex regions. For square regions, the shortest and longest straight tours can be computed in $O(n)$ time [4]. If the regions are all adjacent to the inner boundary of a simple polygon, this problem is known as the Safari Keepers problem [6, 9]. On the other hand, if we want to find a simple tour, the shortest one is not always straight. We also prove that finding the shortest simple tour is NP-hard. This answers an open question posed by Polishchuk and Mitchell [7].

The problem of finding the shortest tour for an *unordered* set of regions has been well studied before and is generally called the Traveling Salesman Problem with Neighbourhoods, or (Planar) Group-TSP. This problem is obviously NP-hard. Mata and Mitchell [5] give a constant factor approximation algorithm for some region models; additional results can be found in [1, 8].

The remainder of this paper is organized as follows. The next section contains the NP-hardness proof for the problem of deciding whether a simple straight tour through an ordered set of vertical line segments exists. Section 3 shows how to extend this result to other imprecision regions like squares and circles. Section 4 describes the NP-hardness of the problem of finding a shortest simple tour. Finally, some concluding re-

[*]This research was partially supported by the Netherlands Organisation for Scientific Research (NWO) through the project GOGO.

[†]Institute of Information and Computing Sciences, Utrecht University, `loffler@cs.uu.nl`

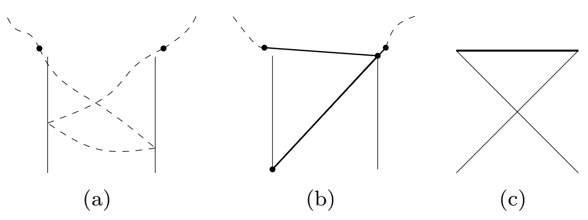


Figure 2: (a) The input for a pair of scissors. (b) One of the solutions, representing the state `True`. The mirrored solution represents the state `False`. (c) Schematic representation.

marks are given in Section 5.

2 Simple Straight Tours through Vertical Line Segments

Given a set of parallel line segments and a cyclic order on them, we want to choose a point on each segment such that the polygon determined by those points in the given order is simple. The decision problem of whether this is possible is NP-complete. The problem is trivially in NP, and we prove NP-hardness by reduction from planar 3-SAT [3]. For different components of a planar 3-SAT instance, we construct polygonal chains that will be connected into a polygon in the end. A simple polygon can be realized if and only if the 3-SAT instance is satisfiable.

In the construction we give here, the simple polygon will be a degenerate one if it exists. A degenerate simple polygon is a polygon for which it is possible to move all vertices over an arbitrarily small distance to make it into a simple polygon. However, we will show that the gadgets can be adapted slightly to also allow non-degenerate simple polygons.

We represent variables by scissor gadgets as in Figure 2(a). This gadget consists of two imprecise points and two precise (or degenerate imprecise) points. The dashed lines depict the order in which the tour should visit these regions. There are two possible ways to make a simple straight tour through this gadget, which represent the two different values of a variable. The solution with a positive sloping diagonal, see Figure 2(b), represents the value `True`, and the negative sloping diagonal represents the value `False`. In the remainder of the proof we use a schematic drawing for this configuration of four imprecise points, see Figure 2(c).

We can make a chain of scissor gadgets that all represent the same variable, as shown in Figure 3. Here each scissor gadget is represented schematically as two crossing diagonal lines and a horizontal line indicating where the gadget is connected to the polygon. For each scissor gadget one of the legs is used in a solution, the other is not. There are only two possible states to this chain: either all of the scissor gadgets use their positive sloping leg or they all use their negative sloping leg.

We can also split this chain into more chains with a junction as shown in Figure 4(a). Here scissor gadgets of two different sizes are used, but still there are only two possible states in the total structure: either all scissor gadgets use their positive sloping leg, or all use their negative sloping leg. The chains can be split again to make as many chains for a variable as needed. Furthermore, we can make chains under a slope of almost 45°, and by zigzagging between junctions we can move over vertical distances, see Figure 4(b).

We represent the clauses of the 3-SAT formula by clause gadgets, as in Figure 5(a). This configuration consists of one imprecise point and four precise points. For clauses, there are three unconnected polygonal chains that visit the gadget, which are represented by dashed lines. The three possible solutions for this situation can be seen in Figures 5(b), 5(c) and 5(d). The idea is that in order to find a total solution, at least one of the three solutions to the clause must be possible. For example, if we want to build the clause `a` $\vee$ `b` $\vee$ $\neg$`c`, we intersect the negative sloping leg of the variable `a` with one of the three solution paths of the clause, the negative sloping leg of the variable `b` with another path, and finally the positive sloping leg of `c` with the remaining path, and the clause can be solved if and only if the logical clause is satisfied, see Figure 5(f).

Now that we have structures for variables and clauses, we can build an instance of planar 3-SAT by embedding the graph in the plane and making it wide enough to fit all the structures such that they do not interfere. However, this does not complete our construction yet. The scissor and clause gadgets have some precise points where the tour is supposed to enter and leave the gadget. We need to construct a tour that visits all gadgets in any order, but in such a way that it does not interfere with the gadgets. We can easily do this by linking neighbouring gadgets together, see Figure 5(g). However, doing this will result in a number of smaller tours instead of one big tour, because the 3-SAT instance partitions the plane

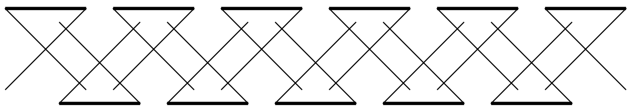


Figure 3: A chain of scissors.

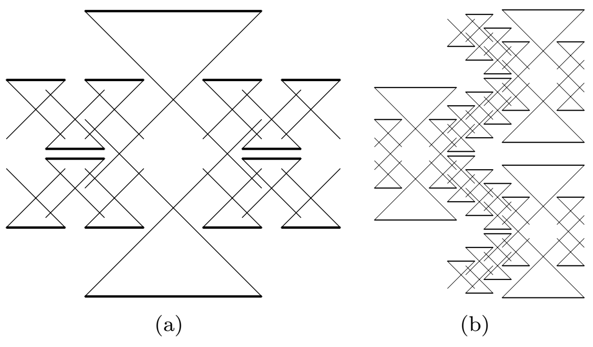


Figure 4: (a) A junction to split the chain of scissors. (b) Going vertical.

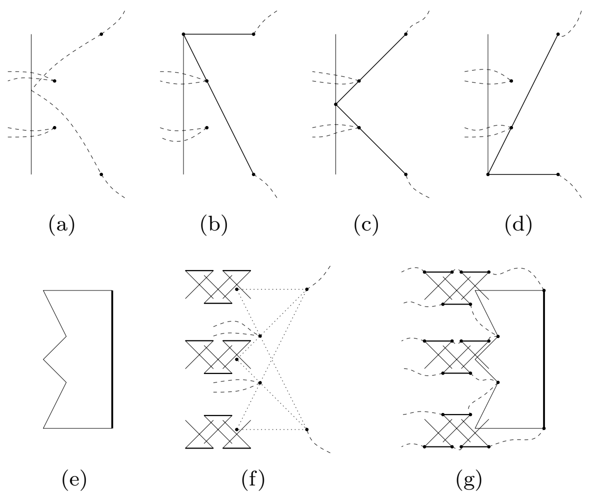


Figure 5: (a) The input for a clause. (b) One of the three solutions. (c) Another solution. (d) The third solution. (e) Schematic representation. (f) The clause attached to the three variables. (g) Connecting the gadgets.

into a number of faces. We need one tour to visit all gadgets, and therefore all faces. This means we need the tour to cross the scissor chains. Therefore, we design another primitive, see Figure 6(a).

This bridge gadget consists of two imprecise points and four precise points, and has two chains passing through it, again indicated by the dashed lines. Like the scissor gadget, the bridge gadget has two possible solutions representing the values `True` and `False` of a variable, see Figure 6(b). A schematic representation is shown in Figure 6(c). Bridge gadgets can be embedded in chains of scissor gadgets, and they preserve the property that the whole chain uses either positive or negative sloping legs, see Figure 7. However, we now have two parts of the tour that cross the chain.

Now we can include bridges into the network such that all faces of the embedded planar 3-SAT graph are connected by bridges, see Figure 8. All we need to do now is connect the fat edges to each other with a fixed part of the tour (a part that only contains precise points), and we have a valid input for the problem. The number of imprecise points in the construction is clearly polynomial in the length of the 3-SAT instance, which completes the proof.

Theorem 1 *Given an ordered set of n vertical line segments, it is NP-hard to decide whether it is possible to choose a point on each segment such that the resulting polygon is simple.*

It is easy to adapt the gadgets slightly to also allow non-degenerate polygons, without damaging the proof. For the scissor gadgets, just make the vertical line segments slightly longer; for the clauses, move the two central precise points slightly towards the imprecise point.

3 Simple Straight Tours through General Regions

In the gadgets of the proof, we can replace the vertical line segments by narrow rectangles. Next, we observe that we can scale the input of the problem in the x direction without interfering with the existence of a simple straight tour. If we model the imprecise points as scaled copies of any connected shape, e.g. circles, we can deform the input such that their bounding boxes become narrow rectangles. Because the regions are connected, there is a point inside the region for every y-coordinate of each narrow rectangle, and the proof works as before.

Theorem 2 *Given an ordered set of n arbitrarily scaled copies of any connected region, it is NP-hard to decide whether it is possible to choose a point in each region such that the resulting polygon is simple.*

4 Simple Tours through Line Segments

If we drop the requirement that the edges between two consecutive points need to be straight line seg-

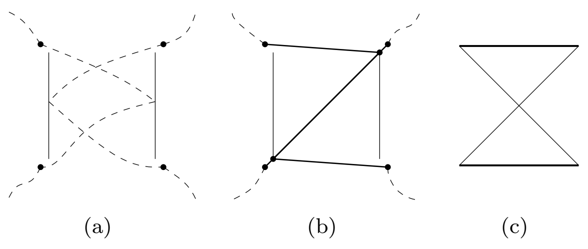


Figure 6: (a) The input for a bridge. (b) One of the solutions, representing the state `True` . The mirrored solution represents the state `False` . (c) Schematic representation.

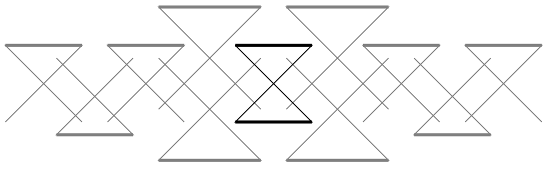


Figure 7: A bridge embedded in a chain of scissors.

ments, a simple tour always exists. In this context, it is interesting to consider *shortest* simple tours. Finding such a shortest tour is also NP-hard. We need to make slightly more complex gadgets. In the scissors gadget, we need to explicitly ensure that the tour goes down, so we add a horizontal segment, see Figure 9. We also need to adapt the clause gadget, to ensure that the three solutions all have the same length, see Figure 10. The bridge gadget still works. Now, if the 3-SAT instance is satisfiable, a tour exists that is considerably shorter than the shortest tour if the instance is not satisfiable. We can extend the proof to squares by noting that in all gadgets the given segments might as well be sides of squares, without allowing any shorter solutions.

Theorem 3 *Given an ordered set of n axis-parallel line segments or squares, it is NP-hard to find a tour that visits all segments or squares in order such that this tour is simple and as short as possible.*

5 Conclusions

We studied the problem of finding a simple straight tour through a sequence of regions, and proved that it is NP-complete to decide whether this is possible. We also proved that it is NP-hard to find the shortest non-straight tour, resolving an open problem from [7].

It still remains open whether a shortest simple tour

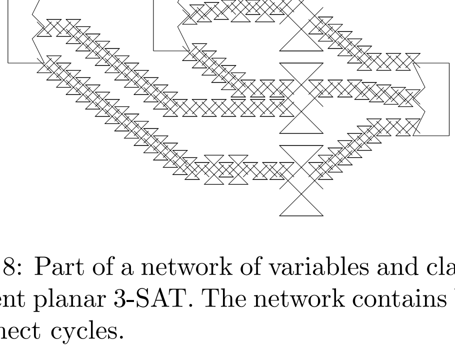


Figure 8: Part of a network of variables and clauses to represent planar 3-SAT. The network contains bridges to connect cycles.

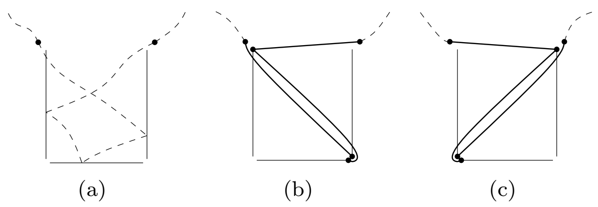


Figure 9: Adapted scissors.

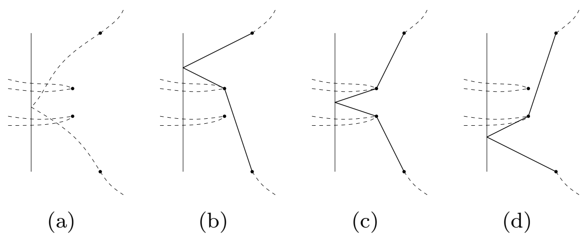


Figure 10: Adapted clause.

visiting a set of single points, instead of regions, can be found efficiently.

Acknowledgements The author thanks Marc van Kreveld for helpful discussions.

References

[1] M. de Berg, J. Gudmundsson, M. J. Katz, C. Levcopoulos, M. H. Overmars, and A. F. van der Stappen. TSP with neighborhoods of varying size. *Journal of Algorithms*, 57(1):22–36, 2005.

[2] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th ACM Sympos. Theory Comput.*, pages 473–482, 2003.

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

[4] M. Löffler and M. van Kreveld. Largest and smallest tours and convex hulls for imprecise points. In *Proc. 10th Scandinavian Workshop on Algorithm Theory*, LNCS 4059, pages 375–387, 2006.

[5] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1995.

[6] S. Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, 1(3):149–170, 1992.

[7] V. Polishchuk and J. S. B. Mitchell. Touring convex bodies - a conic programming solution. In *Proc. 17th Canad. Conf. on Comp. Geom.*, pages 290–293, 2005.

[8] S. Safra and O. Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. *Comput. Complexity*, 14(4):281 – 307, 2006.

[9] X. Tan and T. Hirata. Finding shortest safari routes in simple polygons. *Information Processing Letters*, 87:179–186, 2003.

Convex Approximation by Spherical Patches

Kevin Buchin*‡ Simon Plantinga †‡ Günter Rote*‡ Astrid Sturm*‡ Gert Vegter†‡

Abstract

Given points in convex position in three dimensions, we want to find an approximating convex surface consisting of spherical patches, such that all points are within some specified tolerance bound ϵ of the approximating surface. We describe a greedy algorithm which constructs an approximating surface whose spherical patches are associated to the faces of an inscribed polytope. We show that deciding whether an approximation with not more than a given number of spherical patches exists is NP-hard.

1 Introduction

Problem Statement. We are given a set P of n three dimensional points in convex position. We want to find a convex approximating surface S that consists of *spherical patches*. A spherical patch is part of the boundary of a sphere. There are two quality criteria that we want to optimize: (a) the approximation error, which is defined as the maximum distance from a point of P to S; and (b) the number of patches.

Motivation. Our motivation for studying this problem is based on open problems in polytope approximation as well as on practical considerations. Surface reconstruction and surface simplification is an important area of computer graphics and geometric modeling [1]. One goal is to approximate complex objects by simpler shapes. A lot of research has been done in the field of approximation of three dimensional point sets with polytopes with surfaces of higher order [3].

A first natural step to higher order approximation is the approximation with spheres or spherical patches. Since polyhedral facets can be seen as spherical patches with infinite radius, spherical patch approximation generalizes polytope approximation.

We initiate the study of this problem by considering convex surfaces only, for simplicity. The results might nevertheless be interesting for real data sets, e.g. data sets from imaging procedures such as MRT. A wide range of objects scanned for data consist of *parts* that are convex, and thus our results remain valid at least on a local scale.

The complexity of the approximation problem is related to open problems in polytope approximation, in particular to the complexity of the minimum facet polytope approximation. We hope to use our new methods from the NP-hardness proof of the more generalized problem to solve the complexity question of the minimum facet polytope approximation.

Results and Techniques. We present an algorithm for solving the approximation problem with a specified error bound ε. It is based on a triangulated inscribed polytope which is the convex hull of a subset of the input points, and on which the spherical patches are built. This polytope is successively refined in a greedy manner. We attempt to extend the well-known Douglas-Peuker algorithm for polygonal line approximations of curves to our setting.

Proofs are omitted due to space constraints. A full version is available[1].

2 Approximation of a convex point set by Spherical Patches

Outline The optimization problem we are considering is the *Approximation by Spherical Patches problem* (ASP): the approximation of a convex point set with a number g of spherical patches resulting in a convex surface with all points within some specified tolerance to the surface. We construct a point set defining an instance of the ASP with zero tolerance such that - in the satisfiable case - a minimal solution of the approximation problem corresponds to a truth assignment in the NP-hard grid-3-SAT problem. This point set is lifted to a paraboloid and extended with additional points. We describe the minimal solution in the satisfiable case and prove that more patches than g are needed in the non-satisfiable case for the ASP.

Grid-3-Satisfiability 3-SAT statements consist of a Boolean conjunction of clauses, where each clause consists of a disjunction of three boolean variables, each of which may be negated. Such a statement can be represented by a bipartite graph, where variables

*Institut für Informatik, Freie Universität Berlin, {buchin, rote, sturm}@inf.fu-berlin.de

†University of Groningen Department of Mathematics and Computing Science, {simon, gert}@cs.rug.nl

‡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)

[1]http://page.mi.fu-berlin.de/ sturm/Spheres.pdf

and clauses are represented by vertices. Each clause vertex is connected to its three variable vertices by an edge marked + or − depending on whether this variable occurs negated in that clause. The 3SAT problem is NP hard even if the variable-clause graph of a formula of length n in 3-conjunctive normal form can be embedded on a $c \cdot n^2$ grid with c some constant [2].

Modifying the grid The first step of the reduction requires a refinement of the grid, vertices correspond to facets and edges correspond to rectilinear paths on the grid. Further we disperse the grid cells by a small constant factor δ which creates free space between the cells. Depending on the label of the edge in the variable clause graph, we change the number of facets in the path on the grid corresponding to the edge. A negatively labeled edge is represented by a path with an odd number of cells and a positively labeled edge corresponds to a path with an even number of cells. To achieve this correspondence we need sufficiently many cells on a straight path. The inclusion of an additional cell is done by reducing the size of the cells in a straight segment of the path and fitting in another cell of this size. Next we delete all grid cells corresponding to clauses (the clauses will be represented later by a single point). We also drop the lower right vertex of each grid cell.

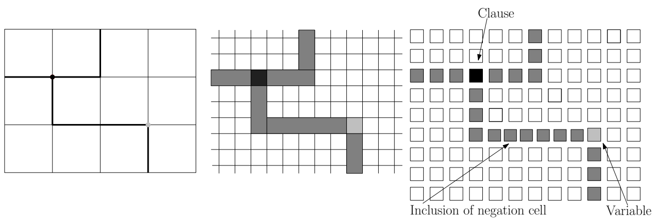


Figure 1: refinement of the grid

Lifting to a paraboloid The next step is a lifting of the point triples of the grid cells onto a very flat paraboloid. The distance between vertices in one grid cell is set to one. For a $c \cdot (n \times n)$ grid we pick a paraboloid of the form $z = \lambda \cdot (x^2 + y^2)$. The parameter λ has to be chosen in such a way that for two neighboring point triples the disks D_i and D_j corresponding to the circles C_i and C_j intersect. This guarantees the existence of valid spherical patches. For a lifting of a δ dispersed $c \cdot (n \times n)$ grid this leads to an upper bound on λ of $\frac{\sqrt{\left(1-1/\sqrt{2}\right)^2-\delta^2}}{\delta^2+2\delta(c+\delta)n}$. For this, δ has to be chosen less than $1 - 1/\sqrt{2}$. For our explicit construction we choose $\delta := 1/10$ and $\lambda := 1/(10m)$ with $m = c(1 + \delta)n$ a bound on the width and length of the dispersed grid (see Theorem 4).

Fill points After lifting the point set we place one point into each triangular face defined by point triples corresponding to grid cell vertices of cells which did not belong to the 3SAT. We refer to these four points as a set of fill-points.

Lemma 1 *Each set of fill points induces exactly one spherical patch and all sets cannot be covered with less than one patch per set.*

Wire The wire corresponds to edges in the variable-clause graph. An edge in the variable-clause graph corresponds to a set of the lifted point triples. Each point triple P_i defines a circle C_i. These circles do not lie on the paraboloid, but (since λ is small) lie close to the lifted circumcircles of the base squares (which are ellipses). The supporting plane of each circle splits the space into an inner half space containing the convex hull of the original point set and an outer half space. Each circle C_i defines a *family of spheres*, i.e. set of all spheres induced by the circle C_i. Candidates for valid patches are only spheres with centers in the inner half space. Furthermore adjacent spherical patches need to intersect properly. The intersection of the outer half space with the circle C_j of an adjacent patch has to lie outside the spherical patch, i.e., the spherical patch should pass below the circular arcs of its neighbors.

We build a wire out of consecutive spherical patches to propagate information from the variables to the clauses. The main idea of the reduction is to place points on the intersection of consecutive spherical patches in the wire. These points narrow down the choice from a family of spheres to only two spheres for each patch a flat or bulbous patch. Furthermore the points force alternating spherical patches in the wire.

The approximating surface is constructed by taking the inner upper hull of the patch intersection. This is the surface of the intersection of the patch defining spheres. We need to guarantee that the additional points on the flat and bulbous patches will lie on the approximating surface (see Lemma 3). We place four points, F_k, on the intersection circle of consecutive flat patches and four points, B_{k_i}, on the intersection of bulbous patches. The points F_{k_i} lie on the circular arc which is in between the intersection of the defining circles and the points B_{k_i} lie on the circular arc outside the intersection (see Figure 2).

For the wire gadget we need to place a point (approximately) on the intersection of two neighboring "bulbous" spheres. If at least one of the two bulbous spheres is chosen the point must lie on the inner hull of the construction. In the following we formulate conditions under which a point lies on the inner hull. Then we pick such a point and prove that the conditions hold.

Since the radii of the bulbous spheres have been chosen in such a way that they only come close to

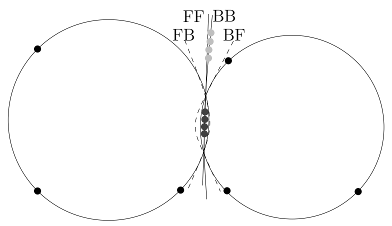


Figure 2: Placing of the points on the circular arc of the FF and BB patch intersection. The light gray points are on BB.

the grid polytope at the face by which it is defined, it suffices to consider the local configuration. The *grid polytope* is the polytope obtained by lifting all the grid vertices restricted to the region of the 3SAT construction. Thus, for a point to lie on the inner hull the following conditions are sufficient:

1. The point lies above (i.e. on the same side as the grid polytope) both of the planes defined by the two triples of points.
2. The point lies below (i.e. on the other side than the grid polytope) the face of the grid polytope between the two faces defining the spheres.
3. The projection of the point lies within (possibly on the boundary of) the face of the grid polytope between the two faces defining the spheres.

For two neighboring triples of points there are two points p_1 and p_2, one of each triple, neighboring in the grid. Let e be the plane orthogonal to the z-axis through these two points. For the intersection point of the plane e with the two spheres we can prove that the conditions above hold.

For a flat patch we choose the sphere with center at infinity, the plane defined by the point triple. For a bulbous patch we request that all points of the grid polytope except the point triple defining the patch lie inside the “bulbous” ball. This leads to a set of constraints on the radius of the ball by considering the radii of the balls defined by the point triple and a set of possible fourth points. These constraints can be fulfilled by a radius linear in n. To guarantee further that all points of the grid polytope except the point triple have distance of at least $1/n^2$ to the surface of the ball the radius can be chosen quadratic in n. We choose for all bulbous patches the same radius.

Variable A variable is a point triple which is handled as a wire point set. Choosing the flat patch corresponds to a 0 assignment and the bulbous patch to a 1 assignment. Choosing the flat patch will result in a covering of all flat points in the free space around the variable point triple, therefore all consecutive wire patches will propagate the same information - all wires starting from this variable will start with a bulbous patch. The case of picking a bulbous patch for a variable point triple is symmetric.

Clause Before the lifting a clause corresponds to a grid cell in the plane which is connected to three wires (from three variables). The vertices of this grid cell are not lifted. In the lifted point set the clause corresponds to a single point. This point is placed in the free space between the three wire point triples and is the intersection point of the bulbous patches of these point sets.

Theorem 2 *There exists a true assignment for the grid 3SAT instance if and only if the lifted point set with all additional points can be approximated with s spherical patches. For a $c(n \times n)$ grid, with g clauses and t cells included for negation, $s = c(n^2) + t - g$*

Lemma 3 *All point sets F_{k_i} and B_{k_i} are on the approximating surface.*

Theorem 4 *For a SAT instance on a $c(n \times n)$ grid let P be the set of points in convex position constructed as above with $\delta := 1/10$, $\lambda := 1/(10m)$, $m := \sqrt{2}(1+\delta)cn$, and the common radius of the bulbous spheres $r := 10m$. Let P' equal P with the exception that the points on the bulbous-bulbous sphere intersections might be displaced by $\epsilon := 1/m^2$. For sufficiently large m the following holds: If the SAT instance is feasible, then there is a surface with g patches such that all points have distance at most ϵ to a patch. If the SAT instance is infeasible, then for every surface with at most g patches there is at least one point which has distance more than 100ϵ from all patches.*

3 Greedy algorithm

In this section we present a construction of curved surfaces based on inscribed polytopes, resulting in a convex surface consisting of spherical patches. The inscribed polytope approach makes our construction suitable for various incremental algorithms. We start with a minimal inscribed polytope consisting of a tetrahedron, and incrementally extend this polytope until the corresponding surface is a valid approximation.

Constructing a curved surface In order to produce a valid surface, we require that spherical caps pass through triples of input points ensuring that adjacent caps intersect properly. The approximating surface is generated by a convex triangulation, in particular the convex hull of a subset of the input points. The triangles of this hull are called *supertriangles*. Our goal is to inflate this polytope by replacing its faces with curved, spherical patches.

First we construct the spherical caps. The supporting plane of each supertriangle splits space into an inner halfspace containing the convex hull, and an outer halfspace. For each supertriangle we construct a spherical cap by first taking a sphere through its vertices with its center in the inner halfspace. Then we take the intersection of this sphere with the outer halfspace. The intersection of the outer halfspace with the circumcircle of an adjacent supertriangle has to lie outside the spherical cap (see Figure 3), i.e., the spherical cap should pass below the circular arcs of its three neighbors.

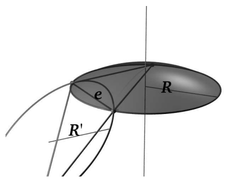


Figure 3: Supertriangle with spherical cap.

For each neighboring supertriangle, the circumcircle and dihedral angle give a lower bound on the radius of the spherical cap, to ensure that the cap is flat enough to pass below that circumcircle. Taking the maximum over the three adjacent supertriangles results in a single lower bound for the cap radius. The centre of the spherical cap now has to lie on a halfline perpendicular to the supertriangle. The approximating surface consists of the inner hull of the union of these spherical patches.

Lemma 5 *If neighboring spherical caps intersect properly, the inner hull of the union of caps forms a convex surface.*

Incremental construction We can now construct a curved convex surface from a subset S of the input points P. The convex hull of S generates a surface as long as the patch radii are large enough, to ensure proper intersection.

For an incremental approach, we initialize S to the four extremal points of the point set P, in the directions of the normals of a regular tetrahedron. Respecting the lower bound on the radii, we try to choose cap radii such that the caps are closer than ϵ to the remaining input points. If this is not possible we add more input points to S.

A supertriangle is *valid* if there exists a corresponding spherical cap with radius larger than its lower bound, such that all points inside the outer halfspace of the supertriangle are closer than ϵ to this spherical cap.

If all supertriangles of the inscribed polytope are valid, all input points lie closer than ϵ to the union of caps. However, it is still possible that they are not ϵ-close to the inner hull of these caps, especially if adjacent supertriangles have a large dihedral angle or if some supertriangles are obtuse. We therefore have to test whether the input points that are ϵ-close to a cap but not to the patch, are ϵ-close to their nearest patch.

Testing supertriangles for validity results in more bounds for the patch radius. The centre of the spherical cap has to lie on the *centre line* of the supertriangle, which is the line passing through the circumcentre and is perpendicular to that triangle. If an input point inside the corresponding halfspace needs to be ϵ-close to the spherical cap, this condition gives an interval of valid cap centres on the centre line. If the intersection of all of these intervals together with the half-line is nonempty, the supertriangle is valid. Since the lower bound for proper intersection of caps corresponds to the surface being convex, we expect the intersection of these intervals to lie within that valid half-line.

First we test all supertriangles for validity. If there are invalid supertriangles, we add an input vertex to S and update the convex hull incrementally. We then test the validity for the newly constructed supertriangles and for previously invalid neighbors of new supertriangles. This way we gradually refine the approximating curved surface without having to revalidate the entire structure.

For an invalid supertriangle we choose the outlying input point corresponding to the smallest radius of the spherical cap. As we increase the number of spherical caps, we increase the radius. This is motivated by the fact that we want flatter patches as the dihedral angles between supertriangles decrease. The incremental algorithm now moves gradually from a curved surface consisting of four patches, to the entire convex hull of P.

The extra test for points close to caps but not to the corresponding patch can also reveal points that are further than ϵ from the approximating surface. These points are also added to S.

Lemma 6 *The greedy algorithm terminates.*

References

[1] T. K. Dey. Curve and surface reconstruction. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, CRC Press, 1997, 2004*, volume 2. 2004.

[2] M. Godau. On the difficulty of embedding planar graphs with inaccuracies. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing*, volume 894 of *LNCS*, pages 254–261. Springer, 1994.

[3] N. Kruithof and G. Vegter. Envelope surfaces. In N. Amenta and O. Cheong, editors, *Symposium on Computational Geometry*, pages 411–420. ACM, 2006.

Guarding Rectangular Partitions

Yefim Dinitz* Matthew J. Katz† Roi' Krakovski‡

Abstract

A rectangular partition is a partition of a rectangle into non-overlapping rectangles, such that no 4 rectangles meet at a common point. A vertex-guard is a guard located at a vertex of the partition (i.e., at a corner of a rectangle); it guards the rectangles that meet at this vertex. An edge-guard is a guard that patrols along an edge of the partition, and is thus equivalent to two adjacent vertex-guards. We consider the problem of finding a minimum-cardinality guarding set for the rectangles of the partition. For vertex-guards, we prove that guarding a given subset of the rectangles is NP-hard. For edge-guards, we prove that guarding all rectangles, where guards are restricted to a given subset of the edges, is NP-hard. For both results we show a reduction from vertex cover in triangle-free 3-connected cubic planar graphs.

For the second NP-hardness result, we obtain a graph-theoretic result which establishes a connection between the set of faces of a plane graph of vertex degree at most 3 and a vertex cover for this graph.

We show that the vertices of a rectangular partition can be colored red, green, or black, such that each rectangle has all 3 colors on its boundary. We conjecture that the above is also true for 4-coloring. Finally, we obtain an upper bound for guarding by edge-guards rectangular partitions with some restriction on their structure.

1 Introduction

A *rectangular partition* R is a partition of a rectangle into non-overlapping rectangles, such that no 4 rectangles meet at a common point. One can think of a rectangular partition as a plane graph with the rectangles as faces, the rectangles' corners as vertices, and the line segments between adjacent vertices as edges. Following this description, we denote by $r(R), v(R)$, and $e(R)$ the set of rectangles, vertices, and edges, respectively, of a rectangular partition R.

Let R be a rectangular partition. The dual graph of R, $D(R)$, is obtained by representing each rectangle of R by a vertex, and drawing an edge between two vertices if their corresponding rectangles have a (boundary) point in common. A *rectangular dual* $R(G)$ of a plane graph $G(V,E)$ is a rectangular partition R consisting of $|V|$ rectangles, such that (i) each vertex $v \in V$ corresponds to a distinct rectangle in R, and (ii) two rectangles in R touch each other, i.e., have a (boundary) point in common, if and only if there exists an edge in E between the two vertices in V that correspond to these two rectangles. A clique cover of a graph $G(V,E)$ is a partition of V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that the subgraph induced by V_i is a complete graph, for $i = 1, \ldots, k$. In the minimum clique cover problem, one needs to find a clique cover of G of minimum cardinality.

It is easy to observe that the problem of finding a minimum guarding set consisting of vertex guards for a rectangular partition R (VGP) is equivalent to finding a minimum clique cover of $D(R)$. Note that if R is not a single row (alternatively, column) of rectangles, then its dual graph is a plane triangulation. Thus, VGP is equivalent to finding a minimum clique cover in plane triangulations that admit a rectangular dual. Despite this connection, we do not know whether VGP is NP-hard.

Czyzowicz et al. [2] showed that the dual graph of a rectangular partition with n rectangles (where n is even) admits a perfect matching. This result immediately implies an $\lceil \frac{n}{2} \rceil$ upper bound on the number of vertex guards needed for guarding a rectangular partition. It would be nice to know whether there exists a coloring argument for this bound. In section 4 we suggest that such an argument may be found using the notion of face-respecting coloring introduced in [1].

A natural generalization of the above problem is the following problem (PVGP): Let R be a rectangular partition, and let $W \subseteq r(R)$ be a subset of rectangles. Is there a set of vertex guards of size k that guards all rectangles in W? A closely related problem is the following problem (SEGP): Let R be a rectangular partition, and let $E_A \subseteq e(R)$ be a subset of edges. Is there a set of size k of edge guards restricted to E_A, that guards all the rectangles of R? In Section 3 we show that both problems are NP-hard. Note that the status of the first problem for the case $W = r(R)$, and the second problem for the case $E_A = e(R)$, remains

*Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, dinitz@cs.bgu.ac.il.
†Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, matya@cs.bgu.ac.il.
‡Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, roikr@cs.bgu.ac.il.
Partially supported by the Lynn and William Frankel Center for Computer Sciences.

unknown.

In Section 2, we prove a theorem that is needed for the NP-hardness proof of SEGP.

2 A face-vertex matching result for planar graphs

In the full version of this paper we prove the following theorem.

Theorem 1 *Let G be a connected non-bipartite plane graph of vertex degree at most 3, and VC a vertex cover for G. Then, one can assign to each internal face of G a distinct vertex of VC, so that the vertex assigned to a face lies on its boundary.*

3 NP-hardness results

Uehara [5] proved that the problem of finding a vertex cover of minimum cardinality is NP-hard for 3-connected cubic planar graphs of girth greater than 3. In particular, since the vertex cover problem is polynomial-time solvable for bipartite graphs, it follows that the vertex cover problem remains NP-hard even when restricted to non-bipartite 3-connected cubic planar graphs of girth greater than 3.

Several definitions are needed before we can proceed. A *plane triangulation* is a 2-connected plane graph in which all faces (except possibly the outer face) are triangles (i.e., 3-cycles). If the outer face is also a triangle then the graph is called a *maximal plane triangulation*. A *separating triangle* in a plane graph is a 3-cycle which is not a face, i.e., a 3-cycle with a vertex in its interior and in its exterior. A *proper plane triangulation* is a plane triangulation that satisfies the following properties: (i) The outer face consists of at least 4 edges, (ii) all internal vertices have degree at least 3, and (iii) all cycles that are not faces are of length at least 4, i.e., there are no separating triangles.

Let G be a 2-connected plane graph. A *shortcut* in G is an edge between two vertices on the outer cycle of G, that is not an edge of this cycle. A *corner implying path* in G is a segment $v_1, v_2, \ldots, v_k$ of the outer cycle of G, such that (v_1, v_k) is a shortcut and $v_2, \ldots, v_{k-1}$ are not endpoints of any shortcut. Finally, for a graph G, its line graph $L(G)$ is constructed by representing each edge of G by a vertex, and drawing an edge between two vertices $u, v \in V(L(G))$ if and only if the edges of G represented by u and v share a common endpoint in G. It is well known that the line graph of a cubic graph is a 4-regular graph, and that the line graph of a cubic planar graph is also planar; see [3].

The following theorem was stated and proved by Kozminski and Kinnen [4].

Theorem 2 (Kozminski and Kinnen [4]) *Let G be a proper plane triangulation with at most four corner implying paths, then G admits a rectangular dual. The rectangular dual of G can be constructed in linear time.*

3.1 PVGP is NP-hard

We show that the following problem (which we named PVGP) is NP-hard. Let R be a rectangular partition, and let $W \subseteq r(R)$ be a subset of R's rectangles. Is there a vertex guarding set of size at most k for W?

The reduction. Let $G = (V, E)$ be a non-bipartite 3-connected cubic plane graph of girth greater than 3. Construct the line graph $L(G) = (V_L, E_L)$ of G. Since the line graph of a cubic planar graph is also planar, $L(G)$ is a planar graph. For each (internal) face of $L(G)$ whose boundary consists of four or more edges, triangulate it by creating a new vertex in the interior of the face and connecting this vertex to all vertices on the boundary of the face. Denote the resulting graph by $L_1(G) = (V_1, E_1)$, and let $V_H \subseteq V_1$ be the set of newly created vertices. Construct the rectangular dual, R_G, of $L_1(G)$. Let $H \subseteq r(R_G)$ be the subset of rectangles corresponding to the vertices in V_H. We call the rectangles in H *holes*, and put $W = r(R_G) - H$. See Figure 1.

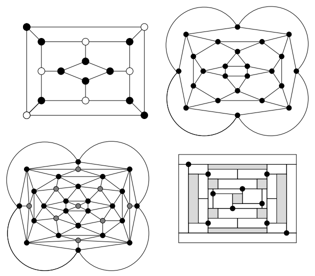


Figure 1: The reduction. Top left: a non-bipartite 3-connected cubic plane graph G of girth 4 with 16 vertices, 24 edges and 9 faces (excluding the outer face). The 9 vertices of a minimum vertex cover for G are in black. Top right: $L(G)$. Bottom left: $L_1(G)$; the vertices in V_H are in grey. Bottom right: R_G; the rectangles in H are in grey; and the 9 vertices of a minimum vertex guarding set for W are in black.

Our goal is to prove Lemma 6 below, but first we need to justify the last step of the reduction.

Lemma 3 *$L_1(G)$ is a proper plane triangulation with no corner implying paths. Thus according to Theorem 2 it has a rectangular dual.*

Proof. We already know that $L(G)$ is 4-regular and planar. By the definition of $L(G)$, since G is 3-connected, $L(G)$ is also 3-connected. Now, since G is triangle-free, there are exactly $|V(G)|$ triangles in

$L(G)$, created around each of the vertices of G. These triangles are empty, thus $L(G)$ has no separating triangles.

Now consider $L_1(G)$. From $L(G)$'s properties and since no separating triangles are introduced in the triangulation step, it follows that $L_1(G)$ is a 3-connected plane triangulation with no separating triangles. Moreover, $L_1(G)$ has no corner implying path, since the existence of such a path would contradict the 3-connectivity of $L_1(G)$. Thus, according to Theorem 2, $L_1(G)$ admits a rectangular dual. □

Definition 1 *Let R_G be the rectangular partition obtained by the reduction above. An (x, y)-meeting point of R_G is a vertex of R_G that is a meeting point of $x + y$ rectangles, of which exactly y are holes. Note that $1 \le x + y \le 3$ and $y \le 1$.*

Lemma 4 *(i) There is a natural one-to-one correspondence between the vertices of G and the (3,0)-meeting points of R_G. (ii) There is a natural one-to-one correspondence between the edges of G and the rectangles of W. (iii) There is a natural one-to-one correspondence between the internal faces of G and the holes of H.*

Following Lemma 4, it is easy to verify the following statements. (i) Each non-hole rectangle of R_G has exactly two (3,0)-meeting points on its boundary, (ii) no two (3,0)-meeting points are adjacent, (iii) a (2,1)-meeting point is adjacent to exactly one (3,0) meeting point, and (iv) a (2,0)-meeting point is adjacent to exactly one (3,0)-meeting point.

Lemma 5 *Let T be a vertex guarding set for W. Then the guards in T can be moved, so that all guards will lie on (3,0)-meeting points.*

Lemma 6 (Reduction correctness) *There exists a vertex cover of size at most k for G if and only if there exists a vertex guarding set of size at most k for W. Thus PVGP is NP-hard.*

Proof. $\Rightarrow$ Assume there exists a vertex cover VC of size k for G. For each $v \in VC$, place a guard, g_v, on the (3,0)-meeting point of R_G corresponding to v. Let $v \in VC$, and let e_1, e_2, e_3 be the edges of G adjacent to v. Clearly, g_v sees the 3 rectangles corresponding to the edges e_1, e_2, e_3. Now, since VC covers all edges of G, and the edges of G correspond to the rectangles of W, we are done.

$\Leftarrow$ Let T be a vertex guarding set of size k for R_G. From Lemma 5 we may assume that all guards in T are located at (3,0)-meeting points. Now simply put into VC all vertices corresponding to these (3,0)-meeting points. From the same considerations as above, VC is a vertex cover of size k for W. □

3.2 SEGP is NP-hard

We show that the following problem (which we named SEGP) is NP-hard. Let R be a rectangular partition, and let $E_A \subseteq e(R)$ be a subset of R's edges. Is there an edge guarding set restricted to E_A of size k that guards all rectangles of R?

The reduction. The reduction is a direct continuation of the one presented in Section 3.1. Consider the rectangular partition R_G obtained in that reduction. Let $E_F \subseteq e(R_G)$ be the set of all edges e, such that e is an edge either on the boundary of a hole (i.e., a rectangle in H), or e is on the outer boundary of R_G. E_F is the set of *forbidden* edges, i.e., all edges that cannot serve as edge guards. Put $E_A = e(R_G) - E_F$.

Lemma 7 *There exists an edge guarding set restricted to E_A of size at most k for R_G if and only if there exists a vertex guarding set of size at most k for W.*

Proof. $\Leftarrow$ Let T, $|T| \le k$, be a vertex guarding set for W. We describe how to construct an edge guarding set restricted to E_A of size at most k for R_G. By Lemma 5 we may assume that all guards in T are at (3,0)-meeting points. Let VC be the subset of vertices of G corresponding to T; by Lemma 6, VC is a vertex cover for G. Now, by Theorem 1, there exists a complete matching of $F(G)$, the set of internal faces of G, into VC, such that the vertex matched to a face lies on the face's boundary. Thus, applying Lemma 4, we obtain that there exists a complete matching from H into T, such that if a guard $g \in T$ is matched to a rectangle $h \in H$, then by transforming g into an edge guard, g can see h, in addition to the rectangles that it already sees. The guards of T that were not matched are transformed arbitrarily into edge guards. See Figure 2.

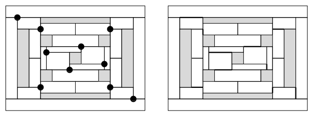


Figure 2: The connection between vertex guards and edge guards.

$\Rightarrow$ Let $T_E \subseteq E_A$, $|T_E| \le k$, be an edge guarding set for R_G. Let e be an edge in T_E. Since e is neither on the boundary of a hole, nor on the outer boundary of R_G, one of its endpoints must be a (3,0)-meeting point and the other must be a (2,i)-meeting point, $0 \le i \le 1$. Therefore, if we replace e by a vertex guard at the endpoint of e that is a (3,0)-meeting point, we only (possibly) lose sight of a hole. By doing this for each edge guard in T_E, we obtain a vertex guarding set of size at most k for W. □

4 Upper bounds

4.1 Face-respecting coloring

Let $G = (V, E)$ be a plane graph. A face-respecting k-coloring of G is a coloring of the vertices of G with k colors, such that each internal face has all k colors on its boundary. Note that adjacent vertices may have the same color.

Bose et al. [1] proved that any simple plane graph has a face-respecting 2-coloring. Obviously, since a rectangular partition is a plane graph, we have that every rectangular partition admits a face-respecting 2-coloring.

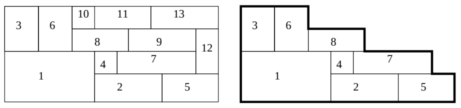


Figure 3: Left: A rectangular partition and a staircase ordering. The number in each rectangle is its rank in the ordering. Right: The staircase formed by the first eight rectangles in the ordering.

We show that every rectangular partition admits a face-respecting 3-coloring. Let R be a rectangular partition with n rectangles. A *staircase ordering* for R, is a (not necessarily unique) ordering $(R_1, R_2, \ldots, R_n)$ of the rectangles of R, such that for any $1 \leq i \leq n$, the boundary of the union of the rectangles $R_1, R_2, \ldots, R_i$ forms a staircase (see Figure 3). It is not difficult to show that every rectangular partition admits a staircase ordering. Now, using the staircase ordering for R, we present in the full version of this paper an incremental algorithm for computing a face-respecting 3-coloring of R. We thus obtain the following lemma.

Lemma 8 *Every rectangular partition admits a face-respecting 3-coloring.*

It remains, therefore, to consider face-respecting 4-colorings. For that we conjecture:

Conjecture 1 *Every rectangular partition admits a face-respecting 4-coloring.*

Despite considerable effort, we were not able to prove or disprove this conjecture. Besides its relevance to guarding, the conjecture is, so we believe, of independent interest.

4.2 Guarding rectangular partitions by edges

We would like to derive an upper bound on the number of edge guards needed to guard a rectangular partition of n rectangles. For that, we introduce the notion of a *suitable Hamiltonian path.*

Definition 2 *A Hamiltonian path (or cycle) in a plane triangulation G is suitable if it satisfies the following locality condition: Any 3 consecutive vertices on the path either belong to a single triangle or belong to two triangles that share an edge. That is, for any 3 consecutive vertices u, v, w on the path, there exist two triangles Δ_1, Δ_2 with a common edge, such that $u, v, w \in \Delta_1 \cup \Delta_2$. A Hamiltonian path (or cycle) is k-semi-suitable if there are at most k triples of consecutive vertices not satisfying the locality condition. Note that a suitable Hamiltonian path is a 0-semi-suitable Hamiltonian path.*

Lemma 9 *Let R be a rectangular partition, and let $D(R)$ be its dual graph. Let u, v, w be 3 vertices of $D(R)$ satisfying the locality condition. Then, the corresponding rectangles of u, v, w in R can be guarded with a single edge guard.*

Theorem 10 *Let R be a rectangular partition in which every rectangle is surrounded by at most five rectangles (considering the outer face as a rectangle). Then the dual graph of R (not including R's outer face) has a 4-semi-suitable Hamiltonian path. Thus R can be guarded with $\lceil \frac{n}{3} \rceil + 4$ guards.*

We suspect that every 4-connected maximal plane triangulation admits a suitable Hamiltonian path. If this is indeed true, then we can obtain a $\lceil \frac{n}{3} \rceil + 2$ upper bound on the number of edge guards that are always sufficient for guarding a rectangular partition.

References

[1] P. Bose, D. Kirkpatrick, and Z. Li. Worst-case-optimal algorithms for guarding planar graphs and polyhedral surfaces. *Computational Geometry: Theory and Applications*, 26(3), pp. 209–219, 2003.

[2] J. Czyzowicz, E. Rivera-Campo, N. Santoro, J. Urrutia, and J. Zaks. Guarding rectangular art galleries. *Discrete Mathematics*, to appear.

[3] R. Greenlaw and R. Petreschi. Cubic graphs. *ACM Computing Surveys*, 1995.

[4] K. Kozminski and E. Kinnen. An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits. In *Proc. 21st Design Automation Conf.*, pp. 655–656, 1984.

[5] R. Uehara. NP-complete problems on a 3-connected cubic planar graph and their applications. Tech. report TWCU-M-0004, Tokyo Woman's Christian University, 1996.

Carpenter's Rule Packings — A Lower Bound

Oliver Klein[*] Tobias Lenz[*]

Abstract

A carpenter's rule problem is considered, where a folding of a rule should be covered by a convex figure with diameter 1. The problem to compute this figure is known to be NP-hard and a $(1+\varepsilon)$-approximation exists, so we look at the worst-case bounds. An upper bound of 0.614 is known for the area of a convex figure with diameter 1 such that for any rule there is a folding which can be covered by this figure. The existing lower bound of 0.375 is increased in this paper to 0.475, narrowing the gap significantly.

1 Introduction

The problem of "folding" an object composed of rods and joints in a restricted environment was originally motivated by robot arm motion. Hopcroft, Joseph and Whitesides [2] have shown that even abstract simplifications are NP-hard. One of these abstract formulations is the carpenter's rule problem, i.e. minimizing the area of a convex polygon with diameter 1 covering at least one folding of any rule, which is NP-hard even for fixed dimension. Approximation algorithms exist, see Călinescu and Dumitrescu [1] and Hopcroft et al. [2], but the exact worst-case bound for the necessary area to cover a 2d-folding is still unknown.

2 The Problem

Definition 1 *A* carpenter's rule L *of length* n *is a chain of* n *oriented straight line segments* $s_1, \ldots, s_n$, *called* links, *with fixed lengths* $l_1, \ldots, l_n$, *where* $0 < l_i \leq 1$ *for* $1 \leq i \leq n$. *The end point of* s_i *is hinged to the start point of* s_{i+1} *for* $1 \leq i < n$.

Definition 2 *A* folding *of a carpenter's rule* L *of length* n *is a specification of all* $n-1$ *angles between the links of* L.

The problem at hand is to find a convex figure P of minimal area with diameter less or equal to 1, such that for all rules there exists a folding which is completely covered by P.

[*]Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany, {oklein,tlenz}@mi.fu-berlin.de

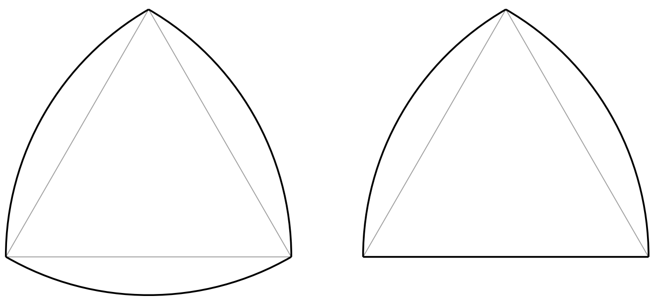


Figure 1: The left shape is a Reuleaux triangle (R3), the right shape is called R2.

3 Known Bounds

Dropping the diameter condition leads to an obvious solution with all angles equal to 0 or π. This gives a total area of 0 and Hopcroft et al. [2] showed that 2 is an upper and lower bound for the optimal length of such a flat folding. Therefore this is not applicable for our problem.

Călinescu and Dumitrescu [1] showed an upper bound of $\frac{\sqrt{3}}{4} \approx 0.614$ using the right shape in Figure 1 called R2, which is a Reuleaux triangle (left shape) with one circular arc removed. This is the best known upper bound.

A Reuleaux triangle can be constructed by taking an equilateral triangle and adding the circular arcs between two points, centered in the third point.

In the same paper a lower bound of $\frac{3}{8} = 0.375$ is derived from a rule of length 3 with $l_1 = l_3 = 1$ and $l_2 = \frac{\sqrt{7}-1}{2}$. The authors showed that for such a rule the area of the convex hull of every folding is at least 0.375. We will improve this bound considerably to approximately 0.476.

4 Upper Bound Lemma

Although this paper is focused on lower bounds, the following theorem may help in future research for better upper bounds. Figure 2 illustrates its proof, where L, R, T are the vertices, a_L, a_R are the circular arcs, e is the base edge and the dashed line is the rule. Please note, that the shape R2 has diameter 1.

Theorem 1 *No subset of R2 with smaller area than R2 covers all rules.*

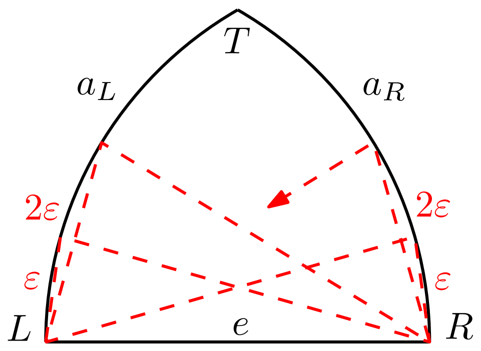


Figure 2: Covering virtually all of R2.

Proof. Consider a rule of length $4n+1$ with the link lengths $1, \varepsilon, 1, \varepsilon, 1, 2\varepsilon, 1, 2\varepsilon, 1, 3\varepsilon, \ldots, n\varepsilon, 1$ with $\varepsilon = \frac{1}{n+1}$. Now we try to embed this into R2 without leaving the shape. Every vertex must lie on a_L or a_R, because it is start or end point of a link of length 1 enclosed in a shape with diameter 1.

Starting the embedding on one of the arcs, we immediately end in L or R after the first 1, assume L. Since the upcoming ε segment must end on one of the arcs, its end is uniquely determined on a_L. The next segment of length 1 must end in R and again, the endpoint of the following ε segment is uniquely determined on a_R. This continues until the whole rule is embedded, climbing higher and higher on the arcs. The idea is illustrated as a red dashed line in Figure 2.

Assume that we start the embedding in R. From there we must go 1 to the arc a_L, L or T. If we go to T, the next 1 would lead to the case already discussed. Going to L leads directly to the above case. So we go to a_L and if the upcoming ε segment ends on a_L, a_R or T, we are again in the above case. Therefore we consider the case where we end up in L. Now we are back to the beginning, having consumed the first two links. Following this strategy describes the same embedding as the first case, but in a different order.

In the limit for $n \to \infty$ and therefore $\varepsilon \to 0$, this covers the whole shape of R2, except the point T and the segment e, but removing a single point and a line segment does not decrease the area. $\square$

5 Improving the Lower Bound

Consider a special rule with 5 segments of lengths $1, t, 1, t, 1$ and $t = \frac{\sqrt{5}-1}{2}$, the golden ratio. Folding this to minimal area with diameter at most 1 gives the desired result. The main challenge is to show that our folding is minimal. We define some nomenclature in Figure 3.

Using simple geometry, one can easily derive the following observation.

Observation 1 *All segments of length 1 must pairwise intersect.*

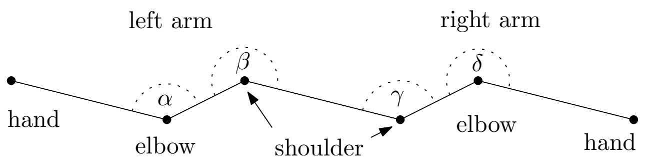


Figure 3: Giving things names: Parts of our special carpenter's rule.

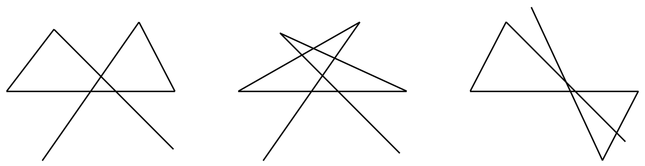


Figure 4: These are schematically the only distinct cases: both segments of length t point in the same direction from the middle segment and they do not cross (left), both segments of length t point in the same direction from the middle segment and they cross (middle), the segments of length t point in opposite directions from the middle segment (right).

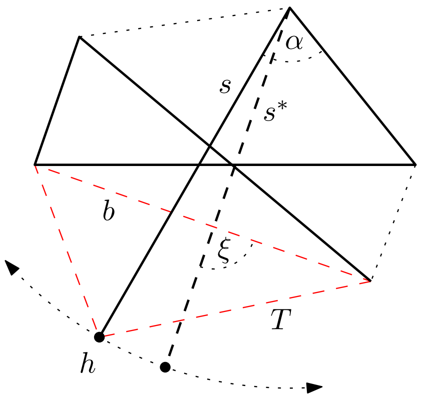


Figure 5: Only a single triangle of the convex hull area is influenced by an elbow angle, if the other angles are fixed.

Due to Observation 1 and the symmetry of the rule and its possible foldings, it is sufficient to consider the three cases depicted in Figure 4. W.l.o.g., let the end points of the middle segment be fixed to $(0,0)$ and $(1,0)$.

Lemma 2 *Fixing three angles and considering the area of the convex hull of the folding as a function $A(\alpha)$ in the fourth angle, A is strictly concave.*

Proof. Consider a $1, x, 1, y, 1$ ruler, x and y arbitrary, exemplarily shown in figure 5. Changing the elbow angle α rotates the hand h, thereby changing the convex hull. The only part of the convex hull dependent on α is the dashed triangle T. The base b of T does not change and the area of T is maximized for $s = s^*$. Let ξ denote the angle between s and b. Then, for

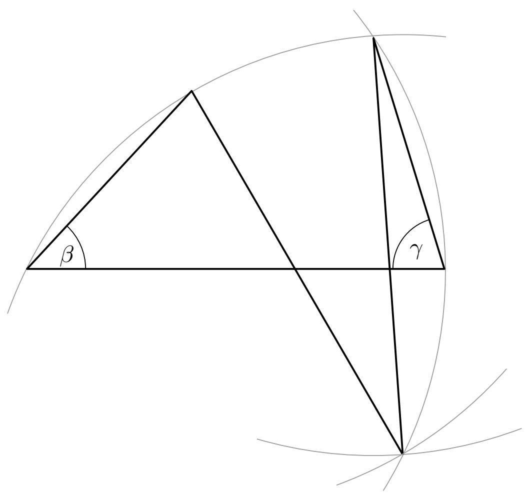


Figure 6: Boundary case: fixed γ and minimal β.

$s = s^*$ we have $\xi = \pi/2$. The area of T is composed of $\cos\xi$ and some constants and obviously $\xi \in [-\frac{\pi}{2}; \frac{\pi}{2}]$. The cosine is a concave function in this domain and therefore our area function, too.

Due to symmetry this holds for both elbows and analogous for the shoulder angles, which influence two triangles at once, but the sum of concave functions is again concave. □

The upcoming paragraphs focus on the first two cases from Figure 4 but for the third case, analogous statements hold.

The angles α and δ are uniquely determined by finding some angle at the boundary minimzing the area. Therefore, it follows by Lemma 2 that it suffices to have an area function A in two parameters (shoulder angles) and to search for the minimum at the boundary of the domain of $A(\beta,\gamma)$. Geometrically, for each fixed γ we look at the "extreme folding", meaning that β is uniquely chosen as small as possible, see Figure 6, resulting in an area formula solely dependent on γ. The other extreme case with β as large as possible, particularly $\beta > \gamma$, is omitted because due to symmetry $A(\beta,\gamma) = A(\gamma,\beta)$ holds and it thus suffices to check cases in which the second angle is not smaller than the first one.

Determining the area of these shapes depending on γ involves long and complicated terms computing two circle-circle intersections. For a lower bound on the area it suffices to look at a subset $A_s(\gamma)$ composed of only two triangles, shown gray in Figure 7. This area $A_s(\gamma)$ only depends on the angle γ and bounds the convex hull area from below. It is an approximation for the crossing and for the non-crossing case.

Depending on the link length t we get the following for the point coordinates.

$$c = 1 - t\cos\gamma$$
$$d = t\sin\gamma$$

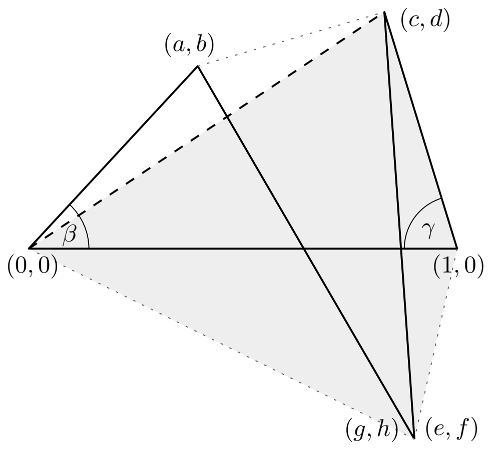


Figure 7: The real area A one has to compute (convex hull) and the lower bound A_s (gray shaded) excluding the top left triangle.

The point (g,h) is the intersection of the circle around $(0,0)$ with radius 1 and around (c,d) with radius 1. This leads to

$$g = \frac{c}{2} + d\sqrt{\frac{1}{c^2+d^2} - \frac{1}{4}}$$
$$h = \frac{d}{2} - c\sqrt{\frac{1}{c^2+d^2} - \frac{1}{4}}$$

Choosing $e = g$ and $f = h$ gives a valid folding and does not increase the area, so this must be optimal.

Using $t = \frac{\sqrt{5}-1}{2}$ and putting all this together in the area formula $A_s(\gamma) = \frac{d}{2} + \frac{h}{2}$ yields

$$A_s(\gamma) = \frac{\sqrt{5}-1}{8}\sin x + \frac{2+\left(1-\sqrt{5}\right)\cos x}{8}\sqrt{\frac{8}{p(x)} - 1}$$

with $p(x) = 5 - \sqrt{5} - 2\left(\sqrt{5}-1\right)\cos x$. This function, see Figure 8, should be minimized, so with standard means of differentiation we get

$$A'_s(\gamma) = \frac{\sqrt{5}-1}{8}\left(\cos x + \sin x\left(\sqrt{\frac{8}{p(x)} - 1} + \frac{8\left(\left(\sqrt{5}-1\right)\cos x - 2\right)}{p(x)^2\sqrt{\frac{8}{p(x)} - 1}}\right)\right).$$

Solving $A'_s(\gamma) = 0$ in the interval $\left(0; \frac{\pi}{2}\right)$ yields a single minimum of A_s which is smaller than the boundary values. This is attained for $\gamma^* = \arccos\frac{1+\sqrt{5}}{4}$. Fortunately this is exactly the case where $(a,b) = (c,d)$, so the skipped triangle has area zero, thus making the approximate area function an exact area function:

$$A(\gamma^*) = A_s(\gamma^*) = \frac{\sqrt{10+2\sqrt{5}}}{8} \approx 0.4755.$$

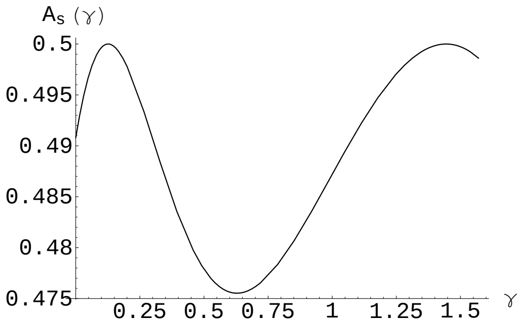


Figure 8: A plot of the simplified area function A_s.

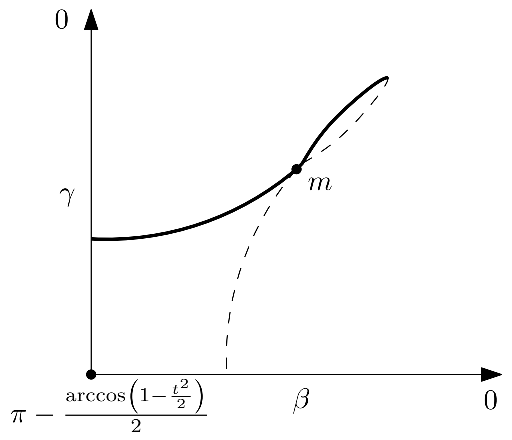


Figure 9: The boundary of the domain of $A(\beta, \gamma)$.

This case is depicted in the middle in Figure 10.

Schematically, the boundary of the area where $A(\beta, \gamma)$ is defined, looks like Figure 9. The fat part is the univariate function from the previous section, the dashed part is obtained due to symmetry. The computed minimum is in point m. One point is still missing in the considerations above—the origin in the diagram. Computing this separately with both angles equal to

$$\pi - \frac{\arccos\left(1 - \frac{t^2}{2}\right)}{2},$$

see left image in Figure 10, yields the same area as above. In fact, we choose t such that these areas are equal.

Due to space constraints, we skip the third case from Figure 4. The chain of arguments is similar to the discussed cases and the minimal area is again $\frac{\sqrt{10+2\sqrt{5}}}{8}$, which is depicted on the right in Figure 10.

This whole section proves our main theorem.

Theorem 3 *A convex shape with diameter at most* 1*, which covers at least one folding of every possible carpenter's rule, must have an area of at least*

$$\frac{\sqrt{10 + 2\sqrt{5}}}{8} \approx 0.4755.$$

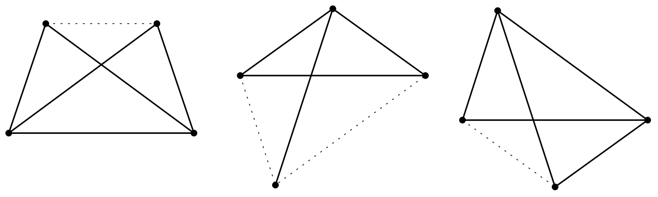


Figure 10: Three optimal foldings (solid lines) with minimal convex hull area (solid and dotted lines). From left to right, they correspond to the cases from figure 4.

6 Conclusion

The main contribution of this paper is the raise of the lower bound for the area necessary to cover at least one folding of every carpenter's rule. The existing gap between upper and lower bound was shrunken by approximately 40%, which is a significant improvement.

We naturally extended the lower bound proof of Călinescu and Dumitrescu [1] to longer rules. Cleverly chosen link lengths allow to omit a large case distinction due to high symmetry. Although the proof is somewhat lengthy, it is a nice interplay of geometric arguments and calculus and has its own beauty. Also the appearance of the golden ratio in this context indicates, that the problem bears some deeper, yet unidentified, structure.

Extending this approach to rules of length 7 does not give any new results without giving up the symmetry in the link lengths. Variable link lengths would imply much more complicated formulas and many distinct cases due to missing symmetry.

We believe that the true bound lies close to the known upper bound. Maybe the shape R2 is already the best possible for arbitrary rules. What remains is to prove it.

References

[1] G. Călinescu and A. Dumitrescu. The carpenter's ruler folding problem. *Combinatorial and Computational Geometry*, 52:155–166, 2005.

[2] J. E. Hopcroft, D. A. Joseph, and S. H. Whitesides. On the movement of robot arms in 2-dimensional bounded regions. *SIAM J. Comput.*, 14:315–333, 1985.

On the Exact Maximum Complexity of Minkowski Sums of Convex Polyhedra[*]

Efi Fogel[†] Dan Halperin[†] Christophe Weibel[‡]

Abstract

We present a tight bound on the exact maximum complexity of Minkowski sums of convex polyhedra in $\mathbb{R}^3$. In particular, we prove that the maximum number of facets of the Minkowski sum of two convex polyhedra with m and n facets respectively is bounded from above by $f(m,n) = 4mn - 9m - 9n + 26$. Given two positive integers m and n, we describe how to construct two convex polyhedra with m and n facets respectively, such that the number of facets of their Minkowski sum is exactly $f(m,n)$. We generalize the construction to yield a lower bound on the maximum complexity of Minkowski sums of many convex polyhedra in $\mathbb{R}^3$. That is, given k positive integers $m_1, m_2, \ldots, m_k$, we describe how to construct k convex polyhedra with corresponding number of facets, such that the number of facets of their Minkowski sum is $\sum_{1 \le i < j \le k}(2m_i - 5)(2m_j - 5) + \binom{k}{2} + \sum_{1 \le i \le k} m_i$. We also provide a conservative upper bound for the general case. The polyhedra models and an interactive program that computes their Minkowski sums and visualizes them can be downloaded from `http://www.cs.tau.ac.il/~efif/Mink`.

1 Introduction

Let P and Q be two compact convex polyhedra in $\mathbb{R}^d$. The Minkowski sum of P and Q is the convex polyhedron, polytope for short, $M = P \oplus Q = \{p + q \,|\, p \in P, q \in Q\}$.

Minkowski-sum computation constitutes a fundamental task in computational geometry. Minkowski sums are frequently used in areas such as robotics and motion planing [6, 8] and many additional domains, like solid modeling, design automation, manufacturing, assembly planning, virtual prototyping, etc., as Minkowski sums are closely related to proximity queries [7].

Various methods to compute the Minkowski sum of two polyhedra in $\mathbb{R}^3$ have been proposed. One common approach is to decompose each polyhedron into convex pieces, compute pairwise Minkowski sums of pieces of the two, and finally the union of the pairwise sums. Computing the Minkowski sum of two convex polyhedra remains a key operation. The combinatorial complexity of the sum can be as high as $\Theta(mn)$ when both polyhedra are convex.

Recently a few complete implementations of output-sensitive methods for computing exact Minkowski sums have need introduced: (i) a method based on Nef polyhedra embedded on the sphere [4], (ii) an implementation of Fukuda's algorithm by Weibel [2, 9], and (iii) a method based on the cubical Gaussian-map data structure [1]. These methods exploit efficient innovative techniques in the area of exact geometric-computing to minimize the time it takes to ensure exact results. However, even with the use of these techniques, the amortized time of a single arithmetic operation is large in comparison with a single arithmetic operation carried out on native number types, such as floating point. Thus, the constants involved in the expressions of these algorithm time complexities increases, which makes the question this paper attempts to answer, "What is the exact maximum complexity of Minkowski sums of polytopes in $\mathbb{R}^3$?", even more relevant.

Gritzmann and Sturmfels [5] formulated an upper bound on the number of features f_i^d of any given dimension i of the Minkowski sum of many polytopes in d dimensions. Fukuda and Weibel [3] obtained upper bounds on the number of edges and facets of the Minkowski sum of two polytopes in $\mathbb{R}^3$ in terms of the number of vertices of the summands: $f_2(P_1 \oplus P_2) \le f_0(P_1)f_0(P_2) + f_0(P_1) + f_0(P_2) - 6$. They also studied the properties of the Minkowski sums of perfectly centered polytopes and their polars, and provided a tight bound on the number of vertices of the sum of polytopes in any given dimension.

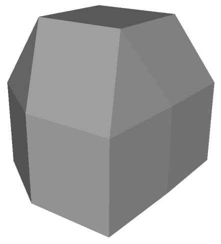
2 The Upper Bound

The *Gaussian Map* $G = G(P)$ of a compact convex polyhedron P in $\mathbb{R}^3$ is a set-valued function from P to the unit sphere $\mathbb{S}^2$, which assigns to each point p the set of outward unit normals to support planes to P at p. The overlay of the Gaussian maps of two polytopes P and Q respectively identifies all pairs of features of P and Q respectively that have common supporting

[*]This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes), by the Israel Science Foundation (grant no. 236/06), by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University, and by the Swiss National Science Foundation Project 200021-105202, "Polytopes, Matroids and Polynomial Systems".

[†]School of Computer Science, Tel-Aviv University, Israel. `{efif,danha}@post.tau.ac.il`

[‡]Mathematics Institute, EPFL, Laussane, Switzerland.`christophe.weibel@epfl.ch`


(a)

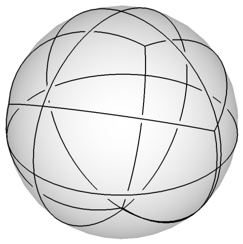

(b)

Figure 1: (a) The Minkowski sum of a tetrahedron and a cube and (b) the Gaussian map of the Minkowski sum.

planes, as they occupy the same space on the unit sphere, thus, identifying all the pairwise features that contribute to the boundary of the Minkowski sum of P and Q. A facet of the Minkowski sum is either a facet f of Q translated by a vertex of P supported by a plane parallel to f, or vice versa, or it is a facet parallel to two parallel planes supporting an edge of P and an edge of Q respectively. A vertex of the Minkowski sum is the sum of two vertices of P and Q respectively supported by parallel planes.

The number of facets of the Minkowski sum M of two polytopes P and Q with m and n facets respectively is equal to the number of vertices of the Gaussian map of M. A vertex in the Gaussian map of M is either due to a vertex in the Gaussian map of P, due to a vertex in the Gaussian map of Q, or due to an intersection between an edge of the Gaussian map of P and an edge of the Gaussian map of Q. Thus, the exact complexity $f(m,n)$ of M can be upper bounded by the expression $g(m,n) + m + n$, where $g(m,n)$ is the number of edge intersections in the Gaussian map of M.[1]

Corollary 1 *The maximum exact number of edges in a Gaussian map $G(P)$ of a polytope P with m facets is $3m - 6$. The exact number of faces in such a Gaussian map is $2m - 4$.*

The above can be obtained by a simple application of Euler's formula for planar graphs to the Gaussian maps $G(P)$. It can be trivially used to bound the exact number of facets of the Minkowski sum of two polytopes. We can plug the bound on the number of dual faces, which is the number of primal vertices, in the expression introduced by Fukuda and Weibel, (see Section 1), to obtain: $f(m,n) \leq (2m-4)\cdot(2n-4) + (2m-4) + (2n-4) - 6 = 4mn - 6m - 6n + 2$. We can improve the bound, but first we need to bound the number of faces in $G(M)$.

Lemma 2 *Let G_1 and G_2 be two Gaussian maps, and let G be their overlay. Let f_1, f_2, and f denote the number of faces of G_1, G_2, and G respectively. Then, the number of faces f cannot exceed $f_1 \cdot f_2$.*

[1]The exact complexity is strictly equal to the given expression, only when no degeneracies occur.

This lemma is similar to the one where convex planar maps replace the Gaussian maps, the proof of which appears in several flavors in the literature. We are ready to tackle the upper bound.

Theorem 3 *Let P and Q be two polytopes in $\mathbb{R}^3$ with m and n facets respectively, and let $f(m,n)$ denote the number of facets of their Minkowski sum $M = P \oplus Q$. Then, $f(m,n) \leq 4mn - 9m - 9n + 26$. The maximum complexity is attained only when the number of edges of each of P and Q is maximal for the given number of facets.*

Proof. Let v_1, e_1, f_1 and v_2, e_2, f_2 denote the number of vertices, edges, and faces of $G(P)$ and $G(Q)$ respectively. Recall that $v_1 = m$, $v_2 = n$, and $v = f(m,n)$, where v denotes the number of vertices of $G(M)$. The number of edges and faces of $G(M)$ is similarly denoted as e and f respectively. Assume that P and Q are two polytopes, such that the number of facets $f(m,n)$ of their Minkowski sum is maximal. First, we need to show that vertices of $G(P)$, vertices of $G(Q)$, and intersections between edges of $G(P)$ and edges of $G(Q)$ do not coincide. Assume to the contrary that some do. Then, one of the polytopes P or Q or both can be slightly rotated to escape this degeneracy, but this would increase the number of vertices $v = f(m,n)$, contradicting the fact that $f(m,n)$ is maximal. Therefore, the number of vertices v is exactly equal to $v_1 + v_2 + v_x$, where v_x denotes the number of intersections of edges of $G(P)$ and edges of $G(Q)$ in $G(M)$.

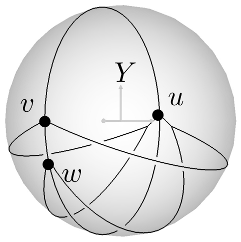
The total count of degrees of all vertices of $G(M)$ is twice the number of edges e of $G(M)$ on one hand, as each edge contributes two to this count. On the other hand, it is equal to the sum of degrees of all vertices of $G(P)$, vertices of $G(Q)$, and intersection vertices. Each edge of $G(P)$ and each edge of $G(Q)$ contributes exactly two to the count of degrees of the original vertices, and the degree of each new intersection is exactly four. Thus, we have $2e_1 + 2e_2 + 4v_x = 2e$. Applying Euler's formula and Lemma 2 yields $v_x \leq f_1 f_2 + v_1 + v_2 - 2 - e_1 - e_2$.

Corollary 1 sets an upper bound on the number of edges e_1. Thus, e_1 can be expressed in terms of ℓ_1, a non-negative integer, as follows: $e_1 = 3v_1 - 6 - \ell_1$. Applying Euler's formula, the number of facets can be expressed in terms of ℓ_1 as well: $f_1 = e_1 - 2 - v_1 = 2v_1 - 4 - \ell_1$. Similarly, we have $e_2 = 3v_2 - 6 - \ell_2$ and $f_2 = 2v_2 - 4 - \ell_2$ for some non-negative integer ℓ_2. $G(P)$ consists of a single connected component. Therefore, the number of edges e_1 must be at least $v_1 - 1$. This is used to obtain an upper bound on ℓ_1 as follows: $v_1 - 1 \leq e_1 = 3v_1 - 6 - \ell_1$, which implies $\ell_1 \leq 2v_1 - 5$, and similarly $\ell_2 \leq 2v_2 - 5$.

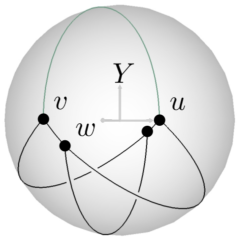
Plugging all this in the above inequality results with $v_x \leq 4v_1v_2 - 10v_1 - 10v_2 + 26$, and since $f(m,n) = v_1 +$

v_2+v_x, we conclude that $f(m,n) \leq 4v_1v_2-9v_1-9v_2+26$. The maximum complexity can be reached when $h(\ell_1,\ell_2)$ diminishes. This occurs when $\ell_1 = \ell_2 = 0$. That is, when the number of edges of $G(P)$ and $G(Q)$, (respectively P and Q), is maximal. □

3 The Lower Bound

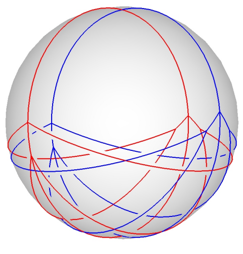


Given two integers $m \geq 4$ and $n \geq 4$, we describe how to construct two polytopes in $\mathbb{R}^3$ with m and n facets respectively, such that the number of facets of their Minkowski sum is exactly $4mn - 9m - 9n + 26$. More precisely, given i, we describe how to construct a skeleton of a polytope P_i with i facets, $3i - 6$ edges, and $2i - 4$ vertices, and prove that the number of facets of the Minkowski sum of P_m and P_n properly adjusted and oriented is exactly $4mn-9m-9n+26$. The figures above and below depict the Gaussian map of P_5 and P_4 respectively.



We use the subscript letter i in all notations X_i to identify some object X with the polytope P_i. For example, we give the Gaussian map $G(P_i)$ of P_i a shorter notation G_i First, we examine the structure of the Gaussian map G_i. Let V_i denote the set of vertices of G_i. Recall that the number of vertices, edges, and faces of G_i is i, $3i-6$, and $2i-4$ respectively. The unit sphere, where G_i is embedded on, is divided by the plane $y = 0$ into two hemispheres $H^- \subset \{(x, y, z) \,|\, y \leq 0\}$ and $H^+ \subset \{(x, y, z) \,|\, y > 0\}$. One vertex v_i is located exactly at the pole $(0, 0, 1)$. Another vertex w_i lies in H^- very close to v_i. A third vertex u_i is located very close to the opposite pole $(0, 0, -1)$. It is the only vertex (out of the i vertices) that lies in H^+. All the remaining $i - 3$ vertices in $V' = V_i \setminus \{u_i, v_i, w_i\}$ are concentrated near the pole $(0, 0, -1)$ and lie in H^-. The edge $\overline{uv}_i$ is the only edge whose interior is entirely contained in H^+. Every vertex in V' is connected by two edges to v_i and w_i respectively. These edges together with the edge $\overline{uw}_i$ form a set of $2i - 5$ edges, denoted as E'. The length of all edges in E' is almost π, due to the near proximity of u_i, v_i, and w_i to the respective poles.

It is easy to verify that if the polytope P_i is not degenerate; namely, its affine hull is 3-space, then any edge of G_i is strictly less than π long. Bearing this in mind, the main difficulty in arriving at a tight-bound construction is to force all edges but one of the Gaussian map of one polytope to intersect all edges but one of the Gaussian map of the other polytope, and on top of that force the pair of excluded edges, one from each Gaussian map, to intersect as well. As shown below, this is the best one can do in terms of intersections.



The number of facets in the Minkowski sum of P_m and P_n is maximal, when the number of vertices in the overlay of G_m and G_n is maximal. This occurs, for example, when one of G_m and G_n is rotated 90° about the Y axis, as depicted on the right for the case of $m = n = 5$. In this configuration, each edge of the $2m - 5$ edges in E'_m intersects each edge of the $2n - 5$ edges in E'_n. These intersections occur in H^-. In addition, the edge $\overline{uv}_m$ intersects the edge $\overline{uv}_n$ near the pole $(0, 1, 0)$. Counting all these intersections results with $(2m-5)(2n-5)+1 = 4mn - 10m - 10n + 26$. Adding the original vertices of G_m and G_n yields the desired result.

All the vertices of P_i lie on the boundary of a cylinder the axis of which coincides with the Z axis. When P_i is looked at from $z = \infty$, two facets are visible, and when looked at from $z = -\infty$, the remaining $i - 2$ facets are visible. The precise details that govern the construction of $P_i, i \geq 4$, which match the description of G_i above, are omitted due to lack of space.

4 Maximum Complexity of Minkowski Sums of Many Polytopes

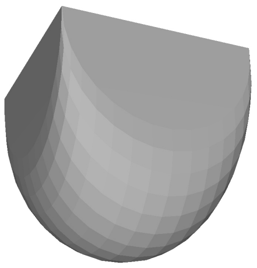
In this section we discuss the bounds on the exact complexity of the Minkowski sum many polytopes generalizing some of the arguments presented above.

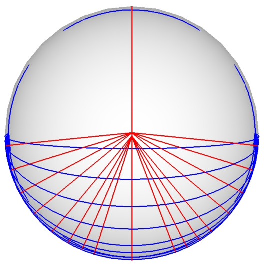
Conjecture 1 *Let $P_1, P_2, \ldots, P_k$ be a set of k polytopes in $\mathbb{R}^3$, such that the number of facets of P_i is m_i for $i = 1, 2, \ldots, k$. The exact maximum complexity of the Minkowski sum $P_1 \oplus P_2 \oplus \ldots \oplus P_k$ is $\sum_{1 \leq i<j \leq k}(2m_i - 5)(2m_j - 5) + \binom{k}{2} + \sum_{i=1}^{k} m_i$.*

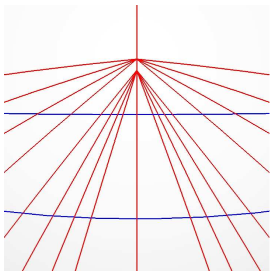
In the following sections we establish the lower bound, but provide only a conservative upper bound, which leaves a gap between the two bounds.

4.1 The Lower Bound

Given k positive integers $m_1, m_2, \ldots, m_k$, such that $m_i \geq 4$, we describe how to construct k polytopes in $\mathbb{R}^3$ with corresponding number of facets, such that the number of facets of their Minkowski sum is exactly $\sum_{1 \leq i<j \leq j}(2m_i - 5)(2m_j - 5) + \binom{k}{2} + \sum_{i=1}^{k} m_i$. More precisely, given i, we describe how to construct a skeleton of a polytope P_i with i facets, $3i - 6$ edges, and $2i-4$ vertices, and prove that the number of facets of the Minkowski sum $M = P_1 \oplus P_2 \oplus \ldots \oplus P_k$ of the k polytopes properly adjusted and oriented is exactly the expression above. We use the same construction described in Section 3.


(a)


(b)


(c)

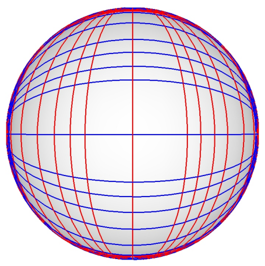
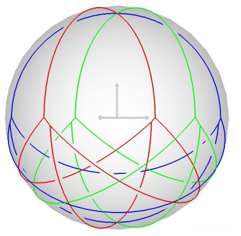

(d)

Figure 2: (a) The Minkowski sum $M_{11,11} = P_{11} \oplus P'_{11}$, where P'_{11} is P_{11} rotated 90° about the Y axis. (b) The Gaussian map of $M_{11,11}$ looked at from $z = \infty$. (c) A scaled up view of the Gaussian map of $M_{11,11}$ looked at from $z = \infty$. (d) The Gaussian map of $M_{11,11}$ looked at from $y = -\infty$.



The number of facets in the Minkowski sum of $P_i, i = 1, 2, \ldots, k$ is maximal, when the number of vertices in the overlay of $G_i, i = 1, 2, \ldots, k$ is maximal. This occurs, for example, when G_i is rotated $180^\circ i/k$ about the Y axis for $i = 1, 2, \ldots, k$, as depicted on the right for the case of $m_1 = m_2 = m_3 = 4$. In this configuration, all the $2m_i - 5$ edges in E'_i intersect all the $2m_j - 5$ edges in E'_j, for $1 \leq i < j \leq k$. These intersections occur in H^-. In addition, the edge $\overline{uv}_{m_i}$ intersects the edge $\overline{uv}_{m_j}$ for $1 \leq i < j \leq k$. These intersection points lie in H^+ near the pole $(0, 1, 0)$. Counting all these intersections results with $\sum_{1 \leq i < j \leq j} (2m_i - 5)(2m_j - 5) + \binom{k}{2}$. Adding the original vertices of $G(P_i), i = 1, 2, \ldots, k$, yields the bound asserted in Conjecture 4.

4.2 An Upper Bound

We apply a similar technique to the one used in Section 2 to obtain an upper bound. First, we extend Lemma 2.

Lemma 4 *Let $G_1, G_2, \ldots, G_k$ be a set of k Gaussian maps, and let G be their overlay. Let f_i denote the number of faces of G_i, and let f denote the number of faces of G. Then, the number of faces f of G cannot exceed $\sum_{1 \leq i < j \leq k} f_i \cdot f_j$.*

The proof of the lemma above is a simple generalization of the proof of lemma 2. Secondly, we count the total degrees of vertices in $G(M)$. Let $P_1, P_2, \ldots, P_k$ be k polytopes in $\mathbb{R}^3$ with $m_1, m_2, \ldots, m_k$ facets respectively. Let $G(P_i)$ denote the Gaussian map of P_i, and let v_i, e_i, and f_i denote the number of vertices, edges, and faces of $G(P_i)$ respectively. Let v_x denote the number of intersections of edges of $G(P_i)$ and edges of $G(P_j)$, $i \neq j$ in $G(M)$. Starting with $\sum_{i=1}^{k} e_i + 2v_x = e$, and applying Lemma 4 and Theorem 3 we get $v_x \leq \sum_{1 \leq i < j \leq k} (2v_i - 4)(2v_j - 4) - 2\sum_{i=1}^{k} v_i + 6k - 2$.

For example, the complexity of the Minkowski sum of k tetrahedra is $v_x + \sum_{i=1}^{k} v_i$, and by the inequality above it is bounded from above by $8k^2 - 6k - 2$. The construction described in the previous section yields a configuration of k tetrahedra, the Minkowski sum of which is $5k^2 - k$. For $k = 2$ both expressions evaluate to 18.

References

[1] E. Fogel and D. Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. In *Proc. 8th Workshop Alg. Eng. Exper. (Alenex'06)*, 2006.

[2] K. Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272, 2004.

[3] K. Fukuda and C. Weibel. On f-vectors of Minkowski additions of convex polytopes. *Discrete and Computational Geometry*, 2006. To appear.

[4] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3D selective Nef complexes: Data structure, algorithms, and implementation. In *Proc. 11th Annu. Euro. Sympos. Alg.*, volume 2832 of *LNCS*, pages 174–186. Springer-Verlag, 2003.

[5] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: Computational complexity and applications to Gröbner bases. *SIAM J. Disc. Math*, 6(2):246–269, 1993.

[6] D. Halperin, L. Kavraki, and J.-C. Latombe. Robotics. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 48, pages 1065–1093. CRC, 2004.

[7] M. C. Lin and D. Manocha. Collision and proximity queries. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 35, pages 787–807. CRC, 2004.

[8] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 47, pages 1037–1064. CRC, 2004.

[9] C. Weibel. Minkowski sums. `http://roso.epfl.ch/cw/poly/public.php`.

Covering points by axis parallel lines

Daya Ram Gaur[*] Binay Bhattacharya [†]

Abstract

We consider the problem of covering points in R^d ($d \geq 2$) by minimum number of axis parallel lines. The problem is NP-complete for all $d \geq 3$. We give a $d-1$ approximation algorithm based on a deterministic rounding of the optimal solution to a linear program.

1 Introduction

Given a set of n points in R^d the objective is to cover all the points using minimum number of axis parallel lines. We call this problem, the point cover problem. The point cover problem was first examined by Hassin and Megiddo [8], who showed that it can be solved in polynomial time in two dimensions, they also established that for all $d \geq 3$, the problem is NP-complete. A natural greedy algorithm is to pick the line that covers the maximum number of points in each iteration. The performance ratio for the greedy heuristic was shown to be $\geq \log n$ [8] even in two dimensions. On the other hand Johnson [10] and Chvatal [6] showed that the greedy heuristic has performance ratio at most $\log n$ for the set cover problem.

Consider the following algorithm, again from [8]: while there exists an uncovered point, pick all the lines that go through the point; Hassin and Megiddo noted that the previous algorithm has performance ratio d. We are not aware of any algorithm with performance ratio better than d for the point cover problem.

The variant of the problem where the lines can have arbitrary finite slopes also has a rich history. It is NP-complete even in two dimensions [13], and the corresponding optimization version is APX-hard [1]. Langerman and Morin [12] established that the problem is tractable in two dimensions when the number of lines in the optimal cover is small. Grantson and Levcopoulos [7] describe an approximation algorithm, and also give an improved exact algorithm (improvement over [12]), again the results are in two dimensions.

[*]Department of Math and Computer Science, University of Lethbridge, Lethbridge, AB, Canada, gaur@cs.uleth.ca

[†]School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, binay@cs.sfu.ca

2 Approximation algorithm

Let $x_1, \ldots, x_n$ be a set of points in R^d. Let L be the set of all the axis parallel lines going through the n points. $L(x)$ denotes the set of lines through point x. Note that $|L| \leq nd$. Next we describe an integer program for the point cover problem. Associated with each line $l \in L$ is a binary variable y_l whose value is 1 if the line is picked in the solution, 0 otherwise.

$$\text{IP: } \min \sum_{l \in L} y_l \tag{1}$$

$$\sum_{l \,:\, l \in L(x_i)} y_l \geq 1 \quad \forall\, x_i \tag{2}$$

$$y_l \in \{0,1\} \tag{3}$$

The linear programming relaxation to the integer program IP, obtained by replacing constraints of type (3) with non-negativity constraints $y_l \geq 0$, above is denoted LP. The linear programming dual of LP is:

$$\text{LP-dual: } \max \sum_{i=1}^{n} z_i \tag{4}$$

$$\sum_{i \,:\, l \in L(x_i)} z_i \leq 1 \quad \forall\, l \in L \tag{5}$$

$$z_i \geq 0 \tag{6}$$

Let y^* be the optimal solution to the linear program LP. $value(y^*)$ denotes the value of the optimal solution. y^* can be computed in $O(n^5)$ time using the the strongly polynomial algorithm of Tardos [17], as the coefficient matrix contains only 0s and 1s. Let A denote the coefficient matrix of the linear program LP with dimensions $n \times nd$. $A[i]$ denotes the i^{th} row in matrix A. Note that each row of A contains exactly d ones, all the other entries are 0 in the row.

Next we show that the d approximation algorithm in [8] can be visualized as a primal-dual algorithm. While there exists an uncovered point, the algorithm picks all the d axis parallel lines that go through the point. We construct an integral primal and an integral dual feasible solution to the linear programs as follows: set $y_l = 1$ if the line is picked by the algorithm else $y_l = 0$. Let x_i be the uncovered point picked in iteration j, then set $z_i = 1$. Dual feasibility requires that at most one uncovered point is picked from each line. Clearly, no two uncovered points picked in two different iterations, share a line that was picked (dual feasibility). The primal solution is feasible by construction. Furthermore, the value of the primal solution is at most d times the value of the dual solution, i.e. the performance ratio is d.

Another d approximation algorithm can be obtained by rounding the optimal solution to the linear program LP as shown in [9]. If y^* is the optimal solution to LP, then select all the lines l such that $y_l^* \geq \frac{1}{d}$. Each covering constraint has exactly d 1s, therefore at least one of the lines going through any point has value $\geq \frac{1}{d}$ in the optimal fractional solution y^*. Hence, the rounding gives an integral solution that is feasible. The rounding process increases the value of the solution at most d fold, therefore the performance ratio is d. Note that the primal-dual based algorithm is preferable over the rounding algorithm, as it takes $O(nd)$ time.

In two dimensions, the problem is equivalent to vertex cover in bipartite graphs as established in [8]. We give their reduction below: each line corresponds to a vertex, and each point correspond to an edge between the two vertices that correspond to the lines going through the point. A set of lines that cover all the points, corresponds to a vertex cover in the graph and vice-versa. In two dimensions a feasible primal solution to the integer program IP corresponds to a vertex cover, and an integral solution to the LP-dual corresponds to a matching.

Next we recall König–Egerváry theorem:

Theorem 1 (König–Egerváry) *The size of the minimum vertex cover is the same as the size of the maximum matching in a bipartite graph.*

This implies that the optimal solutions to the dual pairs of linear programs LP and LP-dual are integral and have the same value when $d = 2$. Therefore, the minimum vertex cover in bipartite graphs can be computed in polynomial time. For general graphs the vertex cover problem is known to be NP-Complete, and there is evidence that probably it is hard to approximate vertex cover strictly better than 2 [11]. Several approximation algorithms with performance ratio 2 (asymptotic) are known [2, 9, 4, 3, 14] for the vertex cover problem.

Next, we describe an application of the Local Ratio theorem of Bar-Yehuda and Even [3], (as noted in [15]) to improve the performance ratio of the novel algorithm due to Hochbaum [9]. The algorithm in [9] uses a result of Nemhauser and Trotter [16] to approximate vertex cover within a factor of $2 - \frac{2}{k}$ for graphs that are k-colourable. We begin with the theorem of Nemhauser and Trotter [16].

Theorem 2 (Nemhauser–Trotter [16]) *There exists an optimal solution to the linear program LP for the vertex cover problem which is half-integral (all the values are in the set $\{0, 1, \frac{1}{2}\}$).*

We assume that our graph is k-colourable. Given a half-integral optimal solution y^* to the linear programming relaxation LP of the vertex cover problem, let $V_0(V_1)$ be the set of vertices that are assigned value $\frac{1}{2}(1)$ respectively. First we note that each edge that does not have any vertex incident on V_1, has both its end-points in V_0 (else we do not have a fractional vertex cover).

Given a k-colouring of the vertices in V_0, let V' be the colour class of largest size ($\geq |V_0|/k$). We note that $V_1 \cup V_0 \setminus V'$ is a vertex cover. Size of the vertex cover thus computed is $|V_1 \cup V_0 \setminus V'| \leq |V_1| + |V_0|(1 - \frac{1}{k})$. The optimal fractional vertex cover is of size $|V_1| + \frac{|V_0|}{2}$. Therefore, we get the following theorem.

Theorem 3 *For k-colourable graphs, there exists a vertex cover of size $\leq (2 - \frac{2}{k})y^*$, where y^* is the optimal solution to the linear programming relaxation LP of the vertex cover problem.*

2.1 A first approximation

In this section we describe a, $d-1$-approximation algorithm for covering points with axis parallel lines in d dimensions.

Recall that A is the coefficient matrix for the linear programming relaxation to the point cover problem when each point lies in R^d. Furthermore, each row of A contains exactly d non-zero elements (ones). Let C_i be the indices of columns with non-zero entries in row i of the coefficient matrix A. Alternatively stated $C_i = \{j \mid A[i,j] = 1\}$. For each a, b & $a \neq b \in C_i$, let $A(a,b)[i]$ be a row vector derived from $A[i]$ by setting the $c^{th}, c \neq a, b$ column to 0, i.e, $A(a,b)[i,j] = A[i,j]$ if $j = a, b$; 0 otherwise, where $a, b \in C_i$.

Lemma 4 *For each row i in the coefficient matrix A there exists $a, b \in C_i$ such that $A(a,b)[i]\ y^* \geq \frac{2}{d}$.*

Proof. Each row of A contains d ones. Wlog, assume that the ones occur in the first d columns of row i. Assume that all for all $a, b \in [1..d]$

$$A(a,b)[i]y^* < \frac{2}{d}. \tag{7}$$

where y^* is the optimal solution to the linear program LP. Summing up the previous equation over all $a, b \in [1..d]$, we get

$$(d-1)A[i]y^* < \frac{2}{d}\binom{d}{2}$$
$$A[i]y^* < 1$$

This is a contradiction as y^* is a feasible solution in LP. □

Theorem 5 *The point cover problem in R^d can be approximated within a factor of $d-1$ (in $O(n^5)$ time assuming constant d).*

Proof. Let A' be the matrix whose i^{th} row $A'[i] = A(a,b)[i]$ such that $A(a,b)[i]\ y^* \geq \frac{2}{d}$. At least one pair (a,b) with the desired property exists for each row due to Lemma 4. If for row i more than one pair (a,b) exists then we pick one arbitrarily.

Consider the following integer program:

$$\text{IR: } \min y \tag{8}$$

$$A'y \geq 1 \tag{9}$$

$$y_i \in \{0,1\} \tag{10}$$

y is a column vector with nd entries, each row of A' contains exactly two non-zero elements (1s). Integer program IR describes the vertex cover problem for a graph whose vertices are the elements of vector y, and edges corresponds to pairs of non-zero elements in each row of A'. Furthermore, this graph is d-colourable; assign lines associated with each dimension a unique colour. For instance all the lines parallel to x-axis get color 1, and so on.

Given the linear programming relaxation LP to the point cover problem, construct the linear program IR as shown above. Let LPR be the linear programming relaxation to the integer program IR. If y^* is the fractional optimal solution to LP then $\frac{dy^*}{2}$ is feasible in LPR. Let y" be the optimal solution to LPR. Then $y" \leq \frac{dy^*}{2}$. By Theorem 3 we can find an integral vertex cover (to the problem described by integer program IR) of size $\leq (2-\frac{2}{d})y" \leq \frac{(2(d-1)}{d}\frac{dy^*}{2}$. Therefore the performance ratio $\leq d-1$.

One can use either the algorithm of Nemhauser and Trotter [16] or the algorithm due to Bourjolly and Pulleyblank [5] to compute a half-integral fractional vertex cover in $O(|E||V|)$ time. The d-colouring can be computed in $O(|V|)$ time. Therefore, the algorithm in Theorem 3 can be implemented in $O(|E||V|)$ time. As there are n rows in the coefficient matrix A', we have n edges in the graph over nd vertices. As our coefficient matrix A contains only 0s and 1s we can use the strongly polynomial algorithm of Tardos [17] with running time $O(n^5)$ to obtain the optimal linear programming solution y^* (assuming constant d). Hence, we can compute a $d-1$-approximate solution to the point cover problem in d dimensions in $O(n^5)$ time. Note that all the steps, except the first step required to compute y^*, can be implemented in $O(n^2)$ time.

□

3 Discussion

We consider the problem of covering points in R^d with minimum number of axis parallel lines. This problem is known to be NP-complete for all $d \geq 3$, and the current, long standing and obvious, bound on the performance ratio is d [8].

We give a $d-1$ approximation algorithm. Our algorithm is based on deterministic rounding of the optimal solution to a linear program, and relies heavily on the approximation algorithm for the vertex cover problem. It would be interesting to see how far can we push this technique, whether it is possible to obtain a $\frac{d}{c}+O(1)$ approximation algorithm for any constant c using the approach presented here. Also interesting would be a combinatorial algorithm for computing the optimal solution y^* to the linear programming relaxation.

References

[1] V. S. Anil Kumar, A Sunil Arya, and A H. Ramesh, *Hardness of Set Cover with Intersection 1,* Proceedings of the 27th International Colloquium on Automata, Languages and Programming, (2000), 624–635.

[2] R. Bar-Yehuda, and S. Even, *A linear time algorithm for the weighted vertex cover problem,* Journal of Algorithms, 2 (1981), 198–203.

[3] R. Bar-Yehuda, and S. Even, *A local ratio theorem for approximating the weighted vertex cover problem,* Annals of Discrete Math, 25 (1985), 27–45.

[4] K. L. Clarkson, *A modification of the greedy algorithm for vertex cover,* Information Processing Letters, 16 (1983), 23–25.

[5] J. M. Bourjolly and W. R. Pulleyblank, *König-Egerváry graphs, 2-bicritical graphs and fractional matchings,* Discrete Applied Mathematics, 24 (1989) 63–82.

[6] V. Chvatal, *A greedy heuristic for the set-covering problem,* Math. Oper. Res. 4 (1979) 233–235.

[7] M. Grantson and C. Levcopoulos, *Covering a Set of Points with a Minimum Number of Lines,* CIAC (2006) 6–17, also in EWCG, Delphi, 2006.

[8] R. Hassin and N. Megiddo, *Approximation algorithms for hitting objects with straight lines,* Discrete Appl. Math. 30(1) (1991) 29–42.

[9] D. S. Hochbaum, *Approximation Algorithms for the Set Covering and Vertex Cover Problems,* SIAM J. Comput. 11(3) (1982) 555-556.

[10] D.S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci. 9 (1974) 256–278.

[11] S. Khot and O. Regev, *Vertex Cover Might be Hard to Approximate to within* $2-\varepsilon$, IEEE Conference on Computational Complexity (2003) 379–386.

[12] S. Langerman, P. Morin, *Covering Things with Things,* Discrete & Computational Geometry 33(4) (2005) 717–729.

[13] N. Megiddo and A. Tamir, *On the complexity of locating linear facilities in the plane,* Oper. Res. Lett., 1 (1982) 194–197.

[14] B. Monien, and E. Speckenmeyer, *Ramsey numbers and an approximation algorithm for the vertex cover problem*, Acta Informatica, 22 (1985), 115–123.

[15] R. Motwani, *Lecture Notes on Approximation Algorithms – Vol I.*

[16] G. L. Nemhauser and L. E. Trotter, Jr., *Vertex Packing: Structural Properties and Algorithms,* Mathematical Programming, 8 (1975), 232–248.

[17] E. Tardos, *A strongly polynomial algorithm to solve combinatorial linear programs,* Oper. Res. 34 (1986), 250–256.

Inflating the Cube by Shrinking

Kevin Buchin* Igor Pak† André Schulz*

Abstract

We present a continuous, submetric deformation of the surface of the cube which increases the enclosed volume by about 25.67%. This improves the previous bound of about 21.87% by Bleecker

1 Introduction

We address the problem how large the volume of a body with a surface isometric to that of the unit cube can be. The idea of considering volume-increasing isometric embeddings is due to Bleecker [2]. He proved that a volume-increasing continuous isometric deformation exists for every simplicial convex surface in $\mathbb{R}^3$. A deformation is called isometric if it preserves the geodesic distances between any two points on the surface. Bleecker also gives a direct construction for the cube and other platonic solids. By Alexandrov's uniqueness theorem [1] a body resulting from such a deformation must be non-convex.

Most recently, Pak [6] gave an easy construction for increasing the volume of the unit cube to about 1.1812 based on the work of Milka [4]. Bleecker's more involved construction yields a volume of about 1.2187. A simple upper bound can be obtained by the volume of the sphere which has the same surface as the cube. This gives an upper bound on the volume of 1.3820. This bound is not sharp as the cones around cube vertices are not isometric to spherical regions.

Bleecker conjectured that for every (not necessarily simplicial) polyhedron $P \subset \mathbb{R}^3$ there exists a volume-increasing deformation of ∂P [2]. Bleecker's conjecture was positively resolved by Pak [5], who also extended it to non-convex polyhedra and polyhedra in higher dimensions.

It was observed by Pak [6] that one can also consider submetric embeddings, a larger class containing the isometric embeddings. In a submetric embedding geodesic distances on the surface are non-increasing. By a result of Burago and Zalgaller [3], for every submetric embedding there is an isometric embedding arbitrary close to it. Thus the bound achieved by submetric embeddings coincides with that by isometric embeddings.

In this paper, we present a shrinking, i.e., a continuous, submetric deformation of the unit cube for which the resulting volume is at least 1.2567. This also improves the lower bound on the volume of a surface isometric to that of a unit cube. The shrinking problem and the idea of looking at shrinkings in order to get isometric embeddings is due to Pak [5].

2 A Shrinking of the Cube with Large Volume

We present volume-increasing submetric embeddings of the cube. The embeddings are parametrized by $\varepsilon \in [0, 0.5]$. Increasing ε from zero yields a continuous deformation. In this section we show a construction with volume about 1.2444, which we improve in the next section further.

Our approach is a refinement of Igor Pak's work [5, 6]. The original cube is given as the convex hull of the set $\{0, 1\}^3$. We denote vertices on the surface of the cube by p_i. The same vertex in the deformed cube is denoted as v_i.

As a first step we cut off ε-cubes in every corner of the cube (see Figure 1) Now we are going to deform

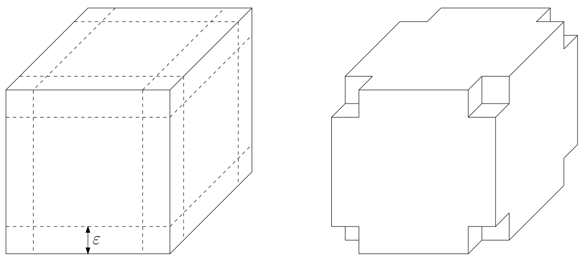


Figure 1: Cutting off ε cubes.

the remaining part of the cube. We place one vertex in the middle of every ε segment as shown in Figure 2.a. The segments defined between $p_1, p_{5/4}, p_{3/2}, p_{7/4}, p_2$ have the length $\varepsilon/2$. Let the framework induced by this chain be C. We move the vertices of C such that $v_1, \ldots, v_2$ lie on a quarter-circle (depicted in Figure 2.b). We apply the deformation for all corresponding pairs of ε segments. This leads to a body which we divide into a *corpus* and 12 *bars*. Figure 3 shows the parts. A bar is a prism with a 6-gon as base area. The 6-gon is inscribed in quarter circle. Its shorter edges have the length $\varepsilon/2$. The radius of the quarter circle is denoted by δ. Expressed in terms of ε we

*Institut für Informatik, Freie Universität Berlin, [buchin|schulza]@inf.fu-berlin.de

†Department of Mathematics, Massachusetts Institute of Technology, pak@math.mit.edu

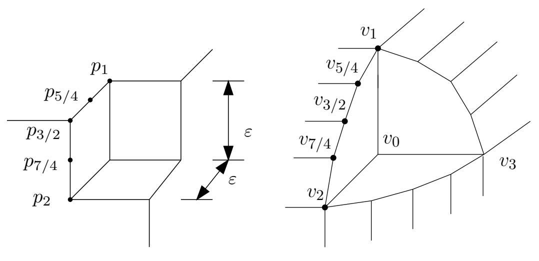


Figure 2: Bending the chains induced by a pair of ε segments.

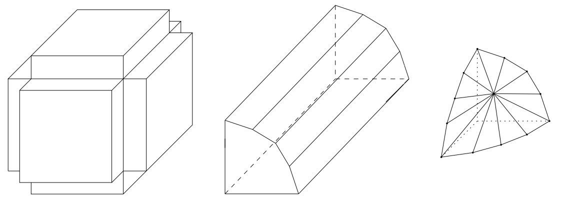


Figure 3: Corpus and bar and star.

obtain

$$\delta = \frac{1}{2}\varepsilon\sqrt{\frac{1}{2-2\cos(\pi/8)}}.$$

The volume of one bar is given by

$$V_{bar} = 2(1-2\varepsilon)\delta^2\sin(\pi/8).$$

The corpus is the remaining part after cutting out the bars. Its volume is given by

$$V_{corpus} = (1-2\varepsilon)^3 \; + \; 6\delta(1-2\varepsilon)^2.$$

It remains to place the cut-outs at the corners of the body. We have to deform the ε-cubes, such that they fit into the open 12-gons (formed by three chains C) of the body. Consider the open part depicted in Figure 2.b. One vertex is part of the corpus and the three bars, which we denote by v_0. In the following we refer to an orthogonal coordinate system. Its origin lies at v_0 and its x,y, and z directions are defined by the rays passing through v_1, v_2, and v_3. The object we glue into this part is called *star*. It is defined as the convex hull of the vertices on the chains between v_1, v_2, v_3 together with v_0 and a vertex v_*. The coordinate of v_* is chosen in such a way, that the embedding is submetric. We place v_* on a line given by $x = y = z$. The condition for a submetric embedding is fulfilled if no distance is enlarged. The crucial distances are obtained in the original cube between $p_1, p_{5/4}$ and $p_{3/2}$ and the original corner vertex of the cube p_c. All other distances which occur are symmetric variants of these distances. Therefore we have to choose $v_* = (a, a, a)$ such that the following conditions hold.

$$\begin{aligned} \|p_1 - p_c\| &= \sqrt{2}\varepsilon &\geq \|v_1 - v_c\| \\ \|p_{5/4} - p_c\| &= \tfrac{\sqrt{5}}{2}\varepsilon &\geq \|v_{5/4} - v_c\| \\ \|p_{3/2} - p_c\| &= \varepsilon &\geq \|v_{3/2} - v_c\| \end{aligned}$$

To compute the distances we need the coordinates of $v_1, v_{5/4}$ and $v_{3/2}$ in the specified coordinate system which are

$$\begin{aligned} v_1 &= (0, \delta, 0), \\ v_{5/4} &= (\delta\sin(\pi/8), \delta\cos(\pi/8), 0), \\ v_{3/2} &= \left(\delta/\sqrt{2}, \delta/\sqrt{2}, 0\right). \end{aligned}$$

We are left with three equation systems which lead to different upper bounds on a. It turns out that the smallest feasible solution for a is obtained by the distance between v_1 and v_*, namely $0.976468\,\varepsilon$. If we set v_* to $(0.9764\,\varepsilon, 0.9764\,\varepsilon, 0.9764\,\varepsilon)$ all distances decrease.

Finally, we have to evaluate the volume of the stars. Each star is divided into tetrahedra. There are two types of tetrahedra, one is given by the convex hull of $v_0, v_1, v_{5/4}, v_*$ and the other by the convex hull of $v_0, v_{5/4}, v_{3/2}, v_*$. Both tetrahedra appear 6 times in every star. That leads to the following expression for the volume of a star;

$$V_{star} = 1.227259706\varepsilon^3.$$

Now we can evaluate the volume of the complete body which is

$$V = V_{corpus} + 12V_{bar} + 8V_{star}.$$

See Figure 4 for the graph of $V(\varepsilon)$ for the feasible values of ε. The volume $V(\varepsilon)$ is maximized at about $\varepsilon_0 = 0.351311$, which induces a volume $V(\varepsilon_0) = 1.2444$. Thus, this improves the bound of Bleecker [2]. The deformed cube for this value of ε is shown in Figure 5. Each star has 3 concave edges depicted as dashed lines. In the next section we refine our construction.

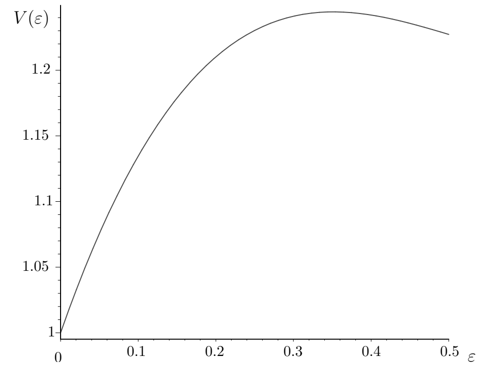


Figure 4: The volume of the deformed cube in terms of ε.

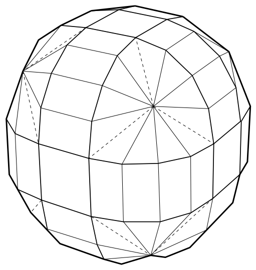


Figure 5: Deformed cube.

3 A Refined Construction

We refine the construction to increase the volume of the cube. A crucial part of the construction was to take two adjacent edges of length ε and turn them into a chain of 4 edges of length $\varepsilon/2$. The deformation puts all vertices of the chain on a quarter circle with radius δ. In the previous section the chain C contained 5 vertices. If we increase the number of vertices on C the deformed cube becomes more "spherish", promising a larger volume. In the limit C is a spherical arc. In the following, we consider this situation.

The value of δ is the radius of a circle with perimeter 8π, therefore

$$\delta = \frac{4\varepsilon}{\pi}.$$

The volume of the corpus is the same as calculated in Section 2. Every bar is a prism with a quarter circle of radius δ as base area. This leads to

$$V_{bar} = \frac{1}{4}(1-2\varepsilon)\pi\delta^2.$$

The stars consist of three equally sized quarter cones. The base area coincides with the base area of the bars. The height of the quarter cones is given by a. The value of a has to be chosen in such a way that the embedding is submetric. We consider the point p_x on the C (See Figure 6). Let its distance from p_1 be x. For the deformed cube we consider the same co-

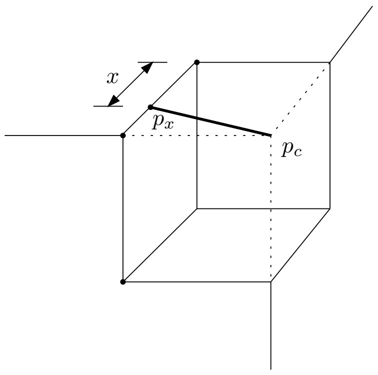


Figure 6: The point p_x.

ordinate system like in Section 2. The coordinates of the point v_x are $(\delta\cos(x/\delta), \delta\sin(x/\delta), 0)$. The distance between p_x and p_c (depicted in Figure 6) equals $\sqrt{(\varepsilon - x)^2 + \varepsilon^2}$. This leads to the following expression for the submetric condition:

$$(\varepsilon - x)^2 + \varepsilon^2 \geq (a - \cos(x/\delta))^2 + (a - \delta\sin(x/\delta))^2 + a^2.$$

The inequality holds with equality if,

$$a(x,\varepsilon) = \tfrac{1}{3\pi}\Bigg(4\cos\left(\tfrac{x\pi}{4\varepsilon}\right)\varepsilon + 4\sin\left(\tfrac{x\pi}{4\varepsilon}\right)\varepsilon + \sqrt{32\varepsilon^2(\cos\left(\tfrac{x\pi}{4\varepsilon}\right)\sin\left(\tfrac{x\pi}{4\varepsilon}\right) - 1) + 3\pi^2(x^2 + 2\varepsilon^2 - 2x\varepsilon)}\Bigg)$$

The variable a depends on x and ε. We choose x as a multiple of ε. Minimizing the expression over all $x \in [0,1]\varepsilon$ yields a value for a of about $a = 0.9772\ \varepsilon$ which is obtained at about $x = 0.1144\ \varepsilon$. Therefore we can describe the volume of the star by

$$v_{star} = \frac{1}{4}\ 0.9772\ \varepsilon\ \delta^2\pi.$$

Finally we maximize the volume of the whole body (1 corpus, 12 bars, 8 stars) over $\varepsilon \in [0, 0.5]$. It turns out that the maximum is at least 1.2567 which is obtained at about $\varepsilon = 0.37712$.

4 Future Work

Our construction leads to a non-convex body. Due to Alexandrov [1] we know that there exist no convex isometric embedding for a convex polytope with larger volume. It would be interesting to convexify our construction to find a submetric embedding for the cube, which is convex and has largest possible volume. The example given in [6] gives a convex polyhedral construction with a volume only a little large than 1. Related to this question is a conjecture, posed in [6]:

Conjecture 1 (Pak) *Let S_0 be a convex polyhedral surface in $\mathbb{R}^3$ and let S_1 be a convex polyhedral surface submetric to S_0 of greater volume. Then there exists a volume increasing shrinking from S_0 to S_1.*

Since volume-increasing shrinkings exist for polyhedral surfaces in any dimension [6], one can ask for shrinkings for hypercubes. What ratios of the volume can be obtained in dimensions larger than 3? What is the relation between the ratio and the dimension? Notice that if we mimic the construction given in Section 3 in $\mathbb{R}^2$, we end up with a circle, which matches the upper bound.

We have concentrated in our paper on shrinkings of the cube. Our technique is applicable to other surfaces as well. However the computations for other interesting bodies (like platonic solids) still has to be done.

Acknowledgments

The first and third author would like to thank Günter Rote for suggesting the problem to them. The second author was partially supported by the NSF.

References

[1] A. D. Alexandrov. *Convex polyhedra*. Springer Monographs in Mathematics. Springer, 2005. Original Russian edition published by Gosudarstv. Izdat. Tekhn.-Teor. Lit., Moscow-Leningrad, 1950.

[2] D. D. Bleecker. Volume increasing isometric deformations of convex polyhedra. *J. Diff. Geom.*, 43(3):505–526, 1996.

[3] Y. D. Burago and V. A. Zalgaller. Isometric piecewise-linear embeddings of two-dimensional manifolds with a polyhedral metric into $\mathbb{R}^3$. *St. Petersburg Math. J.*, 7:369–385, 1996.

[4] A. D. Milka. Linear bendings of regular convex polyhedra (in Russian). *Mat. Fiz. Anal. Geom.*, 1:116–130, 1994.

[5] I. Pak. Inflating polyhedral surfaces. *preprint*, 2006. 37 pp.

[6] I. Pak. Inflating the cube without stretching. *preprint*, 2006. 3 pp. to appear in *Amer. Math. Monthly.*

Hamiltonian Tetrahedralizations with Steiner Points

Francisco Escalona[*] Ruy Fabila-Monroy[†] Jorge Urrutia [†]

Abstract

A tetrahedralization of a point set in 3-dimensional space is Hamiltonian if its dual graph has a Hamiltonian cycle. Let S be a set of n points in general position in 3-dimensional space. We prove that by adding to S at most $\lfloor \frac{m-2}{2} \rfloor$ Steiner points in the interior of the convex hull of S, we obtain a point set that admits a Hamiltonian tetrahedralization. We also obtain an $O(m^{\frac{3}{2}}) + O(n \log n)$ algorithm to solve this problem, where m is the number of elements of S on its convex hull. We also prove that point sets with at most 20 convex hull points have a Hamiltonian tetrahedralization without the addition of any Steiner points.

1 Introduction

Let S be a set of n points in $\mathbb{R}^3$ in general position. The convex hull of S ($Conv(S)$) is the intersection of all convex sets containing S.

The points of S lying on the boundary of $Conv(S)$ are called convex points and the points lying in the interior of $Conv(S)$, interior points.

A tetrahedralization $\mathcal{T}$ of S is a partition of $Conv(S)$ into tetrahedra with vertices in S such that:

1. The tetrahedra only intersect at points, lines or faces.

2. The tetrahedra do not contain points of S in their interior.

In a similar way, a triangulation of a point set in the plane is a partition of its convex hull into triangles satisfying the above properties.

Given a tetrahedralization $\mathcal{T}$ of S, we define $D_{\mathcal{T}}$, the dual graph of $\mathcal{T}$, to be the graph whose vertex set is the tetrahedra of $\mathcal{T}$, two of which are adjacent if and only if they share a common face.

In this paper, we are interested in tetrahedralizations such that their dual graph contains a Hamiltonian cycle or path. In general, we call such tetrahedralizations Hamiltonian tetrahedralizations. To differentiate between cycles and paths, we write Hamiltonian cycle and Hamiltonian path tetrahedralizations. We say that S admits a Hamiltonian tetrahedralization if there exists a Hamiltonian tetrahedralization of S.

A well known problem in computational geometry (see [5], Problem 29) asks if every convex polytope in $\mathbb{R}^3$ admits a Hamiltonian tetrahedralization, that is, a tetrahedralization of the set of vertices of the polytope.

The question was raised in [1], where a Hamiltonian triangulation was sought; in that paper the same problem was solved in the plane. In [3] it was proved that triangulations produced by applying Graham's Scan to calculate the convex hull of point sets are Hamiltonian.

It was observed in [1] that Hamiltonian triangulations allow for faster rendering of triangular meshes. The same holds true for tetrahedra. In [1], the problem of finding a Hamiltonian tetrahedralization for a convex polytope in $\mathbb{R}^3$ was conjectured to be NP-complete.

The existence of a Hamiltonian tetrahedralization of a convex polytope remains open. In this paper we study the following related problem: Given a convex polytope P in $\mathbb{R}^3$, how many Steiner points must be placed in the interior such that the set of vertices of P together with the added Steiner points admits a Hamiltonian tetrahedralization?

We consider the more general case and consider point sets rather than convex polytopes. Let S be a set of n points in $\mathbb{R}^3$ in general position such that its convex hull contains m vertices, and let m' be the number of S that belong to the interior of $Conv(S)$.

We present an algorithm that adds at most $\lfloor \frac{m-2}{2} \rfloor$ Steiner points, located in the interior of $Conv(S)$, to S. Our algorithm produces a Hamiltonian tetrahedralization. The overall complexity of the algorithm is $O(m^{\frac{3}{2}}) + O(n \log n)$.

Finally we show that if $m \leq 20$, no Steiner points need to be added.

2 The algorithm

The main idea is to first add a point to S to obtain a tetrahedralization such that its dual graph can be partitioned into cycles.

We then insert Steiner points to join existing cycles. We continue this process until the cycle partition consists of just one cycle. This final cycle is a Hamiltonian

[*]Facultad de Ciencias, Universidad Nacional Autónoma de México

[†]Instituto de Matemáticas, Universidad Nacional Autónoma de México (`ruy@ciencias.unam.mx`, `urrutia@math.unam.mx`). Supported by CONACYT of Mexico, Proyecto SEP-2004-Co1-45876, and PAPIIT (UNAM), Proyecto IN110802.

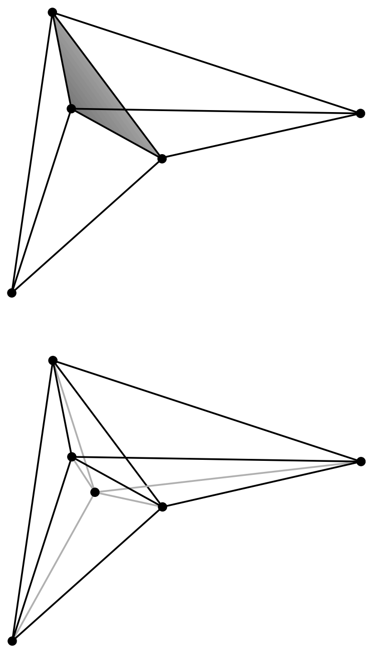


Figure 1: Join Operation.

cycle in the dual graph of the final tetrahedralization.

Actually, we first remove the interior points of S and those convex hull points of degree 3 (that is, points adjacent to 3 other points in the boundary of $Conv(S)$). We can do this in view of the following:

Lemma 1 *If the convex hull points of S admit a Hamiltonian tetrahedralization, so does S.*

Proof. Consider an interior point x of S and suppose $S - \{x\}$ admits a Hamiltonian tetrahedralization $\mathcal{T}$. Let τ be the unique tetrahedron of $\mathcal{T}$ that contains x in its interior. If we remove τ from $\mathcal{T}$ and add the four tetrahedra induced by the faces of τ with x, we obtain a tetrahedralization of S and the Hamiltonian cycle of $D_{\mathcal{T}}$ can be extended to a Hamiltonian cycle of the new tetrahedralization. Applying this process recursively, the theorem follows. □

In the same manner we can suppose that S does not have any convex hull vertices of degree 3.

Theorem 2 *Let x be a convex hull point of S of degree 3. If $S - \{x\}$ admits a Hamiltonian tetrahedralization, then so does S.*

Proof. Suppose $S - \{x\}$ admits a Hamiltonian tetrahedralization $\mathcal{T}$. The three convex hull vertices of S adjacent to x form a face F of the boundary of $Conv(S - \{x\})$. Let τ_1 be the only tetrahedron of $\mathcal{T}$ that contains F as a face and let τ_2 be the tetrahedron induced by x and F. Clearly $\tau_1 \cup \tau_2$ is convex. If we remove τ_1 and τ_2 from $\mathcal{T}$ and replace them with the

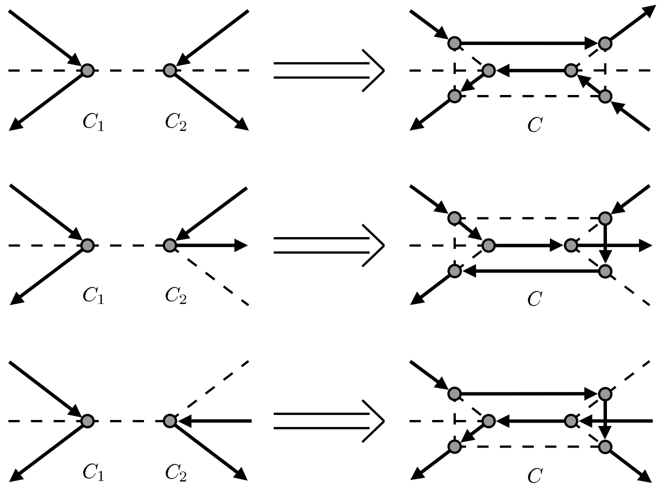


Figure 2: $D_{\mathcal{T}}$ before and after the join operation.

three tetrahedra induced by the faces of τ_1 (except F) and x, we obtain a tetrahedralization $\mathcal{T}'$ of S. The Hamiltonian cycle of $D_{\mathcal{T}}$ can now be extended to a Hamiltonian cycle of $D_{\mathcal{T}'}$

□

Assume now that S does not contain interior points or convex hull points of degree 3.

We insert a point p_0 in the interior of $Conv(S)$ and join every face of the boundary of $Conv(S)$ to it, forming a tetrahedralization $\mathcal{T}$ of $S \cup \{p_0\}$.

Let G be the graph induced by the 1-skeleton of the boundary of $Conv(S)$; that is, the graph whose vertex set consists of the convex hull points of S and whose edges are the edges of the boundary of $Conv(S)$. It is easy to see that both G and its dual graph are planar and 3-connected. By construction, the dual graph of G is isomorphic to $D_{\mathcal{T}}$. Since every face of G is a triangle, $D_{\mathcal{T}}$ is a regular graph of degree 3.

To obtain the initial partition, we use a theorem of Petersen [8] that states that every 2-connected cubic graph contains a perfect matching. Since $D_{\mathcal{T}}$ is 3-connected, in particular it is 2-connected and therefore contains a perfect matching M. If we remove the edges of M from $D_{\mathcal{T}}$, we obtain a regular graph of degree 2. This subgraph of $D_{\mathcal{T}}$ is the initial cycle partition.

2.1 Joining cycles

Consider two disjoint cycles, C_1 and C_2, in our cycle partition of $D_{\mathcal{T}}$, and furthermore suppose that there is an edge e of $D_{\mathcal{T}}$ that has its end points τ_1 and τ_2 in C_1 and C_2 respectively. Since τ_1 and τ_2 are tetrahedra in $\mathcal{T}$, e corresponds to a shared face F of τ_1 and τ_2.

The join operation consists of adding a point p to the interior of τ_1 so that the line segment joining the point q in τ_2 opposite to F in τ_2 intersects F. We now remove τ_1 and τ_2 and replace them by the six tetrahedra induced by the faces of τ_1, τ_2 and p (except F) as shown in Figure 1.

It can now be shown that there is a cycle that passes through all the vertices of $C_1 \cup C_2 - \{\tau_1, \tau_2\}$ plus the six new tetrahedra containing p as a vertex (see Figure 2).

We repeat this process until a single cycle is obtained. We will show in the next section that the number of Steiner points we need to insert before a Hamiltonian cycle is reached is at most $\lfloor \frac{n-2}{2} \rfloor$.

3 Complexity and implementation.

In this section we will analyze the running time and implementation issues of the algorithm sketched in Section 2.

Suppose that S is a point set with n points in $\mathbb{R}^3$ with m convex hull points and m' interior points, $m + m' = n$. We first calculate the convex hull of S in $O(n \log n)$, and then remove the points of S in the interior of $Conv(S)$.

Next, we remove the convex hull vertices of degree 3. This can be done in $O(m)$ by using a priority queue with all convex hull vertices of degree 3. Each time one is removed, the degree of its neighbors is checked and if necessary they are added to the queue.

Adding the first Steiner point p_0 and tetrahedralizing as in the previous section takes time $O(m)$.

The complexity of finding the initial cycle partition described at the end of Section 2 is that of finding a perfect matching in G. In a graph with $|V|$ vertices and $|E|$ edges, a perfect matching can be found in time $O(|E|\sqrt{|V|})$ [7]. Since we are dealing with a cubic graph, we have $|E| = \frac{3}{2}|V|$. Thus we can find the cycle cover in $0(\frac{3}{2}m\sqrt{m}) = O(m^{\frac{3}{2}})$ time.

Once we have the initial cycle cover, we return the vertices that were removed. This is done before the join operations in order to take advantage of the structure of the tetrahedralization to return the convex hull points of degree 3 and interior points efficiently. Using the fact that $D_{\mathcal{T}}$ is a planar graph, the interior points and convex hull points of degree 3 can be added using point location at a cost of $O(\log m)$ per point. The convex hull points of degree 3 are added first and the interior points afterwards. As these points are returned, the initial cycle partition is updated as in Lemma 1 and Theorem 2.

We have to be careful about the order in which the interior points are added. Suppose we have a tetrahedra τ which contains k interior points that remain to be added, and that we return q_0, one of these points. When we retetrahedralize the point set, τ would be split into 4 new tetrahedra. We have to guarantee that each of these tetrahedra receives a linear fraction of the points in τ, for otherwise the iterative process could take as much as $O(k^2)$. That is, we need a splitter vertex (see [2]). Such a vertex can be found in time $O(k)$, thus ensuring a total of $O(m' \log m')$ running time.

Finally we proceed to merge the set of cycles obtained thus far into a single cycle as in Subsection 2.1. Each time we join two cycles, we insert one Steiner point. Since G has m vertices, the number of faces of G is $2m - 4$, and since all the cycles obtained have at least four vertices, the initial cycle partition contains at most $\lfloor \frac{2m-4}{4} \rfloor$ elements. Thus the number of Steiner points required is at most $\lfloor \frac{m-2}{2} \rfloor$. This can be done in $O(n \log n)$ since there are $O(n)$ edges in H. The overall complexity of the algorithm is thus $O(m^{\frac{3}{2}}) + O(n \log n)$.

4 Hamiltonian Convex Hulls

To conclude the paper, we show that if the dual graph G defined by the convex hull of S is Hamiltonian, then no Steiner points need to be added.

Theorem 3 *Let S be a point set in $\mathbb{R}^3$ such that the dual graph H of G is Hamiltonian. Then S admits a Hamiltonian path tetrahedralization.*

Proof. Consider a planar embedding of H and a Hamiltonian cycle C of H. Let F be a face in this embedding such that all except one of its edges are in C.

Observe that there is a one-to-one mapping between the vertices of G and the faces of H. Let v be the vertex of G corresponding to F. Observe that each face of $Conv(S)$ (not containing v as one of its vertices) together with v induces a tetrahedron, and that the union of these tetrahedra forms a tetrahedralization of $Conv(S)$.

It is easy to see that the dual of this tetrahedralization is isomorphic to $H - F$, and thus contains a Hamiltonian path. □

Using Euler's formula and the fact that all 3-connected cubic planar graphs with 36 or fewer vertices have a Hamiltonian cycle (see [6]), we obtain the following corollary:

Corollary 4 *Let S be a point set in $\mathbb{R}^3$ having at most 20 convex hull points. Then S admits a Hamiltonian path tetrahedralization.*

The tetrahedralization mentioned in the proof of Theorem 3 (where all the points are joined to a given point) is known in the literature as a "pulling" tetrahedralization. Recently, point sets with no Hamiltonian path pulling tetrahedralizations have been shown to exist [4].

5 Conclusions

We presented an algorithm for computing Hamiltonian tetrahedralizations of a given point set S in $\mathbb{R}^3$ by adding Steiner points.

The algorithm has a running time of $O(m^{\frac{3}{2}}) + O(n \log n)$ and inserts at most $\lfloor \frac{m-2}{2} \rfloor$ Steiner points. We believe that this bound is not optimal.

We also showed that point sets with at most 20 convex hull points always admit a Hamiltonian path tetrahedralization.

We remark that we have restricted ourselves to adding Steiner points to the interior of $Conv(S)$. If we allow the use of Steiner points in the exterior of $Conv(S)$, four exterior points (the vertices of a tetrahedron containing the elements of S in its interior) suffice.

References

[1] E. Arkin, M. Held, J.S.B. Mitchell, and S. Skiena. *Hamiltonian triangulations for fast rendering.* Visual Comput., 12(9):429–444, 1996.

[2] D. Avis, H. ElGindy. *Triangulating point sets in space.* Discrete & Computational Geometry, 2:99–11, 1987.

[3] R. Fabila-Monroy, and J. Urrutia. *Graham Triangulations and Triangulations With a Center are Hamiltonian.* Information Processing Letters, 96, pp. 295-299, 2005.

[4] F. Chin, Q. Ding and C. Wang *On Hamiltonian tetrahedralizations of convex polyhedra* Proceedings of ISORA'05 (Lecture Notes on Operations Research), Vol. 5, pp. 206–216, August 2005, Lhasa, China.

[5] E.D. Demaine, J.S.B. Mitchell, and J. O'Rourke, editors. *The Open Problems Project.* http://maven.smith.edu/orourke/TOPP/ welcome.html

[6] D.A. Holton and B. D. Mckay. *The smallest non-Hamiltonian 3-connected cubic planar graphs have 38 vertices.* J. Combin. Theory, Ser. B. 45:315–319, 1988.

[7] S. Micali, V. V. Vazirani, *An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs.* 21st Annual Symp. on Foundations of Computer Science, Syracuse, NY, 1980, pp 17–27.

[8] J. Petersen. *Die Theorie de regulären Graphen* Acta Math. 15:193–220, 1891.

[9] H Whitney. *Congruent graphs and the connectivity of graphs* Amer. J. Math. 54:150–168, 1932.

A conjecture about Minkowski additions of convex polytopes [*]

Komei Fukuda[†] Christophe Weibel[‡]

Abstract

This paper introduces a conjecture about Minkowski sums of polytopes. When the facets of the sum have exact decomposition, a linear relation can be observed on the augmentation of the number of faces between the sum and the summands. We prove this relation holds for two particular cases.

1 Introduction

Let $P_1, \dots, P_n$ be polytopes in $\mathbb{R}^d$. Their Minkowski sum is defined as

$$P = \{x_1 + \cdots + x_n \mid x_i \in P_i \ \forall i\}.$$

Minkowski sums can be used in a wide variety of subjects, such as robotics ([8]), manufacturing ([9]) and algebraic statistics ([5]). One of the main subjects of interest, for efficiency computation as well as some theoric applications, is the study of the number of faces of the sum related to that of the summands.

A first bound has been presented by Gritzman and Sturmfels ([5]), bounding the number of faces of any dimension in terms of the number of edges of the summands. More recently, other bounds have been proposed in terms of facets or vertices, but mostly in low dimensions ([2, 4]).

We will now introduce a conjecture relating to the number of faces of different dimensions in a certain type of Minkowski sums. The term *face* here defines a set on which a linear function is optimal in the polytope. Vertices, edges and facets are faces of dimension 0, 1, and $d-1$ respectively. The *f-vector* of a polytope P, noted by $(f_0(P), \dots, f_{f-1}(P))$, contains the number of faces of different dimensions.

Any face F of a Minkowski sum of polytopes can be *decomposed* uniquely into a sum of faces of the summands ([3]). We will say that the decomposition is *exact* when the dimension of the sum is equal to the sum of the dimensions of the summands. When all facets have an exact decomposition, we will say the summands are *relatively in general position.*

Though this condition is rather restrictive, it can be shown that for fixed number of faces in the summands, the maximum number of faces of any dimension can only be attained if the summands are relatively in general position.

This is our main conjecture:

Conjecture 1 *Let $P_1, \dots, P_n$ be d-dimensional polytopes relatively in general position, and $P = P_1 + \cdots + P_n$ their Minkowski sum. Then*

$$\sum_{k=0}^{d-1} (-1)^k k (f_k(P) - (f_k(P_1) + \cdots + f_k(P_n))) = 0.$$

Note that the form is rather similar to Euler's Equation:

$$\sum_{k=0}^{d-1} (-1)^k f_k(P) = 1 - (-1)^d.$$

By using Euler's Equation, we can write the conjecture slightly differently:

Corollary 1 *Let $P_1, \dots, P_n$ be d-dimensional polytopes relatively in general position, and $P = P_1 + \cdots + P_n$ their Minkowski sum. Then for all a,*

$$\sum_{k=0}^{d-1} (-1)^k (k-a)(f_k(P) - (f_k(P_1) + \cdots + f_k(P_n)))$$

$$= a - a(-1)^d.$$

2 Cases proved

We will show here proofs for two different families of Minkowski Sums. General proofs for three-dimensional sums can be found in [2] and [4].

2.1 Proof for zonotopes sums

The f-vector of general zonotopes are completely known. Since the sum of two zonotopes is again a zonotope, it is possible to prove the following:

Theorem 2 *Let $Z_d^{m_1}$ and $Z_d^{m_2}$ be two d-dimensional zonotope in generated by m_1 respectively m_2 segments in general position, then the main conjecture holds for their Minkowski sum.*

[*]Supported by the Swiss National Science Foundation Project 200021-105202, "Polytopes, Matroids and Polynomial Systems".

[†]Mathematics Institute, EPFL, Lausanne, Switzerland. Also affiliated with Institute for Operations Research and Institute of Theoretical Computer Science ETH Zentrum, Zurich, Switzerland. `fukuda@ifor.math.ethz.ch`

[‡]Mathematics Institute, EPFL, Lausanne, Switzerland. `christophe.weibel@epfl.ch`

Proof. *As stated in [6] the f-vector of Z_d^m is given by:*

$$f_k(Z_d^m) = 2\binom{m}{k}\sum_{h=0}^{d-k-1}\binom{m-k-1}{h}.$$

Defining $d' = d-1$, $k' = k-1$ and $m' = m-1$, and using the identity $k\binom{m}{k} = m\binom{m'}{k'}$, and we can write:

$$\sum_{k=0}^{d-1}(-1)^k k f_k(Z_d^m) =$$

$$\sum_{k=1}^{d-1}(-1)^k k 2\binom{m}{k}\sum_{h=0}^{d-k-1}\binom{m-k-1}{h} =$$

$$-\sum_{k'=0}^{d'-1}(-1)^{k'} m 2\binom{m'}{k'}\sum_{h=0}^{d'-k'-1}\binom{m'-k'-1}{h} =$$

$$-m\sum_{k'=0}^{d'-1}(-1)^{k'} f_{k'}(Z_{d'}^{m'}).$$

Since $Z_{d'}^{m'}$ is a polytope, Euler's formula tells us that the alternating sum of its f-vector is equal to $1-(-1)^{d'}$, and so

$$\sum_{k=0}^{d-1}(-1)^k k f_k(Z_d^m) = -m(1+(-1)^d).$$

The conjecture is therefore equivalent to the obvious statement:

$$-(m_1+\cdots+m_n)(1+(-1)^d)+$$

$$m_1(1+(-1)^d)+\cdots+m_n(1+(-1)^d) = 0.$$

□

2.2 Proof for Nesterov roundings of perfectly centered polytopes

We will here present the proof for the particular case of perfectly centered polytopes summed with their own dual. We will use *extended f-vectors*, which give for a set of dimensions $S = \{s_1,\ldots,s_k\} \subseteq \{0,\ldots,d-1\}$ the number of different chains of faces $F_1 \subseteq \cdots \subseteq F_k$ so that $dim(F_i) = s_i$.

We will need to use the Dehn-Sommerville relations for extended f-vectors:

Lemma 3 ([1],[7]) *Let P be an Eulerian poset of rank d, $S \subset \{0,\ldots,d-1\}$, $\{i,k\} \subseteq S \cup \{-1,d\}$, $i < k-1$, and S contains no j so that $i < j < k$. Then*

$$\sum_{j=i+1}^{k-1}(-1)^{j-i-1} f_{S\cup j}(P) = f_S(P)(1-(-1)^{k-i-1}).$$

If we use for convenience the notation $f_{i,i,k} := f_{i,k}$, we can write the special case of D-S relations where $S = \{i,k\} \subseteq \{-1,\ldots,d\}$ as:

Lemma 4

$$\sum_{j=i}^{k}(-1)^j f_{i,j,k}(P) = 0.$$

A polytope is called *perfectly centered* if for each non-empty face F, the intersection of F with its own normal fan $\mathcal{N}(F)$ is non-empty.

Theorem 5 ([4]) *Let P be a perfectly centered polytope. A subset H of $P+P^*$ is a nontrivial face of $P+P^*$ if and only if $H = G+F^D$ for some ordered nontrivial faces $G \subseteq F$ of P.*

Using this theorem, we can write a formula for the f-vector of $P+P^*$ using the extended f-vector of P.

Theorem 6 *Let P be a perfectly centered polytope, then the f-vector of $P+P^*$ can be written as:*

$$f_k(P+P^*) = \sum_{i=0}^{k} f_{i,i+d-1-k}, \quad \forall k = 0,\ldots,d-1$$

Proof. *Let P be a perfectly centered polytope. For every k, the sumber of k-faces of $P+P^*$ is equal to the number of pairs of faces (F,G) of P, $F \subseteq G$ so that $dim(F)+dim(G^D) = k$, which means $dim(F)+d-1-k = dim(G)$. Which is the number of chains of two non-trivial faces of dimensions i and $i+d-1-k$.* □

Theorem 7 *Let P be a perfectly centered polytope. Then the conjecture holds for the Minkowski sum $P+P^*$.*

Proof. *Let P be a perfectly centered polytope. We have that*

$$\sum_{k=0}^{d-1}(-1)^k k f_k(P) = \sum_{i=0}^{d-1}(-1)^i i f_{i,d}(P)$$

By using $k' = d-1-k$:

$$\sum_{k=0}^{d-1}(-1)^k k f_k(P^*) =$$

$$\sum_{k'=0}^{d-1}(-1)^{d-1-k'}(d-1-k')f_{-1,k'}(P)$$

And finally:

$$\sum_{k=0}^{d-1}(-1)^k k f_k(P+P^*) = \sum_{k=0}^{d-1}\sum_{i=0}^{k}(-1)^k k f_{i,i+d-1-k}(P)$$

Let's replace in this sum k by $k' = i + d - 1 - k$:

$$= \sum_{i=0}^{d-1} \sum_{k'=i}^{d-1} (-1)^{i+d-1-k'} (i + d - 1 - k') f_{i,k'}(P)$$

$$= \sum_{i=0}^{d-1} \sum_{k'=i}^{d-1} (-1)^{i+d-1-k'} i f_{i,k}(P) +$$

$$\sum_{k'=0}^{d-1} \sum_{i=0}^{k'} (-1)^{i+d-1-k'} (d - 1 - k') f_{i,k}(P)$$

Composing the three, we get:

$$\sum_{k=0}^{d-1} (-1)^k k f_k(P + P^*) - f_k(P) - f_k(P^*) =$$

$$\sum_{i=0}^{d-1} \sum_{k=i}^{d} (-1)^{i+d-1-k} i f_{i,k}(P) +$$

$$\sum_{k=0}^{d-1} \sum_{i=-1}^{k} (-1)^{i+d-1-k} (d - 1 - k) f_{i,k}(P) =$$

$$\sum_{i=0}^{d-1} i \sum_{k=i}^{d} (-1)^{i+d-1-k} f_{i,k,d}(P) +$$

$$\sum_{k=0}^{d-1} (d - 1 - k) \sum_{i=-1}^{k} (-1)^{i+d-1-k} f_{-1,i,k}(P) = 0$$

By Dehn-Sommerville relations (4), the inner sums are equal to zero. □

References

[1] M. M. Bayer and L. J. Billera. Generalized Dehn-Sommerville relations for polytopes, spheres and Eulerian partially ordered sets. *Invent. Math.*, 79(1):143–157, 1985.

[2] E. Fogel, D. Halperin, and C. Weibel. On the exact maximum complexity of Minkowksi sums of convex polyhedra. In *23rd Annual ACM Symposium on Computational Geometry*, 2007.

[3] K. Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. *J. Symbolic Comput.*, 38(4):1261–1272, 2004.

[4] K. Fukuda and C. Weibel. On f-vectors of Minkowski additions of convex polytopes. *Discrete and Computational Geometry*, 2006. ACCEPTED.

[5] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: computational complexity and applications to Gröbner bases. *SIAM J. Discrete Math.*, 6(2):246–269, 1993.

[6] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: computational complexity and applications to Gröbner bases. *SIAM J. Discrete Math.*, 6(2):246–269, 1993.

[7] B. Lindström. On the realization of convex polytopes, Euler's formula and Möbius functions. *Aequationes Math.*, 6:235–240, 1971.

[8] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[9] J.-P. Petit. *Spécification géométrique des produits : Méthode de détermination des tolérances. Application en conception assistée par ordinateur.* PhD thesis, Université de Savoie, 2004.

Between Umbra and Penumbra

Julien Demouth [†] Olivier Devillers [‡] Hazel Everett [†] Sylvain Lazard [†] Raimund Seidel [§]

Abstract

Computing shadow boundaries is a difficult problem in the case of non-point light sources. A point is in the umbra if it does not see any part of any light source; it is in full light if it sees entirely all the light sources; otherwise, it is in the penumbra. While the common boundary of the penumbra and the full light is well-understood, less is known about the boundary of the umbra. In this paper we present various bounds on the complexity of the umbra cast by a segment or polygonal light source on a plane in the presence of polygon or polytope obstacles.

1 Introduction

Shadows play a central role in human perception. Unfortunately, computing realistic shadows efficiently is a difficult problem, particularly in the case of non-point light sources. A wide spectrum of approaches has been considered for rendering shadows (see, for example, the surveys [8], [4]); many methods make extensive use of graphics hardware (see the survey [6]).

We say that a point is in the *umbra* if it does not see any part of the light source(s); it is in *full light* if it sees entirely all the light source(s); otherwise, it is in the *penumbra*. While the boundary of the penumbra is well-understood, less is known about the boundary of the umbra. Nevertheless, there is an extensive literature concerning the explicit computation of these boundaries; see, for example, [3, 5, 7]

In this paper we present various bounds, summarized in Table 1, on the complexity (*i.e.*, number of vertices and arcs) of the umbra cast by a segment or polygonal light source on a plane in the presence of polytopal obstacles. We show that the complexity of the umbra cast by k polytopes of total complexity n has an $O(nk^3)$ upper bound in the presence of a segment light source and an $O(n^3k^3)$ upper bound in the presence of a polygonal light source. Even though these bounds are not tight, they improve drastically over the only previously known bounds which were the trivial $O(n^4)$ and $O(n^6)$ upper bounds.

Scene type	Lower bounds	Upper bounds
Segment light source		
2 triangles	4	$O(1)$
2 fat polytopes	$\Omega(n)$	$O(n)$
k polytopes	$\Omega(nk^2 + k^4)$	$O(nk^3)$
n-gon light source		
k polytopes	$\Omega(nk^3 + k^6)$	$O(n^3k^3)$

Table 1: Lower bounds on the number of connected components and upper bounds on the complexity of the umbra. All polytopes have complexity $O(n)$.

We exhibit a configuration where a single segment light source may cast, in the presence of two triangles, four connected components of umbra. We also prove an $\Omega(nk^2)$ lower bound on the maximum number of connected components of the umbra in the presence of a segment light source and k disjoint polytopes of total complexity n. This lower-bound example is rather pathological in the sense that most of the obstacles are long and thin. However, we also present lower bound examples of $\Omega(n)$ with two fat n-gons obstacles and $\Omega(k^4)$ with k convex obstacles. Finally, in the presence of a polygonal light source, we exhibit an $\Omega(nk^3 + k^6)$ lower bound.

It is interesting that, even for the simplest cases of non-point light sources, obtaining tight bounds on the complexity of the umbra and understanding its structure is a very challenging problem.

2 Preliminaries

Our setting is $\mathbb{R}^3$. Let s be a line segment and p a point. We denote by $\langle s, p\rangle$ the set of line transversals of s and p. Similarly, for any triple of segments s_1, s_2 and s_3, we denote by $\langle s_1, s_2, s_3\rangle$ its set of line transversals. It is a well-known fact that $\langle s_1, s_2, s_3\rangle$ consists of lines belonging to the same regulus of a ruled quadric surface. Hence any set of transversals, whether $\langle s, p\rangle$ or $\langle s_1, s_2, s_3\rangle$, forms patches of a quadric (possibly degenerating to one or two planes). Moreover, the set of transversals consists of at most three patches, or more formally, at most three connected components in line space [2]. We will be loose in our use of notation and we let $\langle s, p\rangle$ and $\langle s_1, s_2, s_3\rangle$ not just denote sets of lines but also the surface patches they form.

Let $\mathcal{P}$ be a finite set of disjoint convex polygons or

[†]INRIA-Lorraine – University Nancy 2, LORIA, Nancy, France, Firstname.Lastname@loria.fr
[‡]INRIA, Sophia-Antipolis, France, Olivier.Devillers@inria.fr
[§]Saarland University, FR Informatik, Saarbrücken, Germany, rseidel@cs.uni-sb.de

polytopes in $\mathbb{R}^3$ with $\mathcal{L} \subset \mathcal{P}$ identified as light sources. A surface $\sigma = \langle e, v \rangle$ is called a EV-*surface* if there exist two distinct objects $P, Q \in \mathcal{P}$ so that e is an edge of P, v a vertex of Q and σ intersects a light source. A surface $\sigma = \langle e_1, e_2, e_3 \rangle$ is called a EEE-*surface* if there exist three distinct objects $P, Q, R \in \mathcal{P}$ so that e_1, e_2 and e_3 are respectively edges of P, Q and R and σ intersects a light source.

Any plane Π intersects an EV-surface or an EEE-surface in a set of arcs of a conic (each possibly empty or possibly a line segment). Hence the intersection between Π and *all* the EV and EEE surfaces defines an arrangement of arcs of conics on Π.

Here we are interested in the arcs of conics that correspond to shadow boundaries. In particular, we are interested in maximal free line segments[1] that intersect a light source and are supported by a line that is on an EV or EEE surface. The intersection of these free line segments with Π defines an arrangement of arcs of conics on Π which we call the *shadow arrangement* on the *shadow plane* Π.

A point p is in the umbra if for any point q on a light source, the segment pq intersects an object from $\mathcal{P} \setminus \mathcal{L}$. Similarly, p is in full light if for any point q on a light source, the segment pq does not intersect any object from $\mathcal{P} \setminus \mathcal{L}$. Otherwise, p is in the penumbra.

We will make extensive use of the fact that the boundary of the umbra and penumbra consists of arcs of the shadow arrangement; see, for example [7].

Throughout this paper, we consider the regions of umbra and penumbra on a plane cast by a single light source in the presence of polytopes.

3 Penumbra boundary

In this section we refer to the union of the umbra and penumbra as the *shadow region*. We give lower and upper bounds on the complexity of the shadow boundary. We omit the proofs of these results because of lack of space. Note that the boundary of the shadow region is only composed of line segments induced by EV-surfaces. The absence of boundary induced by quadric EEE-surfaces simplifies the computation of these regions.

Theorem 1 *The complexity of the shadow region cast on a plane Π by a convex polygonal light source of complexity m in the presence of k convex polyhedra of total complexity n is, in the worst case, in $\Omega(n\,\alpha(k) + k\,m + k^2)$ and $O(n\,\alpha(k) + k\,m\,\alpha(k) + k^2)$, where $\alpha(k)$ denotes the pseudo-inverse of the Ackermann function.*

[1]A *maximal free line segment* is a segment that intersects the interior of no polytope and whose endpoints lie on some polytope or at infinity.

4 Upper bounds on the complexity of the umbra

In this section we present the following two upper bounds. The proof of Theorem 2 is omitted due to lack of space.

Theorem 2 *The complexity of the umbra cast by one segment light source and k disjoint polytopes of total complexity n is $O(nk^3)$.*

Theorem 3 *The complexity of the umbra cast by one polygonal light source and k disjoint polytopes of total complexity n is $O(n^3k^3)$.*

To prove Theorem 3, we consider an arrangement A of arcs of conics that contains the shadow arrangement. This arrangement A consists of the intersections of Π with (i) the lines that are transversal to a vertex and an edge of two polytopes and that do not intersect the interior of either of these polytopes (the connected components of these lines form patches of EV-surfaces) (ii) the lines that are transversal to the edges of three polytopes and that do not intersect the interior of these polytopes (the connected components of these lines form patches of EEE-surfaces). We will establish a $O(n^3k^3)$ upper bound on the complexity of A which yields the same bound for the complexity of the umbra.

Lemma 4 *Any line L in Π intersects at most $O(nk^2)$ arcs of conics of A.*

Proof. An intersection point between L and A corresponds to a line Δ which belongs to an EV or EEE surface. Consider first EV-surfaces. The line Δ lies in a plane which contain L and a vertex, say v, of one of the polytopes. There exist $O(n)$ such planes and in each of them there are at most $O(k)$ lines through v that are tangent to a polytope. Thus there are at most $O(nk)$ points on $L \cap A$ which correspond to lines in EV-surfaces.

Now we consider EEE-surfaces. Let n_i be the number of vertices of polytope P_i, for $1 \leq i \leq k$. The number of EEE-surfaces generated by three edges of polytopes P_i, P_j and P_l, not intersecting the interior of P_i, P_j and P_l, and that intersect L is $O(n_i+n_j+n_l)$ [1, Main Lemma]. Since $\sum_{1 \leq i<j<l \leq k} O(n_i+n_j+n_l) = O(nk^2)$, there are at most $O(nk+nk^2) = O(nk^2)$ arcs of A which intersect the line L on Π. $\square$

Proof of Theorem 3. Here, we introduce an arbitrary coordinate frame Oxy in the plane Π. We call Ox the horizontal axis and Oy the vertical axis.

We first show that the number of intersection points on A is $O(nk^2)$ times the number of conic arcs. We first break all conic arcs into maximal “horizontally monotone” pieces. This increases the number of arcs only by a constant factor. Consider a conic arc α_0

and its rightmost endpoint p along Ox. We charge to p all points of intersection between α_0 and another conic arc whose rightmost endpoint is to the right of p. Any arc intersects α_0 in at most $O(1)$ points so the number of such intersection points is bounded by the number of arcs intersected by a vertical line in Π and containing p. By Lemma 4, there are at most $O(nk^2)$ such arcs. Thus, each endpoint is charged at most $O(nk^2)$ times.

We now bound the number of arcs (and thus the number of arc endpoints) generating A. Let n_i be the number of vertices of polytope P_i, $1 \leq i \leq k$ and e an edge. The number of EEE-surfaces pertinent to A and involving e and edges from polytopes P_i and P_j is $O(n_i + n_j)$ [1, Corollary 9]. Thus, for each edge e, there are, at most, $\sum_{1 \leq i < j \leq k} O(n_i + n_j) = O(nk)$ EEE-surfaces having e as a generating segment. Since there exist n edges, the total number of arcs is therefore $O(n^2k)$.

In conclusion, there are at most $O(n^2k)$ arcs generating A, each of them charged with at most $O(nk^2)$ intersection points, hence there are at most $O(n^3k^3)$ intersection points. The total complexity of the A and, thus of the umbra, is $O(n^3k^3)$. □

Note that Theorem 2 can be proved similarly by noticing that, in the case of a single segment light source, the EEE-surfaces $\sigma = \langle e_1, e_2, e_3 \rangle$ and EV-surfaces $\sigma = \langle e, v \rangle$ that contribute to the shadow arrangement are such that e_1, e_2 or e_3 is the segment light source and such that e is the segment light source or v is one of its endpoints.

5 Lower bounds

We present here several lower bounds on the maximum number of connected components of the umbra.

Theorem 5 *A segment light source and two triangles may cast, on a plane, four connected components of umbra.*

We prove Theorem 5 by providing an example of two triangles and a segment light source that admit four connected components of umbra on a particular plane (see Figure 1). Because of lack of space we omit the proof here. We also omit the proof of the following theorem which is similar to Theorem 7.

Theorem 6 *The umbra cast by one segment light source in the presence of two fat convex polygons of total complexity n can have $\Omega(n)$ connected components.*

Theorem 7 *The umbra cast by one segment light source in the presence of k polytopes of total complexity n can have $\Omega(nk^2)$ connected components.*

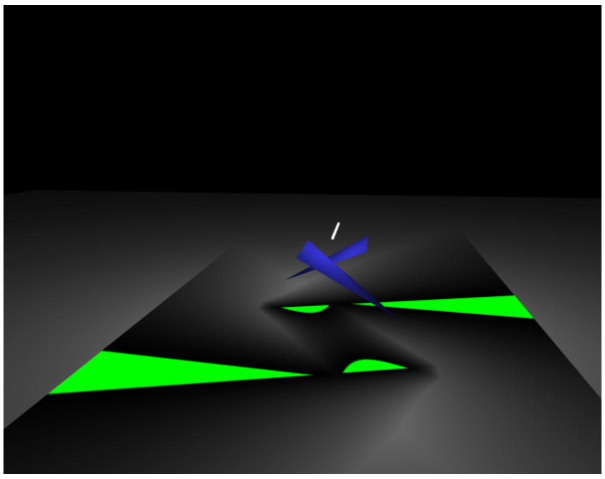


Figure 1: A segment light source, two triangles, and their four induced connected components of umbra. Rendered with OpenRT; courtesy of Andreas Dietrich.

Proof. Consider three non-parallel segments s_1, l_2, and l_3 all parallel to the shadow plane Π and planes $P_2 \supset l_2$ and $P_3 \supset l_3$ parallel to Π, refer to Figure 2. The surface $\langle s_1, l_2, l_3 \rangle$ intersects Π in a conic arc α.

Now consider the following setup: s_1 is the light source; P_2 has k narrow rectangular holes (or slits) parallel and arbitrary close to l_2; similarly P_3 has k slits parallel and arbitrary close to l_3. (A plane with such k slits can be modelled by $O(k)$ rectangles.) Each pair of slits, s_2 from P_2 and s_3 from P_3, together with the light source s_1 induce a piece of penumbra in Π that is essentially a thickened copy of the conic arc α.

We thus get that the umbra covers the whole plane Π except for k^2 curves of penumbra that are all close to α (see Figure 2-left).

Finally, we trim the two planes P_2 and P_3, creating an n-sided convex polygon on Π such that the region outside is in light or penumbra and each edge intersects all the k^2 curves. The umbra then consists of nk^2 regions inside the convex polygon and between the k^2 conics (see Figure 2-right). Note that the $O(k)$ convex obstacles can each be transformed into a polytope by the addition of a single vertex without changing the umbra. □

Theorem 8 *The umbra cast by a segment light source in the presence of k polytopes can have $\Omega(k^4)$ connected components.*

Proof. Refer to Figure 3. We create k^2 curves of penumbra using parallel thin holes. Making a second set of thin holes in each plane, we create a second family of curves of light and penumbra intersecting the first one. The umbra is now the complement of

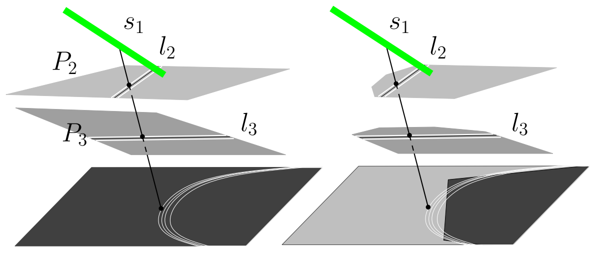


Figure 2: $\Omega(nk^2)$ lower bound.

the union of these two sets of curves and it consists of k^4 connected components. □

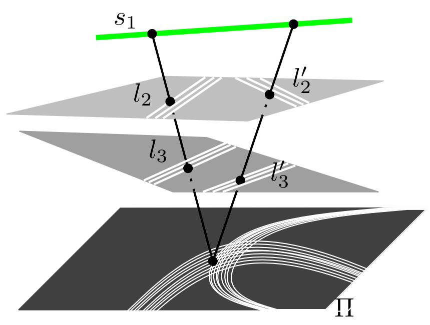


Figure 3: $\Omega(k^4)$ lower bound.

We finally present a lower bound on the complexity of the umbra cast by a polygonal light source in the presence of k polygonal obstacles.

Theorem 9 *The umbra defined by one polygonal light source and k convex obstacles can have $\Omega(nk^3 + k^6)$ connected components.*

Proof. The $\Omega(k^6)$ lower-bound example is built similarly as in the proof of Theorem 8. The $\Omega(nk^3)$ one is created in the same way as in Theorem 7. Due to the lack of space, we omit the detailed description of these examples. □

6 Conclusion

The purpose of this paper is to establish the complexity of the boundaries between the umbra, penumbra and fully lit regions on a plane in a scene consisting of k polytopes of total complexity n.

The results presented here constitute a first step toward understanding the intrinsic structure and complexity of the umbra in this setting. We have proved that if the light is reduced to one line segment, then the umbra may have $\Omega(nk^2 + k^4)$ connected components and $O(nk^3)$ complexity. We have also shown that a polygonal light source could generate an umbra with $\Omega(nk^3 + k^6)$ connected components and $O(n^3k^3)$ complexity. In both cases these components of umbra are delimited by arcs of conics. These results prove that the umbra is intrinsically much more intricate than the penumbra boundary which only contains line segments and has complexity $O(n\,\alpha(k) + k\,m\,\alpha(k) + k^2)$, where m is the complexity of the light source.

Our upper bounds, in fact, apply to the complexity of the arrangement of the curves where the derivative of the light intensity is discontinuous. These arrangements clearly include the limits between penumbra and umbra and those between penumbra and full light. It is thus overkill to use this arrangement for computing an upper bound on the complexity of the umbra. Although a gap remains between our lower and the upper bounds, we still have some tight bounds in the case of a segment light source: on the one hand, the bounds are tight for small k ($k = O(1)$) and for small n ($n = O(k)$); on the other hand, we prove that the upper bound on the arrangement generated by a segment light source is tight, that is, there exists a scene and a shadow plane where the arrangement of derivative discontinuity curves has $\Theta(nk^3)$ complexity.

References

[1] H. Brönnimann, O. Devillers, V. Dujmović, H. Everett, M. Glisse, X. Goaoc, S. Lazard, H.-S. Na and S. Whitesides. On the number of maximal free line segments tangent to arbitrary three-dimensional convex polyhedra. Research Report n° 5671, INRIA, Sept 2005. To appear in *SIAM Journal on Computing*.

[2] H. Brönnimann, H. Everett, S. Lazard, F. Sottile and S. Whitesides. Transversals to line segments in three-dimensional space. *Discrete and Computational Geometry*, 34(3):381–390, 2005.

[3] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics* Proceedings, ACM SIGGRAPH, pages 223-230, 1994.

[4] F. Durand. A multidisciplinary survey of visibility. ACM SIGGRAPH course notes, Visibility, Problems, Techniques, and Applications, 2000.

[5] F. Durand, G. Dretakkis and C. Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics*, 18(2):128–170, 1999.

[6] J-M. Hasenfratz, M. Lapierre, N. Holzschuch and F. Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*, 2003.

[7] P. S. Heckbert. Discontinuity meshing for radiosity. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 203–215, 1992.

[8] A. Woo, P. Poulin and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, 1990.

Good Θ-illumination of Points

Manuel Abellanas*,† Antonio Bajuelos‡,§ Inês Matos‡,¶

Abstract

A point p in the plane is well Θ-illuminated by a set F of n light sources if there is, at least, one light source interior to each cone emanating from p with a given angle Θ. We consider the illumination range of the light sources as a parameter to be optimized and the angle Θ as a fixed value, this is, $\frac{1}{\Theta}$ is constant. Given an angle $\Theta \leq \pi$, we present an algorithm to minimize the light sources' illumination range to well Θ-illuminate a given point p. The algorithm runs in linear time which is optimal. It also computes a minimal embracing set of light sources that well Θ-illuminates p with minimum illumination range. Good Θ-illumination is related to the concept of t-good illumination [1] as it is shown in Proposition 3.

1 Introduction

Visibility or illumination has been a main topic for different papers in the area of Computational Geometry (for more information on the subject, check T. Asano et. al [3] and J. Urrutia [7]). We present an illumination problem which is a generalization of the 1-good illumination of minimum range [1, 2]. In its original definition [1], a point is 1-well illuminated if there is, at least, one light source illuminating p in every open half plane passing through p. We extend this concept to cones and make a brief relation between our problem and the Maxima Problem [4, 6].

Let F be a set of n light sources in the plane. Each light source $f_i \in F$ has limited illumination range $r > 0$, this is, they can only illuminate objects that are within the circle centered at f_i with radius r. A cone emanating from a point p is the region between two rays starting at p.

*Facultad de Informática, Universidad Politécnica de Madrid, mabellanas@fi.upm.es

†Supported by grant TIC2003-08933-C02-01, MEL-HP2005-0137 and partially supported by CAM:P-DPI-000235-0505

‡Departamento de Matemática & CEOC, Universidade de Aveiro, {leslie,ipmatos}@mat.ua.pt

§Supported by CEOC through *Programa* POCTI, FCT, co-financed by EC fund FEDER and by Acção No. E-77/06

¶Supported by a FCT fellowship, grant SFRH/BD/28652/2006, by CEOC through *Programa* POCTI, FCT, co-financed by EC fund FEDER and by Acção No. E-77/06

Definition 1 *Let F be a set of n light sources and $\Theta \leq \pi$ a given angle. We say that a point p in the plane is well Θ-illuminated by F if there is, at least, one light source interior to each cone emanating from p with an angle Θ.*

There is an example of this definition in Figure 1(a). A light source $f_i \in F$ is an embracing site for point p if p is well Θ-illuminated by the set formed by f_i and all the light sources whose distance to p is less or equal than the distance between p and f_i. As there may be more than one embracing site per point, our main goal is to compute the closest embracing site for a given point p. The distance between the closest embracing site and p is called the Minimum Embracing Range (MER) of p and it is denoted by r_m (see Figure 1(b)). A set formed by the closest embracing site f_i and all the light sources closer to p than f_i is called a minimal embracing set for p. Computing the MER of a given point p is important to us because once we have it, the minimum illumination range that the light sources of the minimal embracing set need to well Θ-illuminate p is its MER (see Figure 1(c)).

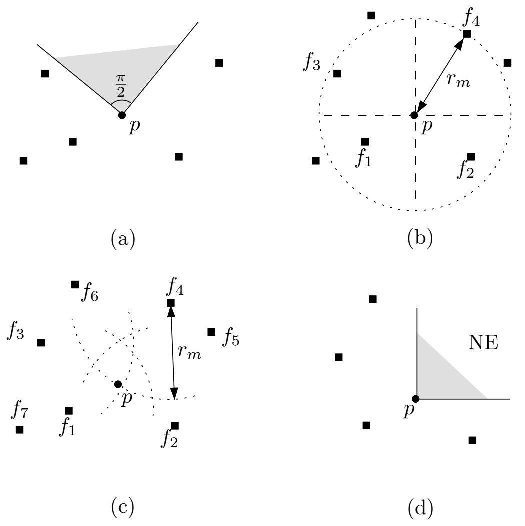


Figure 1: (a) Point p is not well $\frac{\pi}{2}$-illuminated because there is, at least, one empty cone starting at p with an angle $\frac{\pi}{2}$. (b) Light source f_4 is the closest embracing site for p, so the MER of p is $r_m = d(f_4, p)$. (c) Point p is well π-illuminated, its minimal embracing set is $\{f_1, f_2, f_3, f_4\}$ and the illumination range of these four light sources is r_m. (d) Point p is a maximum.

These well Θ-illuminated points are clearly related to dominance and maximal points. Let $p, q \in S$ be two points in the plane. We say that $p = (p_x, p_y)$ dominates $q = (q_x, q_y)$, $q \prec p$, if $p_x > q_x$ and $p_y > q_y$. Therefore, a point is said to be maximal (or maximum) if it is not dominated or in other words, it means that the quadrant NE centered at p must be empty (see Figure 1(d)). This version of maximal points can be extended. According to the definition of Avis et. al [4], a point p in the plane is said to be an unoriented Θ-maximum if there is an empty cone centered at p with an angle of, at least, Θ. The problem of finding all the maximal points of a set S is known as the *maxima problem* [6] and the problem of finding all the unoriented Θ-maximal points is known as the *unoriented Θ-maxima problem* [4]. The next proposition follows from the definitions of good Θ-illumination and unoriented Θ-maxima.

Proposition 1 *Given a point p in the plane and an angle Θ, a point p is well Θ-illuminated by F if and only if it is not an unoriented Θ-maximum of the set $F \cup \{p\}$.*

The next section presents an algorithm to compute the Minimum Embracing Range (MER) and the minimal embracing set for a well Θ-illuminated point, given $\Theta \leq \pi$. It also establishes the bridge between 1-good illumination and good Θ-illumination.

2 Minimum Embracing Range of a Well Θ-Illuminated Point

We now present a linear time algorithm that not only decides if a point is well Θ-illuminated as it also computes the closest embracing site and the minimal embracing set for a given point p in the plane. The main idea is to divide the plane in several cones and test the distribution of the light sources while making a logarithmic search for the MER. We split the algorithm's explanation in two parts: first we show how to decide if a set of light sources well Θ-illuminates p and then we show how to compute the MER of p.

Let F be a set of n light sources, p a point in the plane and $\Theta \leq \pi$ a given fixed angle. To check if p is well Θ-illuminated, we divide the plane in several cones emanating from p with an angle of $\frac{\Theta}{2}$. Let n_c be the number of possible cones, if 2π is divisible by Θ then $n_c = \frac{4\pi}{\Theta}$ (see Figure 2(a)). Otherwise $n_c = \lceil \frac{4\pi}{\Theta} \rceil$ because the last cone has an angle less than $\frac{\Theta}{2}$ (see Figure 2(b)). Since the angle Θ is considered to be a fixed value, the number of cones is constant. Let i be an integer index of arithmetic mod n_c. For $i = 0, \ldots, n_c$, each ray i is defined by the set $\{p + (\cos(\frac{i\Theta}{2}), \sin(\frac{i\Theta}{2}))\lambda : \lambda > 0\}$, while each cone is defined by p and two consecutive rays.

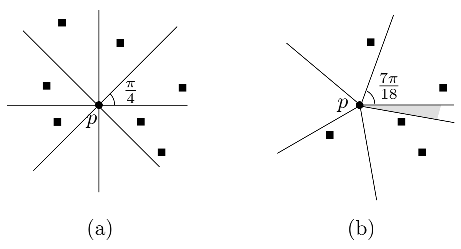


Figure 2: (a) To check if p is well $\frac{\pi}{2}$-illuminated, the plane is divided in eight cones with an angle of $\frac{\pi}{4}$. (b) To check if p is well $\frac{7\pi}{9}$-illuminated, the plane is divided in six cones and the last one has an angle less than $\frac{7}{18}\pi$ because 2π is not divisible by $\frac{7}{18}\pi$.

Since we have cones with an angle of at least $\frac{\Theta}{2}$, it is straightforward to see that p is not well Θ-illuminated if we have two consecutive empty cones (see Figure 3(a)). On the other hand, if all cones have at least one interior light source then p is well Θ-illuminated (see Figure 3(b)). In the last case, there can be at least one empty cone but no two consecutive empty ones (see Figure 4(a)). We need to spread each empty cone, this is, we have to open out the rays that define it until we find one light source on each side. Let f_l be the first light source we find on the left and f_r the first light source we find on the right (see Figure 4(b)). If the angle formed by f_l, p and f_r is at least equal to Θ than there is an empty cone emanating from p with an angle Θ which means p is not well Θ-illuminated.

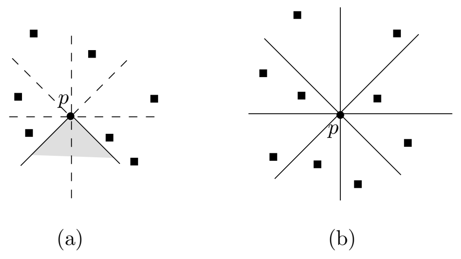


Figure 3: (a) Point p is not well $\frac{\pi}{2}$-illuminated because there is an empty cone with an angle of $\frac{\pi}{2}$. (b) Point p is well $\frac{\pi}{2}$-illuminated since there is a light source interior to each cone.

Now we explain how to make the logarithmic search to compute the MER of p. First, we compute the median of the distances between the light sources and p and divide F in two subsets. The subset F^c contains the $\lceil \frac{n}{2} \rceil$ closest light sources to p, while the set F^f contains the furthest half. Using the method described above, we are able to decide if the light sources of F^c are sufficient to well Θ-illuminate p. If they are then we can forget about the light sources of F^f and

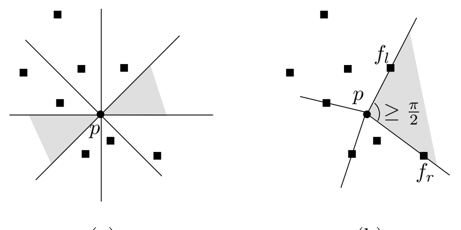


Figure 4: (a) There are two non-consecutive empty cones. (b) Point p is not well $\frac{\pi}{2}$-illuminated since there is an empty cone defined by p and the light sources f_l and f_r with an angle greater than $\frac{\pi}{2}$.

compute the median of the distances between the light sources of F^c and p. We reassign F^c to the closest half and repeat the previous method to check if the new set is still sufficient to well Θ-illuminate p. If F^c does not well Θ-illuminate p then we have to save the location of the empty cones. Since the light sources of F^c are not sufficient, we have to get some more of the set F^f that have not been used. We compute the median of the light sources of F^f and reassign F^c to the closest half of the latter. Now we will test the new set F^c just by checking if its new light sources are interior to the cones left empty in the last iteration. If they are not sufficient, then we repeat the procedure and try again with another half of F^f, otherwise we try half of the ones we have just used.

We repeat this search until $|F^c| = 1$. If p is well Θ-illuminated then the only light source of F^c is the closest embracing site for p. The MER of p is naturally given by the distance between p and its closest embracing site. All the light sources closer to p than its closest embracing site together with the closest embracing site form the minimal embracing set for p. Otherwise p cannot be well Θ-illuminated.

Theorem 2 *Given a set F of n light sources, a point p in the plane and an angle $\Theta \leq \pi$, checking if p is well Θ-illuminated, computing its* MER *and a minimal embracing set for it takes $\Theta(n)$ time.*

Proof. Let F be a set of n light sources, p a point in the plane and $\Theta \leq \pi$ a given angle. Divide the plane in cones with an angle of $\frac{\Theta}{2}$ and assign each light source to its cone takes $\mathcal{O}(n)$ time.

The distances from p to all the light sources can be computed in linear time. Computing the median also takes linear time [5], as well as splitting F in two halves. Since we consider the angle Θ to be a fixed value, the number of cones is constant ($\frac{1}{\Theta}$ is constant). Consequently, spreading each empty cone by computing a light source on each side of the cone is linear. So checking if p is well Θ-illuminated by a set $F^c \subset F$ is linear on the number of light sources of F^c. Note that we never study the same light source twice while searching for the MER of p. So the total time for this logarithmic search is $\mathcal{O}(n+\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+...) = \mathcal{O}(n)$. Therefore, we find a closest embracing site for p and a minimal embracing set in linear time.

All the light sources of F are candidates to be the closest embracing site for a point p in the plane, so in the worst case we have to study every one of them. Knowing this, we have $\Omega(n)$ as a lower bound which makes the linear complexity of this algorithm optimal. □

Note that this algorithm not only computes the minimal embracing set and the MER of a well Θ-illuminated point as it also computes them for a t-well illuminated point. A point p is said to be t-well illuminated if there are at least t light sources illuminating p in every open half plane passing through p. The next theorem solves the t-good illumination of minimum range [1] using the Θ-illumination of minimum range.

Proposition 3 *Given a set F of n light sources, a point p in the plane and a given angle $\Theta \leq \pi$, let r_m be the* MER *to well Θ-illuminate p. Then r_m is also the* MER *to t-well illuminate p, where $t = \lfloor \frac{\pi}{\Theta} \rfloor$.*

Proof. Let F be a set of n light sources, p a point in the plane and $\Theta \leq \pi$ a given angle. If p is well Θ-illuminated then we know that there is always one interior light source to every cone emanating from p with an angle Θ. On the other hand, p is t-well illuminated if there are, at least, t interior light sources to every half-plane passing through p. An half plane passing through p can be seen as a cone emanating from p with an angle π. So if we know that we have at least one light source in every cone emanating from p with an angle Θ then we know that we have at least $\lfloor \frac{\pi}{\Theta} \rfloor$ light sources in every half-plane passing through p. This means that p is $\lfloor \frac{\pi}{\Theta} \rfloor$-well illuminated. So the MER needed to well Θ-illuminate p also $\lfloor \frac{\pi}{\Theta} \rfloor$-well illuminates p. □

Corollary 4 *Let F be a set of n light sources, p a point in the plane and $\Theta \leq \pi$ a given angle. A minimal embracing set that well Θ-illuminates p also t-well illuminates p, where $t = \lfloor \frac{\pi}{\Theta} \rfloor$.*

Proof. Let F be a set of n light sources, p a point in the plane and $\Theta \leq \pi$ a given angle. According to the last proposition, the MER to well Θ-illuminate p also t-well illuminates it, $t = \lfloor \frac{\pi}{\Theta} \rfloor$. So the closest embracing site of p for both types of illumination is the same. Since the minimal embracing set for p is formed by the closest embracing site of p, $f_i \in F$, and by all the light sources closer to p than f_i then the minimal embracing set to well Θ-illuminate p is the same as the minimal embracing set to t-well illuminate p, where $t = \lfloor \frac{\pi}{\Theta} \rfloor$. □

Observation 1 *Note that if a point is well Θ-illuminated, it is also t-well illuminated for $t = \lfloor \frac{\pi}{\Theta} \rfloor$, however the other implication is not necessarily true as it is shown in Figure 5.*

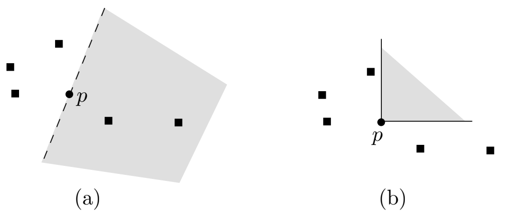


Figure 5: (a) Point p is 2-well illuminated since there are at least two light sources in ever open half plane passing through p. (b) Point p is not well $\frac{\pi}{2}$-illuminated because there is an empty cone with an angle of $\frac{\pi}{2}$.

3 Conclusions

In this paper we presented a generalization of the t-good illumination of minimum range [1] called good Θ-illumination of minimum range. We present an optimal linear time algorithm to compute the Minimum Embracing Range needed to well Θ-illuminate a given point p and a minimal embracing set for it. We also established a connection between both illumination concepts in Proposition 3. The MER to well Θ-illuminate a point is also the MER to t-well illuminate that point, for $t = \lfloor \frac{\pi}{\Theta} \rfloor$.

References

[1] M. Abellanas, A. Bajuelos, G. Hernández and I. Matos. *Good Illumination with Limited Visibility.* Proceedings of the International Conference of Numerical Analysis and Applied Mathematics 2005 (ICNAAM 2005), Wiley-VCH Verlag, 35–38.

[2] M. Abellanas, A. Bajuelos, G. Hernández, F. Hurtado, I. Matos and B. Palop. *Good Illumination of Minimum Range.* Submitted to the International Journal of Computational Geometry and Applications (2006).

[3] T. Asano, S. K. Ghosh and T. C. Shermer. *Visibility in the plane.* In Handbook of Computational Geometry, edited by in J.-R. Sack and J. Urrutia, (Elvevier, 2000), 829–876.

[4] D. Avis, B. Beresford-Smith, L. Devroye, H. Elgindy, E. Guvremont, F. Hurtado and B. Zhu. *Unoriented Θ-maxima in the plane: complexity and algorithms.* Siam J. Computation 28, no. 1, (1998), 278–296.

[5] M. Blum, R.W. Floyd, V. Pratt, R. Rivest and R. Tarjan. *Time bounds for selection.* Journal of Computer and System Sciences 7, (1973), 448–461.

[6] H. Kung, F. Luccio and F. Preparata. *On finding the maxima of a set of vectors.* Journal of ACM 22, (1975), 469–476.

[7] J. Urrutia. *Art Gallery and Illumination Problems.* In Handbook of Computational Geometry edited by in J.-R. Sack and J. Urrutia, (Elsevier 2000), 973–1027.

Good Illumination Maps

Narcís Coll[*] Marta Fort [*] Narcís Madern[*] J. Antoni Sellarès[*]

Abstract

Given a set P of points (lights) and a set S of segments (obstacles), the good illumination of a point q relative to P and S, describes the relationship between q and the distribution of the points in P from which q is illuminated taking into account the effect of the segments of S. A point q is t-well illuminated relatively to P and S if and only if every closed halfplane defined by a line through q contains at least t points of P illuminating q. The greater the number t the better the illumination of q. The good illumination depth of q is the maximum t such that q is t-well illuminated relatively to P and S. The good illumination map is the subdivision of the plane in good illumination regions where all points have the same fixed good illumination depth. In this paper we present algorithms for computing and efficiently drawing, using graphics hardware capabilities, the good illumination map of P and S.

1 Introduction

Given a set P of points, the location depth of a point q relative to P describes, intuitively, the relationship of q to the distribution of the points in P. A depth region is the locus of all points with the same fixed location depth. The *depth map* of P is the subdivision of the plane in depth regions.

The notion of illumination or visibility is an important topic in Computational Geometry. In some applications dealing with an environment of point lights and obstacles, several lights surrounding and illuminating points are needed. In this paper we use the *good illumination* concept introduced by Canales et. al [1, 3]. We will see that, in fact, good illumination combines two well studied concepts: illumination with obstacles and location depth. In [3], Canales studied 1-good illumination when the lights are located in the exterior of a convex polygon and 2-good illumination when lights are located at the vertices of a simple (convex or non-convex) polygon.

In this paper we extend the study of good illumination to the general case of a set P of points (lights) and a set S of segments (obstacles). The good illumination map is the subdivision of the plane in regions whose points have the same good illumination relative to P and S. Drawing the good illumination map of P and S helps to visualize the distribution of the points of P relative to the segments of S. We present algorithms for computing and efficiently drawing, using graphics hardware capabilities, the good illumination map of P and S.

2 Depth Maps

Let P be a set of n points. The location depth of an arbitrary point q relative to P, denoted by $ld_P(q)$, is the minimum number of points of P lying in any closed halfplane defined by a line through q. The k-th depth region of P, represented by $dr_P(k)$, is the set of all points q with $ld_P(q) = k$. For $k \geq 1$, the external boundary $dc_P(k)$ of $dr_P(k)$ is the k-th depth contour of P. The depth map of P, denoted $dm(P)$, is the set of all depth regions of P (see Figure 1). The complexity of $dm(P)$ is $O(n^2)$. This bound is tight, for example, when all points of P are in convex position. We denote $dm_r(P)$ the restriction of $dm(P)$ to a planar region r.

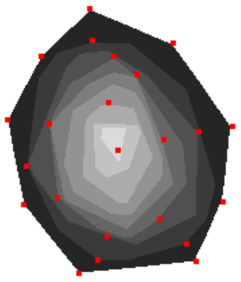


Figure 1: *Depth map of a set of points.*

2.1 Computing Depth Contours

Miller *et al* [7] present an algorithm for computing the depth contours for a set of points that makes an extensive use of duality, and proceeds as follows: given a set P of points, the algorithm maps them to their dual arrangement of lines. Then, a topological sweep is applied to find the planar graph of the arrangement and its vertices are labeled with their levels (the number of dual lines above them). The depth of a vertex can be computed using $\min(\text{level}(v), n-\text{level}(v)+1)$.

[*]Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain, `{coll,mfort,nmadern,sellares}@ima.udg.es`. Partially supported by grant TIN2004-08065-C02-02. Marta Fort and Narcis Madern are also partially supported by grant AP2003-4305 and BES-2005-9541 respectively.

Finally, for a given k, $dc_P(k)$ is computed by finding the lower and upper convex hulls of the vertices at depth k. Each such vertex corresponds to a halfplane in the primal plane, and $dc_P(k)$ is the boundary of the intersection of these halfplanes (which might be empty, in that case $dc_P(k)$ does not exist). The complexity of this algorithm, that has been shown to be optimal, is $O(n^2)$ in time and space.

Since for large n the $O(n^2)$ time of Miller *et al* algorithm could be too large, in [5, 6] an algorithm is presented that draws, using graphics hardware capabilities, an image of the depth contours as a set of colored pixels, where the color of a pixel is its depth value. The algorithm consists of two steps: in the first step, the input point set P is scan-converted to lines in the dual image plane. The algorithm runs on two bounded duals due to the finite size of the dual plane, in order to guarantee that all intersection points of the lines lie in this finite region. Since each dual plane is discrete, it is possible to compute the level of each pixel by drawing the region situated above every dual line of P, incrementing by one the stencil buffer for each region. In the second step, the two images formed by all the dual lines are scanned, and for each pixel on a dual line the corresponding primal line at the appropriate depth is rendered as a colored 3D graphics primitive using the z-buffer. The depth of each primal line is easily determined from the stencil buffer value and the line color must be distinct for each depth. The resulting rendered image (see Figure 1) contains the depth contours of the point set P as the boundaries between colored regions. This method can also be used for drawing the convex hull of a set of points P by introducing minor changes in the second step: when we scan the dual images, if a pixel has a level greater than zero we rasterize its primal line with depth one and using always the same color.

3 Good Illumination Maps

Let P be a set of n points and S be a set of m segments. We assume that no point in P belongs to the interior of a segment in S. The free space F_S relative to S is the complement of S. Given two points $q \in F_S$ and $p \in P$, we say that point p illuminates q if the interior of the segment with endpoints p and q remains completely inside F_S. A point q is t-well illuminated relatively to P and S if and only if every closed halfplane defined by a line through q contains at least t points of P that illuminate q. The good illumination depth of q relative to P and S, denoted by $gid_{P,S}(q)$, is the maximum t such that q is t-well illuminated relatively to P and S.

Lemma 1 *If P_q denotes the subset of points of P illuminating q, then $gid_{P,S}(q) = ld_{P_q}(q)$.*

The k-th good illumination region relative to P and S, denoted $gir_{P,S}(k)$, is the set of all points q with $gid_{P,S}(q) = k$. Observe that $gir_{P,S}(k)$ can be formed by several convex connected components (see Figure 2).

Lemma 2 *If S is empty or does not intersect the convex hull $CH(P)$ then $gir_{P,S}(k) = dr_P(k)$.*

We call the set of all good illumination regions relative to P and S the good illumination map of P and S and denote it with $gim(P,S)$.

Also we denote with $gim_r(P,S)$ the restriction of $gim(P,S)$ to region r.

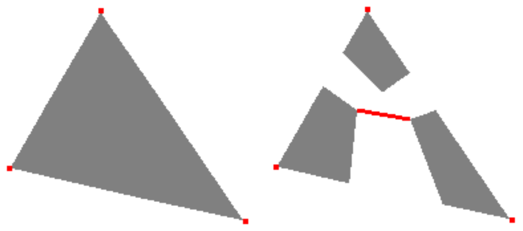


Figure 2: *In the left we have the good illumination map of a set with three points and an empty set of segments; the corresponding* 1*-good illumination region is represented in dark. In the right a segment obstacle has been added.*

4 Computing Good Illumination Maps

From now on we will focus on the non trivial case where $n \geq 3$, $m \geq 1$ and S intersects $CH(P)$.

Lemma 1 induces a way to compute $gim(P,S)$. First we decompose the free space F_S into illumination regions so that all points in a single connected such region are illuminated exactly by the same points in P. Then, in each illumination region we compute the depth map of its illuminating points.

Given a point $p \in P$ and a segment $s \in S$, the shadow region of s with respect to p, denoted $sr(p,s)$, is the set of points non illuminated from p when we consider segment s as an obstacle. Denote s_0, s_1 the endpoints of s. When $p \notin s$, $sr(p,s)$ is the region delimited by the segment s, the ray of origin s_0 and direction $\overrightarrow{ps_0}$ and the ray of origin s_1 and direction $\overrightarrow{ps_1}$. When p is an endpoint of s, for example s_0, the shadow region $sr(p,s)$ is the ray of origin p and direction $\overrightarrow{ps_1}$.

Let $\mathcal{A}(P,S)$ be the arrangement determined by the family of all shadow regions $sr(p,s)$ interior to $CH(P)$, for all $p \in P$ and $s \in S$. All cells in $\mathcal{A}(P,S)$ are convex and all points in a cell c of $\mathcal{A}(P,S)$ are seen from exactly the same subset P_c of points of P. Observe that two cells $c \neq c'$ that are seen from the same subset of points of P may exist, it is to say with $P_c = P_{c'}$.

Theorem 3 *The arrangement $\mathcal{A}(P,S)$ consist of $O(n^2m^2)$ cells and each cell has $O(n)$ illuminating points.*

Proof. Each shadow region is bounded by two rays and one segment, and the convex hull $CH(P)$ has $O(n)$ edges. Then, the arrangement $\mathcal{A}(P,S)$ has $O(nm)$ lines and $O((nm)^2)$ cells. Figure 3 proves that this upper bound is tight. In a) we can see a segment placed in the diameter of a circle and $n/2$ light points p_i placed on the circle and above the segment. The point p_i is put so that one ray of its shadow region intersects $i-1$ rays of all other shadow regions inside the circle and the free space. Then, the number of cells of the line arrangement is $\Omega(\sum_{i=1}^{n/2}(i-1)) = \Omega(n^2)$. In b) the segment is split in m segments. Since we have the same properties of a) for each one of the m segments, the new line arrangement has $\Omega((nm)^2)$ cells. In c) we have placed $n/2$ light points on the circle and under the segments. This placement assures that there are $\Omega((nm)^2)$ cells interior to the $CH(P)$ that see a minimum of $n/2$ illuminating points. Consequently $O(n^2m^2)$ is a well fitted upper bound of the $\mathcal{A}(P,S)$, and $O(n)$ a well fitted upper bound of the illuminating points of each cell.

□

For each cell c of $\mathcal{A}(P,S)$ with illuminated points set P_c, Lemma 1 states that $gim_c(P,S)$ can be computed as $dm_c(P_c)$, the depth map of the set P_c restricted to c. Then, we have:

$$gim(P,S)) = \bigcup_{c\in\mathcal{A}(P,S)} dm_c(P_c)\,.$$

Theorem 4 *The good illumination map of P and S can be computed in $O(n^4m^2)$ time.*

Proof. We associate a set P_c to each cell c of $\mathcal{A}(P,S)$. First, by using a topological sweep [4, 8], $\mathcal{A}(P,S)$ and the associated sets P_c can be computed in $O(n^3m^2)$ time. Next, for each cell c we compute $dm_c(P_c)$ by intersecting the convex cell c with the depth contours determined by $dm(P_c)$. This spends a time of $O(n^2)$ per cell (see section 2.1). Thus, the time needed to compute $gim(P,S)$ is $O((nm)^2n^2) = O(n^4m^2)$.

□

5 Drawing Good Illumination Maps

In this section we describe a method for drawing good illumination maps using GPU capabilities. The method, based on the fact that $gim(P,S)) = \bigcup_{c\in\mathcal{A}(P,S)} dm_c(P_c)$, proceeds in two steps.

First step. We start drawing $CH(P)$ on a black screen, as described in Section 2.1, and we store it in a texture. Next we rasterize the boundary, interior to $CH(P)$, of all shadow regions $sr(p,s)$, $p\in P$,

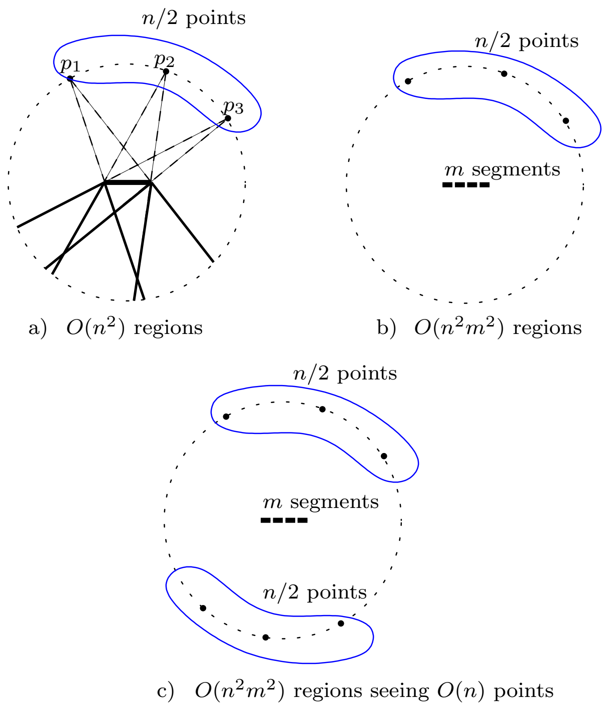


Figure 3: *Upper bound configuration.*

$s\in S$ in white and we transfer the frame buffer to a matrix in the CPU so that each element represents a pixel. Then we find all the cells of $\mathcal{A}(P,S)$ using a CPU based growing method as follows. We take any black pixel of the matrix and we choose an unused color. We paint this pixel with the current color and then we visit its four surrounding pixels. If the visited pixel is white (belongs to the boundary) we store its neighbors in a waiting list and we continue visiting and painting pixels until we have visited a entire cell. While there are pixels in the list, we take the first pixel and if it is black we repeat the process from this position, otherwise the pixel is rejected. By doing this we paint each cell with a different color. During the process we store an interior pixel of each cell and its color. Finally, for each cell c we determine the set P_c by taking the interior pixel of c and drawing the shadow regions defined by the pixel and the m segments of S in white on a black screen. By doing this, a point p illuminates the cell c if its corresponding pixel is black. We obtain the set P_c by checking if the pixel corresponding to each point in P is colored black. Moreover, we assign a distinct color to each different subset P_c so that all cells illuminated by P_c will get the same color. By doing this we ensure that we paint the same depth map at most once in the second step.

Second step. For each cell $c\in\mathcal{A}(P,S)$ we draw $dm_c(P_c)$ using the algorithm described in Section 2.1 that draws depth contours. In order to paint only the pixels inside c we use a fragment shader. The inputs of this fragment shader are the arrangement $\mathcal{A}(P,S)$

represented as a texture and the color assigned to P_c. The fragment shader only paints a pixel (x, y) if the color in the position (x, y) of the texture representing $\mathcal{A}(P, S)$ is equal to the color of c, since in this case the pixel is inside the cell c.

5.1 Results

We have implemented the proposed method using C++ and OpenGL, and all the tests and images have been carried out on a Intel(R) Pentium(R) D at 3GHz with 2GB of RAM and a GeForce 7800 GTX/PCI-e/SSE2 graphics board.

Figures 2, 4 and 5 show some examples of good illumination maps obtained using our implementation. In these figures the points are colored in a grey gradation according to its good illumination depth (black corresponds to level one), however pure white color shows level zero.

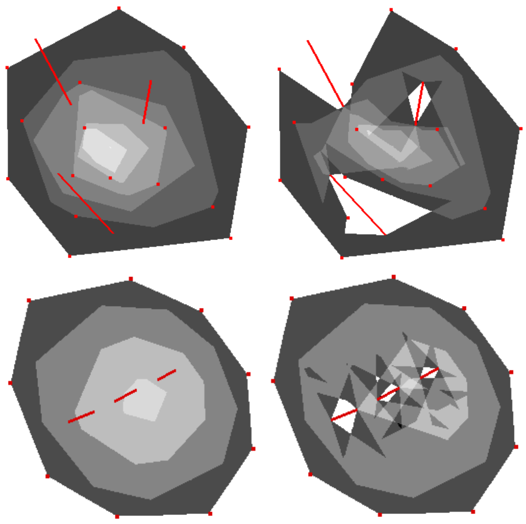


Figure 4: *Good illumination maps of two different configurations of points and segments. On the left we show the depth maps of the points. On the right we show the good illumination maps of the points and the segments.*

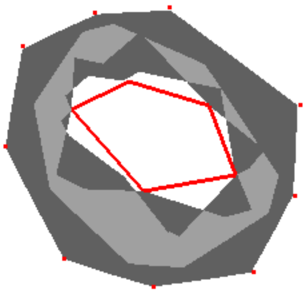


Figure 5: *Good illumination map of a set of points in convex position and a convex obstacle polygon.*

6 Future Work

We are extending our work on good illumination to the case of points modelling source lights of restricted illumination, for example emiting light within an angular region or/and with limited range [2].

We are studying the possibility of developing a more efficient algorithm for computing good illumination maps that, as in the case of depth maps, will work entirely in dual space. We are also interested on a more efficient algorithm for some particular cases as the ones studied in [1, 3].

References

[1] M. Abellanas, S. Canales and G. Hernández. Buena iluminación, *Actas de las IV Jornadas de Matemática Discreta y Algorítmica*, (2004), 239-246.

[2] M. Abellanas, A. Bajuelos, G. Hernández and I. Matos. Good Illumination with Limited Visibility. *Proc. International Conference of Numerical Analysis and Applied Mathematics*, Wiley-VCH Verlag, (2005), 35-38.

[3] S. Canales. Métodos heurísticos en problemas geométricos, Visibilidad, iluminación y vigilancia. *Ph.D. thesis, Universidad Politécnica de Madrid*, 2004.

[4] H. Edelsbrunner and L. Guibas. Topological sweeping an arrangement. *J. Comput. System. Sci.* 38 (1989), 165–194.

[5] Fischer I. and Gotsman C. Drawing Depth Contours with Graphics Hardware. Proceedings of Canadian Conf. on Comp. Geometry, (2006), 177–180.

[6] S. Krishnan, N. Mustafa, and S. Venkatasubramanian. Hardware-assisted computation of depth contours. *Proc. thirteenth ACM-SIAM symposium on Discrete algorithms*, (2002), 558–567.

[7] K. Miller, S. Ramaswami, P. Rousseeuw, J.A. Sellarès, D. Souvaine, I. Streinu and A. Struyf. Fast implementation of depth contours using topological sweep. *Statistics and Computing*, (2003), 13:153–162.

[8] E. Rafalin, D. Souvaine, I. Streinu. Topological Sweep in Degenerate cases. *Proc. of the 4th international workshop on Algorithm Engineering and Experiments*, ALENEX 02, in LNCS 2409, Springer-Verlag, Berlin, Germany, (2002), 155–156.

StrSort Algorithms for Geometric Problems

Christiane Lammersen [*] Christian Sohler [†]

Abstract

In the StrSort model [2], the input is given as a stream, e.g. a sequence of points, and an algorithm can perform (a) *streaming* and (b) *sorting passes* to process the stream. A streaming pass reads the input stream from left to right and writes an output stream, which is the input of the next pass. A sorting pass is a black box operation that sorts a stream according to some partial order.

In this paper, we develop algorithms for two basic geometric problems in the StrSort model. At first, we propose a divide-and-conquer algorithm that computes the *convex hull* of a point set in $2D$ in $\mathcal{O}(\log^2 n)$ passes using $\mathcal{O}(1)$ memory. Then we give a StrSort algorithm to compute a $(1+\epsilon)$-spanner for a point set in $\mathbb{R}^d$ for constant d and constant ϵ that uses $\mathcal{O}(\log^{d-1} n)$ passes and $\mathcal{O}(\log n)$ space. This result implies a $(1+\epsilon)$-approximation of the Euclidean minimum spanning tree in $\mathbb{R}^d$, for constant d and ϵ.

1 Introduction

In the last few years, we see ourselves more and more often confronted with massive data sets, e.g. the Web Graph, IP traffic logs, clickstreams, or the human genome. Often these data sets are of geometric nature such as data sets arising in geographical information systems, astrophysics, and sensor networks.

In many cases, these data sets are too large to be analyzed with traditional algorithms and sometimes even external memory algorithms are too slow. One approach to deal successfully with the data is *streaming*. In streaming, the data items arrive sequentially in worst case order. In this model, we typically do not want to store the whole data, but only a summary of it. However, streaming is also interesting when the data can be stored in secondary memory because the fastest way to access this data is to do a sequential scan. It is also possible to write an output stream at the same time, e.g., by writing to another hard disk, or sending the data to another computer. In this context the W-stream model emerged [7], where one looks at the possible tradeoffs between memory usage and number of passes over the data. However, it seems that to solve more difficult problems in few passes one has still relatively high memory requirements. So a natural question is to consider a model that supports one basic non-local operation on a data stream efficiently. A canonical candidate for such a non-local operation is *sorting*. Sorting is one of the most basic algorithms and highly optimized implementations for large data sets exist. The combination of streaming and sorting is known as the StrSort model and has been introduced in [2].

In this paper, we develop geometric algorithms in the StrSort model. A first such algorithm has been given in [2], where the authors showed how to compute the number of intersections between a set of red and blue line segments.

Related Work. The convex hull problem has been widely considered in the streaming model. In [6] the authors introduced the radial histogram [6], which can be used to approximate the convex hull of a stream of points in the plane with an error of $\mathcal{O}(D/r)$, where r is the sample size and D the diameter of the sample. Hershberger and Suri proposed an adaptive sampling algorithm that reduces the error to $\mathcal{O}(D/r^2)$ [8]. The convex hull problem has also been considered in higher dimensions [1]. Chan and Chen gave exact algorithms to compute convex hulls in data streams [5]. If the input stream consists of n points, that are sorted according to their x-coordinate, their algorithm needs a constant number of passes and $\mathcal{O}(n^{1/2+\delta})$ space for any fixed constant $\delta > 0$. If the input is not sorted, they need $\mathcal{O}(hn^\delta)$ space, where h is the output size.

Suri et al. gave ϵ-approximation algorithms for rectangle and halfplane ranges in d-dimensions [13]. Bagchi et al. proposed a deterministic sampling algorithm to maintain ϵ-nets and ϵ-approximations with polylogarithmic memory requirements [4]. Shahabi and Sharifzadeh showed how to approximate Voronoi cells for fixed two-dimensional site points when the location of the neighbouring sites arrive as a data stream [12]. For other work on streaming algorithms, we refer to the survey by Muthukrishnan [10].

The construction of a t-spanner for a real constant $t > 1$ with $\mathcal{O}(n)$ edges has lead to the definition of the Θ-graph. Ruppert and Seidel [11] and Arya et al. [3] developed $\mathcal{O}(n \log^{d-1} n)$ time algorithms based on the Θ-graph that construct a $(1+\epsilon)$-spanner for a set of n points in $\mathbb{R}^d$.

[*]Heinz Nixdorf Institute and Department of Computer Science, University of Paderborn, `basilisk@upb.de`

[†]Heinz Nixdorf Institute and Department of Computer Science, University of Paderborn, `csohler@upb.de`

2 Preliminaries

Let $\varrho = \langle p_1, \dots, p_n \rangle$ be a sequence of points in $\mathbb{R}^d$, i.e., the *input stream*. We will assume that d is a constant. As usually done in Computational Geometry, we consider an algebraic model of computation, i.e., all computations are done with exact precision arithmetic and we can always store a point coordinate or alternatively an integer with $\mathcal{O}(\log n)$ bits in one memory cell. We further assume that standard arithmetic operations can be done in constant time.

In the StrSort model, an algorithm is allowed to perform two types of passes, a *streaming pass* and a *sorting pass* [2]. A streaming pass reads an input stream ϱ point by point without ever going back and at the same time writes an output stream $Str(\varrho)$. A sorting pass rearranges the input stream according to a partial order $\leq$ on the items, where we assume that for every pair of constant size (in terms of memory cells) items p, q we can decide whether $p < q$, $p > q$ or $p = q$ using a machine that only gets p, q as input.

Between two passes the local memory is maintained. We require that any intermediate stream has length $\mathcal{O}(n)$. We say that a StrSort algorithm is *pass-efficient*, if it uses at most $\log^{\mathcal{O}(1)} n$ passes, and it is *space-efficient*, if it uses at most $\log^{\mathcal{O}(1)} n$ memory cells. Here we assume that the sorting pass is a black box operation that does not require local memory.

The Θ-Graph. Let P be a set of points in $\mathbb{R}^d$. Let $G = (P, E)$ be an undirected graph whose vertices are the points of P. The length of an edge is given by the Euclidean distance between its endpoints. G is called a t-spanner for P, if for any pair $u, v \in P$ there exists a path in G from u to v having length at most t-times the Euclidean distance between u and v. To compute a $(1+\epsilon)$-spanner for a given point set P, we use the construction of the Θ-graph.

Let $k > 2$ be an integer and let $\Theta = 2\pi/k$ be an angle. Let $\mathcal{C}$ be a set of cones with the following property: The apex of each cone $C \in \mathcal{C}$ is at the origin and there exists a halfline l_C having the endpoint at the origin such that the angle between l_C and every halfline of the superficies surface of C is at most $\Theta/2$, and the union of the cones covers the whole space. With the method of Lukovszki [9], it is possible to construct a set $\mathcal{C}$ of $\mathcal{O}((\frac{d^{3/2}}{\Theta})^{d-1})$ simplicial cones which have this property. For every vertex $u \in P$, we devide the space into cones by translating every cone of $\mathcal{C}$ such that its apex is at u. We denote the translated cone by C_u and the translated halfline by $l_{C,u}$. To obtain the edges of the Θ-graph, we connect every point $u \in P$ for each cone $C \in \mathcal{C}$ to the point $v \in C_u \cap P \backslash \{u\}$ whose orthogonal projection onto $l_{S,u}$ is closest to u.

The spanner property of the Θ-graph is proven for an angle $0 < \Theta < \pi/3$:

Lemma 1 ([11, 9]) *Let $P \subset \mathbb{R}^d$ be a set of points. Let $k > 6$ be an integer constant and $\Theta = 2\pi/k$. Then the graph $G_\Theta(P)$ is a t-spanner for $t = \frac{1}{1-2\sin(\Theta/2)}$.*

A direct consequence of this lemma is

Corollary 2 *Let $P \subset \mathbb{R}^d$ be a set of points, d a constant. Then for every $0 \leq \epsilon \leq 1$ there is $\Theta = \Omega(\epsilon)$ such that the graph $G_\Theta(P)$ is a $(1+\epsilon)$-spanner.*

3 Convex Hull

In this section, we present an efficient StrSort algorithm that computes the convex hull $\mathcal{CH}(P)$ of a set of n points $P \subseteq \mathbb{R}^2$ in $\mathcal{O}(\log^2 n)$ passes using $\mathcal{O}(1)$ memory cells. We first give a high level description of the algorithm. Our algorithm proceeds bottom up. First, we sort the input points according to their x-coordinate resulting in a sequence $\langle p_1, \dots, p_n \rangle$. Each point p_i builds a group G_i, so that we obtain the output stream $\langle \mathcal{CH}(G_1), \dots, \mathcal{CH}(G_n) \rangle$. Then in every step, we merge all pairs of consecutive convex hulls, i.e. after the first such step we obtain the output stream $\langle \mathcal{CH}(G_1 \cup G_2), \mathcal{CH}(G_3 \cup G_4), \dots \rangle$. We continue this until we have found the convex hull of $\bigcup_{i=1}^k G_i = P$.

It is well-known that two convex hulls can be merged in linear time. However, the classical merge algorithm does not seem to carry over to the streaming model, because it requires *interleaved* access to the right and left convex hull.

The Merge Step. We first describe the merge step without considering streaming. To merge a pair of consecutive convex hulls, we have to find the tangent of the upper hulls and the tangent of the lower hulls. For symmetry reasons, it is sufficient to show how to find the osculation point of the left upper hull.

Let us consider a pair of convex hulls $\langle \mathcal{CH}(G_i), \mathcal{CH}(G_{i+1}) \rangle$. We denote by $u_1, \dots, u_m$ the vertices of the upper hull of $\mathcal{CH}(G_i)$. Further let $\mathcal{H}_j$ denote the halfspace induced by a line through u_j and u_{j+1} that does not contain $\mathcal{CH}(G_i)$. We say that $\mathcal{CH}(G_{i+1})$ is *visible* from a vertex u_j of $\mathcal{CH}(G_i)$, if $\mathcal{H}_j \cap \mathcal{CH}(G_{i+1})$ is non empty (see Figure 1).

Observation 1 *For $2 \leq j \leq m$ the point u_j is osculation point of the upper left hull, iff $\mathcal{CH}(G_{i+1})$ is visible from u_j and $\mathcal{CH}(G_{i+1})$ is not visible from u_{j-1}.*

We first check whether $\mathcal{H}_1 \cap \mathcal{CH}(G_{i+1})$ is non empty. If this is the case, u_1 is the osculation point for the upper convex hull. Otherwise, we use the following binary search procedure to find the osculation point. Let $[l, r]$ denote the interval of point indices that contains the osculation point. We start with

$j = (l + r)/2$. If u_j is not an osculation point then either (a) $\mathcal{CH}(G_{i+1})$ is visible from both u_j and u_{j-1} or (b) $\mathcal{CH}(G_{i+1})$ is not visible from both u_j and u_{j-1}. In case (a) the osculation point lies between u_l and u_{j-1}. Otherwise, it is between u_{j+1} and u_r.

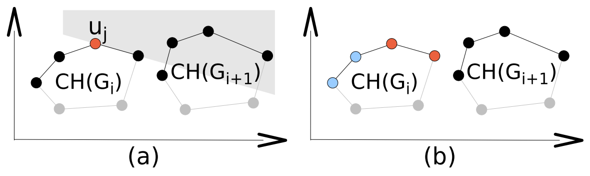


Figure 1: **(a)** Illustration of visibility. **(b)** Illustration of Observation 1. $\mathcal{CH}(G_{i+1})$ is visible from the red vertices.

3.1 The StrSort Algorithm

In this section, we describe the StrSort algorithm for convex hull computation.

Initialization. We start by sorting the points by x-coordinate resulting in a sequence $\langle p_1, \ldots, p_n \rangle$. Then we replace each point p_i by a quadruple (i, α, m, p), where initially $m = 1$ and $\alpha = 0$. This will be our representation for the stream $\langle \mathcal{CH}(G_1), \ldots, \mathcal{CH}(G_n) \rangle$. In general, we have for each vertex p of a convex hull $\mathcal{CH}(G_i)$ one quadruple (i, α, m, p), where m denotes the position in a clockwise ordering of the vertices of $\mathcal{CH}(G_i)$ starting with the vertex with minimum x-coordinate. From now on, we will only use sorting passes that compute a lexicographically increasing order of these quadruples.

Implementing the merge step. The main idea is to perform the merge step in parallel for each consecutive pair of convex hulls in our input stream. Let us consider an arbitrary pair of consecutive convex hulls $\langle \mathcal{CH}(G_i), \mathcal{CH}(G_{i+1}) \rangle$. To implement the binary search for the upper left osculation point, we use the following invariant. The left and right node of the current search interval have α-value 0. All other nodes get an α-value of 1. This way the search interval appears first in a lexicographically sorted sequence. When we pass over the input stream we can compute from the index of the first two nodes, for which vertex we have to perform a visibility test. Let p_j denote this point. The visibility test is done as follows. We store p_j, its predecessor p_{j-1} and successor p_{j+1} in the local memory. Then we only have to test whether $\mathcal{CH}(G_{i+1})$ is visible from p_{j-1} and p_j, i.e., whether a vertex of $\mathcal{CH}(G_{i+1})$ is contained in the halfspaces $\mathcal{H}_{j-1}$ and $\mathcal{H}_j$, respectively. Hence, once we have seen all points of $\mathcal{CH}(G_{i+1})$ we know the result. We write all quadruples unchanged in the output stream, but once we decided the visibility test, we output an additional "dummy point" $(i, -\infty, r, 0)$, where r is an encoding of the result of the test. After we have computed the output stream, we sort it lexicographically. This way, the "dummy point" occurs first among the points of $\mathcal{CH}(G_i)$ and we can use it to adjust the search interval. The above operations can be carried out for all pairs of consecutive hulls during a single pass.

In this way, we can compute the four osculation points using $\mathcal{O}(\log n)$ passes. Now, we can easily identify the points that are vertices of $\mathcal{CH}(G_i \cup G_{i+1})$, the points of $\mathcal{CH}(G_i)$ whose indices are not larger than the index of the upper osculation point or not smaller than the index of the lower osculation point and the points of $\mathcal{CH}(G_{i+1})$ whose indices are between the indices of the corresponding osculation points. After removing all the other points, we can compute the center of gravity and sort the remaining points clockwise around it to compute the new ordering of them.

The algorithm needs $\mathcal{O}(\log^2 n)$ passes and $\mathcal{O}(1)$ memory. Intermediate passes are at most a constant factor larger than the input stream. Hence, we obtain

Theorem 3 *Let ϱ be a stream of n points in $\mathbb{R}^2$. Then there exists a StrSort algorithm that computes the convex hull $\mathcal{CH}(\varrho)$ in $\mathcal{O}(\log^2 n)$ passes with memory requirements of $\mathcal{O}(1)$ cells.*

4 Geometric $(1 + \epsilon)$-Spanners

In this section, we briefly describe our StrSort algorithm to compute a $(1 + \epsilon)$-spanner for a point set P in $\mathbb{R}^d$. We will use the following spanner construction as described in [3]. Let $C \in \mathcal{C}$ be any cone and let $h_1, h_2, \cdots, h_d$ be the bounding hyperplanes of C. Furthermore, let $H_1, H_2, \cdots, H_d$ be lines that run through the origin and are orthogonal to $h_1, h_2, \cdots, h_d$ respectively. $H_1, \cdots, H_d$ are the axes of a transformed coordinate system. We choose their directions such that the cone C is given by $C = \{p \in \mathbb{R}^d \mid \forall 1 \leq i \leq d : p^{(i)} \geq 0\}$, where $p^{(i)}$ denotes the coordinate of p with respect to the i-th transformed coordinate axes. To find the neighbor of u in the cone C_u, we have to ask for the point $p \in P$ with $p^{(i)} \geq u^{(i)}$ for all $1 \leq i \leq d$ whose orthogonal projection onto $l_{C,u}$ is closest to u. Let c be the gradient vector of $l_{C,u}$. Then the neighbor of u in C_u is the point $p \in C_u$ that minimizes $c^T p$.

Our StrSort algorithm computes in parallel for each cone $C \in \mathcal{C}$ the neighbor of every point $u \in P$. For that purpose, we create $|\mathcal{C}|$ streams. For each stream the algorithm determines the neighbors of every point $u \in P$ with respect to one cone $C \in \mathcal{C}$. For an arbitrary cone $C \in \mathcal{C}$, we can focus on the following problem. We are given a stream $\langle p_1, \ldots, p_n, q_1, \ldots, q_n \rangle$ of n points followed by n queries. For each query $q_i \in \mathbb{R}^d$ we want to find the point $p_j \in P$ that minimizes $c^T p_j$

under the constraint that $p_j^{(\ell)} > q_i^{(\ell)}$ for all $1 \leq \ell \leq d$ and for some fixed direction c. We will refer to this problem as *the orthogonal range optimization problem.* We call the area $\{(x^{(1)}, \ldots, x^{(d)}) \mid \forall 1 \leq i \leq d : x^{(i)} > q^{(i)}\}$ the *range* of query q.

The Range Optimization Problem in 1D. We first attach to the points and queries an identifier to distinguish between them. Then we sort decreasingly. We pass over the stream and maintain the point p that minimizes $c^T p$ among the points seen so far. Everytime we reach a query q_i we output the pair (q_i, p). Clearly, p is greater than q_i and minimizes $c^T p$.

Theorem 4 *The range optimization problem in 1D can be solved in $\mathcal{O}(1)$ passes using $\mathcal{O}(1)$ memory cells.*

The Range Optimization Problem in 2D. We only give a high level description of the algorithm. The algorithm works somewhat similar to range trees. First it computes a balanced binary search tree sorted by x-coordinates. Let $v(p)$ denote the node of the search tree that corresponds to point p. Our procedure to answer n range queries consists of $\mathcal{O}(\log n)$ rounds. We think of each query as a pebble on the binary search tree. Let $Q(q)$ be the pebble corresponding to query q. At the beginning all pebbles start at the root of the tree and in every round they move one step on their search pass in the tree, i.e. we maintain the invariant that, at the beginning of each round, all pebbles are in the same level. Each pebble $Q(q)$ remembers its "current optimum", i.e. the point $p \in C_q$ with the smallest $c^T p$ value seen so far.

Now, we consider one round of the algorithm. Let $Q(q)$ be a pebble that is currently placed on the node $v(p)$ of the search tree. If $q^{(1)} \geq p^{(1)}$ then the pebble moves to the right child of $v(p)$. No other updates have to be performed. If $q^{(1)} < p^{(1)}$ then the pebble moves to the left child of $v(p)$. If $p \in C_q$ *and* the objective value $c^T p$ is smaller than the current optimum maintained with $Q(q)$, we replace the current optimum by p. Additionally, we have to check the points in the right subtree of $v(p)$. We know that any point r stored in that subtree satisfies $r^{(1)} > q^{(1)}$ and so we only have to solve a $1D$ range searching problem. To solve this, we create for all queries that move to the left subtree a $1D$ query for the point set in the right subtree. Then we run the StrSort algorithm for the $1D$ case on this problem and replace the current optimum of a query, if necessary. Then the pebbles are moved to the next level and the round ends.

Obviously, after $\mathcal{O}(\log n)$ rounds the pebbles have reached the bottom level of the search tree. Since any point in the range of a query q is stored on the search path of q or in a tree right of the search path, we know that the "current optimum" of $Q(q)$ is the correct answer to the corresponding query.

The $2D$ algorithm can be generalized to higher dimensions, so that we get the following result

Theorem 5 *There is a StrSort algorithm that computes a $(1+\epsilon)$-spanner in $\mathbb{R}^d$ using $\mathcal{O}(\log^{d-1} n)$ passes and $\mathcal{O}(\log n)$ memory.*

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating Extent Measures of Points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004.

[2] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the Streaming Model Augmented with a Sorting Primitive. *Proc. 45th IEEE Sympos. Found. Comput. Sci.*, pp. 540–549, 2004.

[3] S. Arya, D. M. Mount, and M. Smid. Dynamic Algorithms for Geometric Spanners of Small Diameter: Randimized Solutions. *Int. J. Comput. Geom. Applications*, 13(2):91–107, 1999.

[4] A. Bagchi, A. Chaudhary, D. Eppstein, and M. T. Goodrich. Deterministic Sampling and Range Counting in Geometric Data Streams. *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pp. 144-151, 2004.

[5] T. M. Chan and E. Y. Chen. Multi-Pass Geometric Algorithms. *Proc. 21st Annu. ACM Sympos. Comput. Geom.*, pp. 180–189, 2005.

[6] G. Cormode and S. Muthukrishnan. Radial Histograms for Spatial Streams. *DIMACS Technical Report 2003-11*, 2003.

[7] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off Space for Passes in Graph Streaming Problems. *Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algorithms*, 2006.

[8] J. Hershberger and S. Suri. Convex Hulls and Related Problems in Data Streams. *Proc. ACM/DIMACS Workshop on Management and Processing of Data Streams*, 2003.

[9] T. Lukovszki. New Results on Geometric Spanners and Their Applications. *Dissertation*, University Paderborn, 1999.

[10] S. Muthukrishnan. Data streams: Algorithms and applications (invited talk at SODA03). *http:// athos.rutgers.edu/ muthu/stream-1-1.ps*, 2003.

[11] J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. *Proc. 3rd Canad. Conf. Comput. Geom.*, pp. 207–210, 1991.

[12] C. Shahabi and M. Sharifzadeh. Approximate Voronoi Cell Computation on Geometric Data Streams. *Technical Report 04-835*, Computer Sci. Dep. University of Southern California, 2004.

[13] S. Suri, C. D. Toth, and Y. Zhou. Range Counting over Multidimensional Data Streams. *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pp. 160-169, 2004.

I/O-Efficient Map Overlay and Point Location in Low-Density Subdivisions

Mark de Berg* Herman Haverkort* Shripad Thite* Laura Toma†

Abstract

We present improved and simplified I/O-efficient algorithms for two problems on planar low-density subdivisions, namely map overlay and point location. More precisely, we show how to preprocess a low-density subdivision with n edges in $O(sort(n))$ I/O's into a compressed linear quadtree such that one can:

(i) compute the overlay of two such preprocessed subdivisions in $O(scan(n))$ I/O's, where n is the total number of edges in the two subdivisions,

(ii) answer a single point location query in $O(\log_B n)$ I/O's and k batched point location queries in $O(scan(n) + sort(k))$ I/O's.

For the special case where the subdivision is a fat triangulation, we show how to obtain the same bounds with an ordinary (uncompressed) quadtree, and we show how to make the structure fully dynamic using $O(\log_B n)$ I/O's per update. Our algorithms and data structures improve on the previous best known bounds for general subdivisions both in the number of I/O's and storage usage, they are significantly simpler, and several of our algorithms are cache-oblivious.

1 Introduction

The traditional approach to algorithms design considers each atomic operation to take roughly the same amount of time. Unfortunately this simplifying assumption is invalid when the algorithm operates on data stored on disk: reading data from or writing data to disk can be a factor 100,000 or more slower than doing an operation on data that is already present in main memory. Thus, when the data is stored on disk it is usually much more important to minimize the number of disk accesses rather than CPU operations.

This has led to the study of so-called I/O-efficient algorithms, also known as external-memory algorithms. The standard way of analyzing such algorithms is with the model introduced by Aggarwal and Vitter [1]. In this model, a computer has an internal memory of size M and an arbitrarily large disk. The data on disk is stored in blocks of size B, and whenever an algorithm wants to work on data not present in internal memory, the block(s) containing the data are read from disk. The I/O-complexity of an algorithm is the number of I/O's it performs—that is, the number of block transfers between internal memory and disk. Scanning—reading a set of n consecutive items from disk—takes $scan(n) = \lceil n/B \rceil$ I/O's, and sorting takes $sort(n) = \Theta((n/B)\log_{M/B}(n/B))$ I/O's.

One of the main application areas for I/O-efficient algorithms is the area of geographic information systems (GIS), because GIS typically work with massive amounts of data and loading all of it into memory is often infeasible. In GIS data for a particular geographic region is stored as a number of separate thematic layers. There can be a layer storing the road network, a layer storing the river network, a layer storing a subdivision of the region according to land usage or soil type, and so on. To combine information from two such layers—for example to find the crossings between the river network and the road network—one has to compute the overlay of the layers.

Background. The problem of map overlay can be formulated as a red-blue intersection problem: given a set of non-intersecting blue segments and a set of non-intersecting red segments in the plane, compute all intersections between the red and blue segments. Arge et al. show how to do this in $O(sort(n) + k/B)$ I/O's, where k is the number of intersections [2]. This is optimal in the worst case, but it is not satisfactory: the algorithm is complicated and uses $\Theta(n \log_{M/B}(n/B))$ storage. Crauser et al. [4] describe a randomized solution with the same (expected) bound of $O(sort(n) + k/B)$ I/O's and linear space under some realistic assumptions on M, B and n. Whether this algorithm is practical is not clear.

Although the I/O-complexity of the above algorithms is optimal for general sets of line segments, there are important special cases for which this is not clear. For example, in internal memory one can overlay two subdivisions in $O(n+k)$ time when these subdivisions are connected [6]. This brings us to the topic of our paper: is it possible to do the overlay of two planar maps in $O(scan(n + k))$ I/O's?

We will describe solutions based on modifications

*Dept. of Mathematics and Computer Science, Technische Universiteit Eindhoven, the Netherlands, mdberg@win.tue.nl, cs.herman@haverkort.net, sthite@win.tue.nl. MdB and ST were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

†Dept. of Computer Science, Bowdoin College, Brunswick ME, USA, ltoma@bowdoin.edu

of the so-called *linear quadtree*, introduced by Gargantini [8]. The linear quadtree is a quadtree variant where only the leaf regions are stored, and not the internal nodes. To facilitate a search in the quadtree, a linear order is defined on the leaves based on some space-filling curve; then a B-tree is constructed on the leaves using this ordering—see Section 2 for details. The idea of using linear quadtrees to store planar subdivisions has been used by Hjaltason and Samet [9]. They present algorithms for constructing the quadtree and for insertions. Although their experiments indicate their method performs well in practice, the I/O-complexity of their algorithms is not fully understood and does not seem to be worst-case-optimal. Furthermore, the stopping rule for splitting quadtree cells is based on user-defined parameters, so their method is not fully automatic.

Our results. We show how to overcome these disadvantages for two types of subdivisions: fat triangulations and low-density subdivisions [5]. A *δ-fat triangulation* is a triangulation in which every angle is bounded from below by a fixed positive constant δ. A *λ-low-density subdivision* is a subdivision such that any disk D is intersected by at most λ edges whose length is at least the diameter of D, for some fixed constant λ. We believe these two types of subdivisions are representative for most subdivisions encountered in practice, for reasonable values of δ and λ.

We present improved external-memory algorithms for map overlay and point location for such subdivisions. Our results are based on quadtrees which we define to ensure that (i) each leaf intersects only a constant number of edges of the subdivision, (ii) that we create only $O(n)$ leaves, and (iii) that we can construct the leaves efficiently. To store the quadtrees, we combine the ideas of compressed quadtrees and linear quadtrees to get a *linear compressed quadtree.* Our algorithms to construct the quadtrees are simple and elegant—simpler than the algorithm of Hjaltason and Samet [9]—and use only $O(sort(n))$ I/O's.

Our other results then come almost for free: overlaying two subdivisions boils down to a simple merge of the ordered lists of quadtree leaves taking $O(scan(n))$ I/O's, point location can be done in $O(\log_B n)$ I/O's by searching in the B-tree built on top of the list of quadtree leaves, and k batched point location queries can be done in $O(scan(n)+sort(k))$ I/O's by sorting the points along the space-filling curve and merging the sorted list with the list of quadtree leaves. The structure for fat triangulations can be made fully dynamic at the cost of $O(\log_B n)$ I/O's per update.

All our data structures and query algorithms can be made cache-oblivious [7] by plugging in the cache-oblivious variants of the various building blocks used, so that no tuning for B and M is necessary. For triangulations, also the construction and update algorithms can be made cache-oblivious, except that updates then take $O(\log_B n + \frac{1}{B}\log^2 n)$ I/O's.

In this abstract we assume that our inputs are subdivisions of the unit square $[0,1]^2$.

2 Our solution for fat triangulations

Theorem 1 *Let $\mathcal{F}$ be a δ-fat triangulation with n edges. Knowing the memory size M and the block size B, we can construct, in $O(sort(n/\delta^2))$ I/O's, a linear quadtree for $\mathcal{F}$ with $O(n/\delta^2)$ cells such that each cell intersects $O(1/\delta)$ triangles and the total number of intersections between cells and triangles is $O(n/\delta^2)$. With this structure we can do the following:*

(i) **Map overlay:** *Given two δ-fat triangulations with n triangles in total, each stored in such a linear quadtree, we can find all pairs of intersecting triangles in $O(scan(n/\delta^2))$ I/O's.*

(ii) **(Batched) point location:** *for any query point p we can find the triangle of $\mathcal{F}$ that contains p in $O(\log_B n)$ I/O's, and for any set P of k query points we can find for each point $p \in P$ the triangle of $\mathcal{F}$ that contains p in $O(scan(n/\delta^2) + sort(k))$ I/O's.*

(iii) **Updates:** *Inserting a vertex, moving a vertex, deleting a vertex, and flipping an edge can all be done in $O((\log_B n)/\delta^4)$ I/O's.*

The quadtree subdivision for fat triangulations A quadtree is a hierarchical subdivision of the unit square into quadrants, where the subdivision is defined by a criterion to decide what quadrants are subdivided further, and what quadrants are leaves of the hierarchy. A *canonical square* is any square that can be obtained by recursively splitting the unit square into quadrants. For a canonical square σ, let $mom(\sigma)$ denote its parent, that is, the canonical square that contains σ and has twice its width. The leaves of the quadtree form the quadtree subdivision; that is, a *quadtree subdivision* for a set of objects in the unit square is a subdivision into disjoint canonical squares (*quadtree cells*), such that each cell obeys the stopping rule while its parent does not. The stopping rule we use is as follows:

> Stop splitting when all edges intersecting the cell σ under consideration are incident to a common vertex.

Note that the stopping rule includes the case were σ is not intersected by any edges. We can prove that this stopping rule leads to a quadtree subdivision with $O(n/\delta^2)$ cells, such that each cell is intersected by at most $2\pi/\delta$ triangles, and the total number of triangle-cell intersections is $O(n/\delta^2)$.

We store the quadtree subdivision defined above as a linear quadtree [8]. To this end, we define an ordering on the leaf cells of the quadtree subdivision. The ordering is based on a space-filling curve defined recursively by the order in which it visits the quadrants of a canonical square. We use the Z-*order space-filling curve* for this, which visits the quadrants in the order bottom left, top left, bottom right, top right, and within each quadrant, the Z-order curve visits its subquadrants recursively in the same order. Since the intersection of every canonical square with this curve is a contiguous section of the curve, this yields a well-defined ordering of the leaf cells of the quadtree subdivision. We call this order the Z-order.

The Z-order curve not only orders the leaf cells of the quadtree subdivision, but it also provides an ordering for any two points in the unit square—namely the Z-order of any two disjoint canonical squares containing the points. (We assume that canonical squares are closed at the bottom and the left side, and open at the top and the right side.) The Z-order of two points can be determined as follows. For a point $p = (p_x, p_y)$ in $[0, 1\rangle^2$, define its Z-index $z(p)$ to be the value in the range $[0, 1\rangle$ obtained by interleaving the bits of the fractional parts of p_x and p_y, starting with the first bit of p_x. The value $z(p)$ is sometimes called the *Morton block index* of p. The Z-order of two points is now the same as the order of their Z-indices [9]. The Z-indices of all points in a canonical square σ form a subinterval $[z_1, z_2\rangle$ of $[0, 1\rangle$, where z_1 is the Z-index of the bottom left corner of σ. Note that any subdivision of the unit square $[0, 1\rangle^2$ into k leaf cells of a quadtree corresponds directly to a subdivision of the unit interval $[0, 1\rangle$ of Z-indices into k subintervals.

A simple way of storing a triangulation in a linear quadtree is now obtained by storing all cell-triangle intersections in a B-tree [3]: each cell-triangle intersection $(\sigma, \triangle)$ of a cell σ corresponding to the Z-index interval $[z_1, z_2\rangle$ is represented by storing triangle $\triangle$ with key z_1. Thus the leaf cells are stored implicitly: each pair of consecutive different keys z_1 and z_2 constitutes the Z-index interval of a quadtree leaf cell.

Building the quadtree Since the quadtree may have height $\Theta(n)$, a natural top-down construction algorithm could take $\Theta(n^2/B)$ I/O's. Below we describe a faster algorithm that computes the leaf cells that result with our stopping rule directly, using local computations instead of a top-down approach.

For any vertex v of the given triangulation $\mathcal{F}$, let $star(v)$ be the *star* of v in $\mathcal{F}$; namely, it is the set of triangles of $\mathcal{F}$ that have v as a vertex. Recall that a canonical square is any square that can be obtained by recursively subdividing the unit square into quadrants. For a set S of triangles inside the unit square, we say that a canonical square of σ is *active* in S if it lies completely inside S and all edges from S that intersect σ are incident to a common vertex, while $mom(\sigma)$ intersects multiple edges of S that are not all incident to a common vertex. Thus the cells of the quadtree subdivision we wish to compute for $\mathcal{F}$ are exactly the active canonical squares in $\mathcal{F}$.

Lemma 2 *Let $\triangle uvw$ be a triangle of $\mathcal{F}$ and σ a canonical square that intersects $\triangle uvw$. Then σ is active in $\mathcal{F}$ if and only if σ is active in $star(u)$, $star(v)$ or $star(w)$.*

We now construct the linear quadtree as follows:

1. Compute an adjacency list for each vertex.
2. Scan the adjacency lists for all vertices: for each vertex u load its adjacency list in memory and compute the active cells of $star(u)$, with for each cell σ the triangles that intersect σ. Output each triangle with the key z_1 of the Z-index interval $[z_1, z_2\rangle$ that corresponds to σ.
3. Sort the triangles by key, removing duplicates.
4. Build a B-tree on the list of triangles with keys.

This algorithm runs in $O(sort(n/\delta^2))$ I/O's. Note that by Lemma 2, local update operations such as inserting a vertex can be done by computing the structure of the quadtree locally in the area of the update, and determining what the changes entail for the data stored on the disk.

Overlaying maps and point location Recall that each triangulation's quadtree, or rather, subdivision of the Z-order curve, is stored on disk as a sorted list of Z-indices with triangles. To overlay the two triangulations, we scan the two quadtrees simultaneously in Z-order, reporting, for any pair of intersecting leaf cells, the intersections between the triangles stored with the cells. The input has size $O(n/\delta^2)$. The output consists of $O(n/\delta)$ intersections since a δ-fat triangulation has density $O(1/\delta)$, as shown by De Berg et al. [5]. Thus map overlay takes only $O(scan(n/\delta^2))$ I/O's.

To locate a point p we compute its Z-index $z(p)$ and search the B-tree for the triangles with the highest keys less than or equal to $z(p)$. For batched point location, we sort the set P of query points by Z-index, and scan the leaves of the B-tree and P in parallel.

3 Our solution for low-density subdivisions

The *density* of a set of line segments in the plane is the smallest number λ such that the following holds: any disk D is intersected by at most λ line segments with length at least the diameter of D. We say that a subdivision $\mathcal{F}$ has density λ if its edge set has density λ.

Theorem 3 *Let $\mathcal{F}$ be a subdivision or a set of non-intersecting line segments of density λ with n edges. Knowing M and B, we can construct, in $O(sort(\lambda n))$ I/O's, a linear compressed quadtree for $\mathcal{F}$ with $O(n)$ cells that each intersect $O(\lambda)$ edges. With this structure we can do the following:*

(i) **Map overlay:** *If we have two subdivisions (or sets of non-intersecting line segments) of density λ with n edges in total, both stored in such a linear compressed quadtree, then we can find all pairs of intersecting edges in $O(scan(\lambda n))$ I/O's.*

(ii) **(Batched) point location:** *for any query point p we can find the face of $\mathcal{F}$ that contains p in $O(\log_B n)$ I/O's, and for any set P of k query points we can find for each point $p \in P$ the face of $\mathcal{F}$ that contains p in $O(scan(\lambda n)+sort(k))$ I/O's.*

Below we explain our data structure. The query algorithms are the same as in the previous section.

The compressed quadtree subdivision for low-density maps Let $\mathcal{G}$ be the set of vertices of the axis-parallel bounding boxes of the edges of $\mathcal{F}$. We construct a quadtree for $\mathcal{F}$ with the following rule:

$$\boxed{\text{Stop splitting when the cell } \sigma \text{ under consideration contains at most one point from } \mathcal{G}.}$$

To be able to bound the number of cells to $O(n)$, we use five-way splits as in a compressed quadtree [10], as follows. Let σ be a canonical square that contains more than one point from $\mathcal{G}$, and let σ' be the smallest canonical square that contains all points of $\sigma \cap \mathcal{G}$. Then σ is split into five regions. The first region is the *donut* $\sigma \setminus \sigma'$. The remaining four regions are the four quadrants of σ'. Note that the first region does not contain any points of $\mathcal{G}$, so it is never subdivided further. We can prove that a quadtree subdivision based on the above stopping rule and five-way splits has $O(n)$ cells, each intersected by at most $O(\lambda)$ edges.

We store the cell-edge intersections of the compressed quadtree subdivision in a list sorted by the Z-order of the cells, indexed by a B-tree. The only difference with the previous section is that we now have to deal with donuts as well as square cells. Recall that a canonical square (a square that can be obtained from the unit square by a recursive partitioning into quadrants) corresponds to an interval on the Z-order curve. For a donut this is not true. However, a donut corresponds to at most two such intervals, because a donut is the set-theoretic difference of two canonical squares. Thus the solution of the previous section can be applied if we represent each donut by two intervals $[z_1, z_2\rangle$ and $[z_3, z_4\rangle$; edges intersecting the first part of the donut are stored with key z_1 and edges intersecting the second part are stored with key z_3. We merge cells that do not intersect any edge with their immediate successors or predecessors in the Z-order. We call the resulting structure—the B-tree on the cell-edge intersections whose keys imply a compressed quadtree subdivision—a *linear compressed quadtree.*

Building the quadtree We construct the leaves of the compressed quadtree, or rather, the corresponding subdivision of the Z-order curve, as follows. We sort $\mathcal{G}$ into Z-order, and scan the sorted points. For each pair of consecutive points, say u and v, we construct their lowest common ancestor $lca(u, v)$ by examining the longest common prefix of the bit strings representing $z(u)$ and $z(v)$. We output the five Z-indices that bound and separate the Z-order intervals of the four children of $lca(u, v)$. To complete the subdivision of the Z-order curve, we sort the output into a list by Z-order, removing duplicates.

We now build a B-tree on the subdivision of the Z-order curve, and distribute the edges of $\mathcal{F}$ to the leaves that intersect them. This is done in $O(sort(\lambda n))$ I/O's in a straightforward top-down manner.

Finally we collect all edge-leaf intersections, ordered by the Z-indices of the leaf cells, and put a new B-tree on top of them. Each cell σ without any intersecting edges is merged with the cells that precede or follow it in the Z-order.

The complete algorithm runs in $O(sort(\lambda n))$ I/O's.

Acknowledgment

We thank Sariel Har-Peled for his extensive contribution.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.

[2] L. Arge, D. E. Vengroff, and J. S. Vitter. External-memory algorithms for processing line segments in geographic information systems. In *Proc. European Sympos. Algorithms*, pages 295–310, 1995.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press / McGraw-Hill, Cambridge, Mass., 2001.

[4] A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, and E. Ramos. Randomized external-memory algorithms for some geometric problems. *Comput. Geom. Theory Appl.*, 11(3):305–337, June 2001.

[5] M. de Berg, M. J. Katz, A. v. Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34:81–97, 2002.

[6] U. Finke and K. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. ACM Symp. Comp. Geom.*, pages 119–126, 1995.

[7] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. IEEE Symp. Found. Comp. Sc.*, pages 285–298, 1999.

[8] I. Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, 1982.

[9] G. R. Hjaltason and H. Samet. Speeding up construction of pmr quadtree-based spatial indexes. *VLDB Journal*, 11:190–137, 2002.

[10] H. Samet. *Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods.* Addison-Wesley, Reading, MA, 1989.

Streaming Algorithms for Line Simplification under the Fréchet Distance

M.A. Abam[*] M.de Berg[*] P. Hachenberger[*] A. Zarei[†]

Abstract

We study the following variant of the well-known line-simplification problem: we are getting a possibly infinite sequence of points $p_0, p_1, p_2, \dots$ defining a polygonal path, and as we receive the points we wish to maintain a simplification of the path seen so far. We study this problem in a streaming setting, where we only have a limited amount of storage so that we cannot store all the points. We analyze the competitive ratio of our algorithm, allowing resource augmentation: we let our algorithm maintain a simplification with $2k$ (internal) points, and compare the error of our simplification to the error of the optimal simplification with k points.

1 Introduction

Suppose we are tracking one, or maybe many, moving objects. Each object is equipped with a device that is continuously transmitting its position. Thus we are receiving a stream of data points that describes the path along which the object moves. The goal is to maintain this path for each object. We are interested in the scenario where we are tracking the objects over a very long period of time, as happens for instance when studying the migratory patterns of animals. In this situation it may be undesirable or even impossible to store the complete stream of data points. Instead we have to maintain an approximation of the input path. This leads us to the following problem: we are receiving a (possibly infinite) stream $p_0, p_1, p_2, \dots$ of points in the plane, and we wish to maintain a simplification (of the part of the path seen so far) that is as close to the original path as possible, while using not more than a given (fixed) amount of available storage.

The problem described above is a streaming version of line simplification, one of the basic problems in GIS. Here one is given a polygonal path $P := p_0, p_1, \dots, p_n$ in the plane, and the goal is to find a path $Q := q_0, q_1, \dots, q_k$ with fewer vertices that approximates P well. In fact, this problem arises whenever we want to perform data reduction on a polygonal shape in the plane, and so it plays a role not only in GIS but also in areas like image processing and computer graphics. Line simplification has been studied extensively both in these application areas as well as in computational geometry.

The line-simplification problem has many variants. For example, we can require the sequence of vertices of Q to be a subsequence of P (with $q_0 = p_0$ and $q_k = p_n$)—this is sometimes called the *restricted version*—or we can allow arbitrary points as vertices. In this paper, as in most other papers, we consider the restricted version, and we limit our discussion to this version from now on; some results on the unrestricted version can be found in [5, 6, 7]. In the restricted version, each link $q_l q_{l+1}$ of the simplification corresponds to a shortcut $p_i p_j$ (with $j > i$) of the original path, and the error of the link is defined as the distance between $p_i p_j$ and the subpath $p_i, \dots, p_j$. To measure the distance between $p_i p_j$ and $p_i, \dots, p_j$ the Hausdorff distance or the Fréchet distance are usually used. Since we concentrate on the latter, the error of the simplification Q is now defined as the maximum Fréchet error of any of its links. Once the error measure has been defined, we can consider two types of optimization problems: the min-k and the min-δ problem. In the min-k problem, one is given the path P and a maximum error δ, and the goal is to find a simplification Q with as few vertices as possible whose error is at most δ. In the min-δ problem, one is given the path P and a maximum number of vertices k, and the goal is to find a simplification with the smallest possible error that uses at most k vertices.

The line-simplification was first studied for the Fréchet distance by Godau [4]. Alt and Godau [2] proposed an algorithm to compute the Fréchet distance between two polygonal paths in quadratic time; combined with the approach of Imai and Iri [8] this can be used to compute an optimal solution to the min-δ or the min-k problem for the Fréchet distance. Since solving the line-simplification problem exactly is costly—the best known algorithm for the Fréchet distance takes quadratic time or more—Agarwal *et al.* [1] consider approximation algorithms. In particular, they consider the min-k problem for both the Hausdorff distance for x-monotone paths (in the plane) and the Fréchet distance for general paths (in

[*]Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands. Email: {mabam,mdberg,phachenb}@win.tue.nl. MAA was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 612.065.307. MdB and PH were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

[†]Computer Engineering Department, Sharif University of Technology and IPM School of Computer Science, Tehran, Iran. Email: zarei@mehr.sharif.edu.

d-dimensional space). They give near-linear time algorithms that compute a simplification whose error is at most δ and whose number of vertices is at most the minimum number of vertices of a simplification of error at most $\delta/2$. However, these algorithms cannot be used in a streaming setting, because the complexity of the produced simplification for an input path of n points can be $\Theta(n)$.

To state our problem more precisely, we first introduce some terminology and definitions. Let $p_0, p_1, \ldots$ be the given stream of input points. We use $P(n)$ to denote the path defined by the points $p_0, p_1, \ldots, p_n$—that is, the path connecting those points in order—and for any two points p, q on the path we use $P(p,q)$ to denote the subpath from p to q. For two vertices p_i, p_j we use $P(i,j)$ as a shorthand for $P(p_i, p_j)$. A segment p_ip_j with $i < j$ is called a *link* or sometimes a *shortcut*. Thus $P(n)$ consists of the links $p_{i-1}p_i$ for $0 < i \leqslant n$. We assume a function *error* is given that assigns a non-negative error to each link p_ip_j. An *ℓ-simplification* of $P(n)$ is a polygonal path $Q := q_0, q_1, \ldots, q_k, q_{k+1}$ where $k \leqslant \ell$ and $q_0 = p_0$ and $q_{k+1} = p_n$, and $q_1, \ldots, q_k$ is a subsequence of $p_1, \ldots, p_{n-1}$. The error of a simplification Q for a given function *error*, denoted *error*(Q), is defined as the maximum error of any of its links. We consider an error function based on the Fréchet distance, as defined next.

The Fréchet distance between two paths A and B, which we denote by $d_F(A,B)$, is defined as follows. Consider a man with a dog on a leash, with the man standing at the start point of A and the dog standing at the start point of B. Imagine that the man walks to the end of A and the dog walks to the end of B. During the walk they can stop every now and then, but they are not allowed to go back along their paths. The Fréchet distance between A and B is the minimum length of the leash needed for this walk, over all possible such walks. See [4] for a formal definition.

Now consider an algorithm $\mathcal{A} := \mathcal{A}(\ell)$ that maintains an ℓ-simplification for the input stream $p_0, p_1, \ldots$, for some given ℓ. Let $Q_{\mathcal{A}}(n)$ denote the simplification that $\mathcal{A}$ produces for the path $P(n)$. Let $Opt(\ell)$ denote an optimal off-line algorithm that produces an ℓ-simplification. Thus $error(Q_{Opt(\ell)}(n))$ is the minimum possible error of any ℓ-simplification of $P(n)$. We define the quality of $\mathcal{A}$ using the *competitive ratio*, as is standard for on-line algorithms. We also allow *resource augmentation*, i.e., we allow $\mathcal{A}$ to use a $2k$-simplification, but we compare the error of this simplification to $Q_{Opt(k)}(n)$. Thus we define the competitive ratio of an algorithm $\mathcal{A}(2k)$ as

$$\text{competitive ratio of } \mathcal{A}(2k) := \max_{n \geqslant 0} \frac{error(Q_{\mathcal{A}(2k)}(n))}{error(Q_{Opt(k)}(n))}.$$

We say that an algorithm is *c-competitive* if its competitive ratio is at most c.

We present and analyze a simple general streaming algorithm for line simplification. Our analysis shows that the algorithm has good competitive ratio under two conditions: the error function that is used is *monotone*—see Section 2 for a definition—and there is an oracle that can approximate the error of any candidate link considered by the algorithm. We then continue to show that the Fréchet error function is monotone for arbitrary paths in the plane and how to implement the error oracles for this setting. Putting everything together leads to the following result. For paths in the plane and the Fréchet error function we can, for any fixed $\varepsilon > 0$, obtain a $(4\sqrt{2} + \varepsilon)$-competitive streaming algorithm that uses $O((k^2/\sqrt{\varepsilon}) \log^2(1/\varepsilon))$ additional storage and processes each input point in $O((k/\sqrt{\varepsilon}) \log^2(1/\varepsilon))$ amortized time.

2 A general simplification algorithm

In this section we describe a general strategy for maintaining an ℓ-simplification of an input stream $p_0, p_1, \ldots$ of points in the plane, and we show that it has a good competitive ratio under two conditions: the error function is *monotone* (as defined below), and we have an *error oracle* at our disposal that computes or approximates the error of a link. We denote the error computed by the oracle for a link p_ip_j by *error*$^*(p_ip_j)$. Later we will prove that the Fréchet error function is monotone, and we will show how to implement the oracle for this setting.

Suppose we have already handled the points $p_0, \ldots, p_n$. (We assume $n > \ell + 1$; otherwise we can simply use all points and have zero error.) Let $Q := q_0, q_1, \ldots, q_\ell, q_{\ell+1}$ be the current simplification. Our algorithm will maintain a priority queue $\mathcal{Q}$ that stores the points q_i with $1 \leqslant i \leqslant \ell$, where the priority of a point is the error (as computed by the oracle) of the link $q_{i-1}q_{i+1}$. In other words, the priority of q_i is (an approximation of) the error that is incurred when q_i is removed from the simplification. Now the next point p_{n+1} is handled as follows:

1. Set $q_{\ell+2} := p_{n+1}$, thus obtaining an $(\ell+1)$-simplification of $P(n+1)$.
2. Compute *error*$^*(q_\ell q_{\ell+2})$ and insert $q_{\ell+1}$ into $\mathcal{Q}$ with this error as priority.
3. Extract the point q_s with minimum priority from $\mathcal{Q}$; remove q_s from the simplification.
4. Update the priorities of q_{s-1} and q_{s+1} in $\mathcal{Q}$.

Next we analyze the competitive ratio of our algorithm. We say that a link p_ip_j *encloses* a link p_lp_m if $i \leqslant l \leqslant m \leqslant j$, and we say that *error* is a *c-monotone error function* for a path $P(n)$ if for any two links p_ip_j and p_lp_m such that p_ip_j encloses p_lp_m we have

$$error(p_lp_m) \leqslant c \cdot error(p_ip_j).$$

In other words, an error function is c-monotone if the error of a link cannot be worse than c times the error of any link that encloses it. Furthermore, we say that the error oracle is an *e-approximate error oracle* if for any link p_ip_j

$$\mathit{error}(p_ip_j) \leqslant \mathit{error}^*(p_ip_j) \leqslant e \cdot \mathit{error}(p_ip_j)$$

Theorem 1 *Suppose that the error function is c-monotone and that we have an e-approximate error oracle at our disposal. Then the algorithm described above with $\ell = 2k$ is ce-competitive with respect to $Opt(k)$. The time the algorithm needs to update the simplification Q upon the arrival of a new point is $O(\log k)$ plus the time spent by the error oracle. Besides the storage needed for the simplification Q, the algorithm uses $O(k)$ storage plus the storage needed by the error oracle.*

Proof. Consider an arbitrary $n \geqslant 0$, and let $Q(n)$ denote the $2k$-simplification produced by our algorithm. Since the error of $Q(n)$ is the maximum error of any of its links, we just need to show that $\mathit{error}(\sigma) \leqslant ce \cdot \mathit{error}(Q_{Opt(k)}(n))$ for any link σ in $Q(n)$. Let $m \leqslant n$ be such that σ appears in the simplification when we receive point p_m. If $m \leqslant 2k+2$, then $\mathit{error}(\sigma) = 0$ and we are done. Otherwise, let $Q(m-1) := q_0, \ldots, q_{2k+1}$ be the $2k$-simplification of $P(m-1)$. Upon the arrival of $p_m = q_{2k+2}$ we insert $q_{2k+1} = p_{m-1}$ into $\mathcal{Q}$. A simple counting argument shows that at least one of the shortcuts $q_{t-1}q_{t+1}$ for $1 \leqslant t \leqslant 2k+1$, let's call it σ', must be enclosed by one of the at most $k+1$ links in $Q_{Opt(k)}(n)$. Since σ is the link with the smallest priority among all links in $\mathcal{Q}$ at that time, its approximated error is smaller than that of σ'. Therefore,

$$\begin{aligned} \mathit{error}(Q_{Opt(k)}(n)) &\geqslant \tfrac{1}{c}\mathit{error}(\sigma') \geqslant \tfrac{1}{c\cdot e}\mathit{error}^*(\sigma') \\ &\geqslant \tfrac{1}{c\cdot e}\mathit{error}^*(\sigma) \geqslant \tfrac{1}{c\cdot e}\mathit{error}(\sigma). \end{aligned}$$

We conclude that our algorithm is ce-competitive with respect to $Opt(k)$. Besides the time and storage needed by the error oracle, the algorithm only needs $O(k)$ space to store the priority queue and $O(\log k)$ for each update of the priority queue. $\square$

3 An algorithm for the Fréchet error function

We now turn our attention to the Fréchet error function. We will show that we can obtain an $O(1)$-competitive algorithm for arbitrary paths in the plane. The first property we need is that the Fréchet error function is monotone. This has in fact already been proven by Agarwal *et al.* [1].

Lemma 2 [1] *The Fréchet error function is 2-monotone on arbitrary paths.*

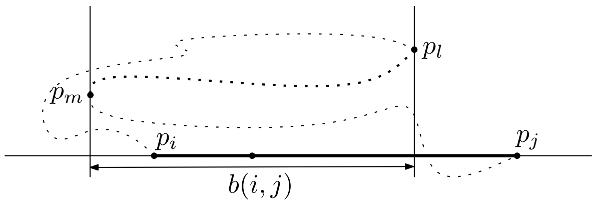


Figure 1: The largest back-path in direction p_ip_j.

Next we turn our attention to the implementation of the error oracle for the Fréchet error function. We use two parameters to approximate $\mathit{error}_F(p_ip_j)$: the width of the points of $P(i,j)$ in the direction p_ip_j and the length of the largest back-path in the direction of p_ip_j.

The *width* of a set of points with respect to a given direction $\overrightarrow{d}$ is the minimum distance of two lines being parallel to $\overrightarrow{d}$ that enclose the point set. Let $w(i,j)$ be the width of the points in subpath $P(i,j)$ with respect to the direction $\overrightarrow{p_ip_j}$. Chan [3] has described a streaming algorithm for maintaining a core-set that can be used to approximate the width of a set in any direction. More precisely, given a data stream $p_0, p_1, \ldots$, he maintains an ε-core-set of size $O((1/\sqrt{\varepsilon})\log^2(1/\varepsilon))$ in $O(1/\sqrt{\varepsilon})$ amortized time per point; with this core-set one can get a $(1+\varepsilon)$-approximation of the width in any direction.

The largest back-path in direction p_ip_j is defined as follows. Assume without loss of generality that p_ip_j is horizontal with p_j to the right of p_i. For two points p_l, p_m on the path $P(i,j)$ with $l < m$ we define $P(l,m)$ to be a *back-path on* $P(i,j)$ if $(p_m)_x < (p_l)_x$. In other words $P(l,m)$ is a back-path if, relative to the direction $\overrightarrow{p_ip_j}$, we go back when we move from p_l to p_m. The *length* of a back-path $P(l,m)$ on $P(i,j)$ is defined to be the length of the projection of p_lp_m onto a line parallel to p_ip_j, which is equal to $(p_l)_x - (p_m)_x$ since we assumed p_ip_j is horizontal. We define $b(i,j)$ to be the maximum length of any back-path on $P(i,j)$. See Figure 1 for an illustration.

Lemma 3 $\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2}) \leqslant \mathit{error}_F(p_ip_j) \leqslant 2\sqrt{2}\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2})$.

In the algorithm as presented in Section 2 we need to maintain (an approximation of) the error of each shortcut q_lq_{l+2} in the current simplification. According to the above lemma, in order to approximate $\mathit{error}_F(p_ip_j)$ it is enough if we can approximate $\max(w(i,j), b(i,j))$.

To approximate the width of the links q_lq_{l+2}, we must maintain a core-set for each link that might be needed at some later time in our simplification. These are the links q_iq_j, with $0 \leqslant i < j-1 < 2k+1$. So we need to maintain a core-set for each of these $O(k^2)$ links. Considering a new point $q_{2k+2} = p_{n+1}$, we must create $O(k)$ new core-sets, one for each of the links $q_i p_{n+1}$, with $0 \leqslant i \leqslant 2k$. We create such core-sets for the links $q_i p_{n+1}$, by copying the core-sets $q_i q_{2k+1}$ and 'inserting' point p_{n+1} to them using Chan's algorithm.

We also need to approximate the maximum length of a back-path on the path from q_l to q_{l+2}. For the moment let's assume that all we need is the maximum length of the back-path with respect to the positive x-direction. Then we maintain for each link $p_i p_j$ of the simplification the following values:

(i) $b(i,j)$, the maximum length of a back-path (w.r.t. the positive x-direction) on $P(i,j)$;
(ii) $xmax(i,j)$, which is defined as the maximum x-coordinate of any point on $P(i,j)$;
(iii) $xmin(i,j)$, which is defined as the minimum x-coordinate of any point on $P(i,j)$.

Now consider a shortcut $q_l q_{l+2}$. Let $q_l = p_i$, $q_{l+1} = p_t$ and $q_{l+2} = p_j$. Then $b(i,j)$, the maximum length of a back-path on $P(q_l, q_{l+2}) = P(i,j)$, is given by

$$\max\ (\ b(i,t),\ b(t,j),\ xmax(i,t) - xmin(t,j)\ ).$$

Adding a point $q_{\ell+2}$ is easy, because we only have to compute the above three values for $q_{\ell+1}q_{\ell+2}$, which is trivial since $q_{\ell+1}$ and $q_{\ell+2}$ are consecutive points on the original path. Removing a point q_s can also be done in $O(1)$ time (let $q_{s-1} = p_i$ and $q_{s+1} = p_j$): above we have shown how to compute $b(i,j)$ from the available information for $q_{s-1}q_s$ and $q_s q_{s+1}$, and computing $xmax(i,j)$ and $xmin(i,j)$ is even easier.

Thus we can maintain the maximum length of a back-path. There is one catch, however: the procedure given above maintains the maximum length of a back-path *with respect to a fixed direction* (the positive x-direction). But in fact we need to know for each $q_i q_{i+2}$ the maximum length of a back-path with respect to the direction $\overrightarrow{q_i q_{i+2}}$. These directions are different for each of the links and, moreover, we do not know them in advance. To overcome this problem we define $2\pi/\alpha$ equally spaced canonical directions, for a suitable $\alpha > 0$, and we maintain, for every link $p_i p_j$, the information described above for each direction. Now suppose we need to know the maximum length of a back-path for $p_i p_j$ with respect to the direction $\overrightarrow{p_i p_j}$. Then we will use $b_{\overrightarrow{d}}(p_i p_j)$, the maximum length of a back-path with respect to $\overrightarrow{d}$ instead, where $\overrightarrow{d}$ is the canonical direction closest to $\overrightarrow{p_i p_j}$ in clockwise order. In general, using $\overrightarrow{d}$ may not give a good approximation of the maximum length of a back-path in direction $\overrightarrow{p_i p_j}$, even when α is small. However, the approximation is only bad when $w(i,j)$ is relatively large, which means that the Fréchet distance can still be approximated well.

Lemma 4 *Let w be the width of $P(i,j)$ in direction $\overrightarrow{p_i p_j}$, let b be the maximum length of a back-path on $P(i,j)$ in direction $\overrightarrow{p_i p_j}$, and let b^* be the maximum length of a back-path on $P(i,j)$ in direction $\overrightarrow{d}$. Then we have: $b^* - \tan(\alpha)\cdot w \leqslant b \leqslant b^* + \tan(\alpha)\cdot(b^* + w)$.*

The final oracle is now defined as follows. Let w^* be the approximation of the width of $P(i,j)$ in direction $\overrightarrow{p_i p_j}$ as given by Chan's ε-core-set method, and let b^* be the maximum length of a back-path on $P(i,j)$ in direction $\overrightarrow{d}$, where $\overrightarrow{d}$ is the canonical direction closest to $\overrightarrow{p_i p_j}$ in clockwise order. We set

$$error^*_F(p_i p_j) \ := \ \sqrt{2}\cdot\max(w^*, b^* + \tan(\alpha)\cdot(b^* + w^*)).$$

Combing Lemma 3 with the observations above, we can prove the following lemma.

Lemma 5 *$error_F(p_i p_j) \leqslant error^*_F(p_i p_j) \leqslant 2\sqrt{2}(1 + \varepsilon)(1 + 4\tan(\alpha)) \cdot error_F(p_i p_j)$*

With ε and α sufficiently small, we get our final result.

Theorem 6 *There is a streaming algorithm that maintains a $2k$-simplification for arbitrary paths under the Fréchet error function and that is $(4\sqrt{2} + \varepsilon)$-competitive with respect to $Opt(k)$. The algorithm uses $O(k^2 \frac{1}{\sqrt{\varepsilon}} \log^2(\frac{1}{\varepsilon}))$ additional storage and each point is processed in $O(k \frac{1}{\sqrt{\varepsilon}} \log^2(\frac{1}{\varepsilon}))$ amortized time.*

References

[1] P.K. Agarwal, S. Har-Peled, N.H. Mustafa and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica* 42:203–219 (2005).

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.* 5:75–91 (1995).

[3] T.M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.* 35:20–35 (2006).

[4] M. Godau. A natural metric for curves: Computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annu. Sympos. Theoret. Asp. Comput. Sci.(STACS)*, pages 127–136, 1991.

[5] M.T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discr. Comput. Geom.* 14:445–462 (1995).

[6] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, and J.S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geom. Appl.* 3:383–415 (1993).

[7] S.L. Hakimi and E.F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graph. Models Image Process.* 53:132–136, 1991.

[8] H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In: G.T. Toussaint (ed.), *Computational Morphology*, North-Holland, pages 71–86, 1988.

Computational Geometry through the Information Lens

Erik Demaine

Esther and Harold E. Edgerton Professor, MIT
Associate Professor of Electrical Engineering and Computer Science
Massachusetts Institute of Technology,
Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge, MA 02139, USA
edemaine@mit.edu

A central issue in computational geometry is the discrepancy between the idealized geometric view of points and lines with infinite precision, and the realistic computational view that everything is represented by (finitely many) bits. The geometric view is inspired by Euclidean geometric constructions from circa 300 BC. The computational view matches the reality of digital computers as we know them today and as set forth by Turing in 1936. This discrepancy is traditionally seen as negative: theoretically simple algorithms with infinite precision become difficult to implement in practice with finite precision. A new body of research views the finite-precision reality to be a feature, not a bug, and analyzes the extent to which it can be exploited to obtain faster algorithms than possible for infinite precision.

The bounded-precision / information-theoretic viewpoint has proved extremely successful in the field of (one-dimensional) data structures, reaching tight upper and lower bounds for many fundamental problems. A simple example, hashing, tells us that searching for an exact copy of a query item in a set of n items requires around $\lg n$ bits of information about the query, regardless of the domain. A more complex example, fusion trees, tells us that we need only b bits of information to search for the one-dimensional nearest neighbor among b numbers each b bits long. The information-theoretic view has led to solutions to long-standing open problems in data structures for both finite- and infinite-precision problems, as well as a better understanding of practical uses of finite precision such as radix sort.

The past year has seen the first exploitation of bounded precision in two-dimensional, nonorthogonal problems. We now have data structures for static planar point location and dynamic convex hulls with sublogarithmic query time, and algorithms for constructing Voronoi diagrams in near-linear time. This work starts an exciting new line of research that is far more challenging than classic one-dimensional problems. Our goal is to elucidate the fundamental ways in which geometric information such as points and lines can be decomposed in algorithmically useful ways, enabling a deeper understanding of the relative difficulty of geometric problems.

Smoothed Analysis of Probabilistic Roadmaps

Siddhartha Chaudhuri* Vladlen Koltun*

Abstract

The probabilistic roadmap algorithm is a leading heuristic for robot motion planning. It is extremely efficient in practice, yet its worst case convergence time is unbounded as a function of the input's combinatorial complexity. We prove a smoothed polynomial upper bound on the number of samples required to produce an accurate probabilistic roadmap, and thus on the running time of the algorithm, in an environment of simplices. This sheds light on its widespread empirical success.

1 Introduction

Smoothed analysis It is well-documented that many geometric algorithms that are extremely efficient in practice have exceedingly poor worst-case performance guarantees. Smoothed analysis [15] addresses this issue by observing that geometric inputs often contain a small amount of random noise, such as with point clouds generated by a laser scanner [12]. It can be argued that small degrees of randomness creep into geometric inputs even if they are created by a human modeler [14]. By this reasoning, finely tuned worst-case examples have a low probability of arising and should not disproportionately skew theoretical measures of algorithm performance.

Smoothed analysis [15] measures the maximum over inputs of the expected running time of the algorithm under slight random perturbations of those inputs. For example, let $A \in \mathbb{R}^{n\times d}$ specify a set of n points in $\mathbb{R}^d$, and let $f_X(A)$, where $f_X : \mathbb{R}^{n\times d} \mapsto \mathbb{R}$, be a measure of the performance of algorithm X on A. Then the smoothed performance of X is

$$\max_{A\in\mathbb{R}^{n\times d}} \mathrm{E}_{R\sim\mathcal{N}} \left[f_X(A + \|A\|R)\right],$$

where $\|A\|$ denotes the Frobenius norm of A and $\mathcal{N} = N(0, \sigma^2 I_{n\times d})$ is a Gaussian distribution in $\mathbb{R}^{n\times d}$ with mean 0 and variance σ^2. The parameter σ controls the magnitude of the random perturbation, and as it varies from 0 to ∞ the smoothed performance measure interpolates between worst-case and average-case performance.

Probabilistic roadmaps The probabilistic roadmap (PRM) algorithm revolutionized robot motion planning [8, 10]. It is a simple heuristic that exhibits rapid performance and has become the standard algorithm in the field [4, 5, 13]. Yet its worst-case running time is *unbounded* as a function of the input's combinatorial complexity. The basic algorithm for constructing a probabilistic roadmap is as follows:

> Sample uniformly at random a set of points, called milestones, from the *configuration space* $\mathcal{C}$ of the robot. Keep only those milestones that lie in the *free configuration space* $\mathcal{C}_{\mathsf{free}}$.[1] Let V be the resulting point set. For every $u, v \in V$, if the straight line segment between u and v lies entirely in $\mathcal{C}_{\mathsf{free}}$, add $\{u, v\}$ to the set of edges E, initially empty. The graph $G = (V, E)$ is the probabilistic roadmap.

Given such a roadmap G, a motion between two points p, q in $\mathcal{C}_{\mathrm{free}}$ can be constructed as follows:

> Find a milestone p' (resp., q') in V that is visible from p (resp., from q). If p' and q' lie in different connected components of G, report that there is no feasible motion between p and q. Otherwise plan the motion using a path in G that connects p' and q'.

The above PRM construction and query algorithms can be efficiently implemented in very general settings. The outstanding issue is what the number of samples should be to guarantee (in expectation) that G accurately represents the connectivity of $\mathcal{C}_{\mathrm{free}}$. Clearly, for the algorithm to be accurate there should be a milestone visible from any point in $\mathcal{C}_{\mathrm{free}}$, and there should be a bijective correspondence between the set of connected components of G and the set of connected components of $\mathcal{C}_{\mathrm{free}}$. Unfortunately, the number of random samples required to guarantee this can be made arbitrarily large even for very simple configuration spaces [5].

A number of theoretical analyses provide bounds for the number of samples under assumptions on the

*Computer Science Department, 353 Serra Mall, Stanford University, Stanford, CA 94305, USA; {sidch,vladlen}@stanford.edu. The first author was supported by a PACCAR Inc Stanford Graduate Fellowship.

[1] A robot's *configuration space* is the set of physical positions it may attain (which may or may not coincide with obstacles), parametrised by its degrees of freedom (so a robot with d degrees of freedom has a d-dimensional configuration space). The robot's *free configuration space* is the subset of these positions which do *not* coincide with obstacles, i.e. are possible in real life. These terms are standard in the motion planning literature.

structure of $\mathcal{C}_{\text{free}}$ such as goodness [2, 9], expansiveness [6], and the existence of high-clearance paths [7]. However, none of these assumptions were justified in terms of realistic motion planning problems. In practice, the number of random samples is chosen ad hoc.

Contributions This paper initiates the use of smoothed analysis to explain the success of PRM. We model the configuration space using a set of n simplices in $\mathbb{R}^d$ whose vertices are subject to Gaussian perturbation with variance σ^2. We prove a smoothed upper bound on the required number of milestones that is polynomial in n and $\frac{1}{\sigma}$. The result extends to all γ-smooth perturbations, see below.

In order to achieve this bound we define a space decomposition called the locally orthogonal decomposition. Previously known decompositions, like the vertical decomposition [3, 11] and the "castles in the air" decomposition [1] turn out to be unsuitable for our purpose. We prove that for the roadmap to accurately represent the free configuration space it is sufficient that a milestone is sampled from every cell of this decomposition (Corollary 2). We then prove a smoothed lower bound on the volume of every decomposition cell (Corollary 5). This leads to the desired bound on the number of milestones (Theorem 6).

Our result is only the first step towards a convincing theoretical justification of PRM. The analysis is quite challenging already for the simple representation of the configuration space using independently perturbed simplices. We suggest extensions to more general configuration space models as future work.

2 Bounding the Number of Milestones

Notation Two objects are ε-close (ε-distant) if the shortest distance between them is at most (at least) ε. $A \oplus B$ denotes the Minkowski sum of A and B. $B_d(r)$ denotes the d-ball of radius r, and $S_{d-1}(r)$ denotes its boundary (the $(d-1)$-sphere of radius r).

The model Let Σ be a fixed, convex, polyhedral bounding box for $\mathcal{C}_{\text{free}}$ in $\mathbb{R}^d$, where d is assumed to be constant. This is the domain from which the milestones are sampled by the PRM algorithm. Let D be the diameter and D_{in} be the inner diameter of Σ (the inner diameter of a region is the diameter of the largest ball contained completely within the region). Let $\mathcal{S}$ be a set of n $(d-1)$-simplices in Σ. These are the $\mathcal{C}$-space obstacles in our model. Thus $\mathcal{C}_{\text{free}} = \Sigma \setminus \bigcup_{s \in \mathcal{S}} s$.

A probability distribution $\mathcal{D}$ on $\mathbb{R}^d$ with density function $\mu(.)$ is said to be γ-smooth, for some $\gamma \in \mathbb{R}$, if

1. $\mu(x) \leq \gamma$ for all $x \in \mathbb{R}^d$, and
2. given any hyperplane H, a point distributed under $\mathcal{D}$ is almost surely not on H.

A symmetric d-variate Gaussian distribution with variance σ^2 (covariance matrix $\sigma^2 I_d$) is $\Theta\left(\frac{1}{\sigma^d}\right)$-smooth. We assume that each vertex of each simplex in $\mathcal{S}$ is independently perturbed according to a γ-smooth distribution within the domain.

We note that these simplices may also be thought of as boundary elements of full-dimensional polyhedral obstacles. Our upper bound on the the number of samples required to build an accurate roadmap applies verbatim, since we will discard those samples which fall in the interior of these polyhedra. However, our analysis is then not completely realistic because our perturbation model destroys the connectivity of these boundaries — an improved model and its analysis form a possible avenue of future work (see the Conclusion section).

The locally orthogonal decomposition The locally orthogonal decomposition $\boxtimes(\mathcal{S})$ of $\mathcal{S}$ is the arrangement of the following two collections of hyperplanes:

- $\text{Aff}(s)$ for each $s \in \mathcal{S}$.
- The hyperplane orthogonal to s that is spanned by f (i.e. contains f, since f is of lower dimension), for each $s \in \mathcal{S}$ and each facet f of s.

Hyperplanes of the second type are called *walls.* A facet of $\boxtimes(\mathcal{S})$ is *bound* if it is contained in some $s \in \mathcal{S}$, otherwise it is *free.* In the following, the decomposition is assumed to be restricted to Σ. The second property of γ-smooth distributions ensures that under our perturbation model, $\boxtimes(\mathcal{S})$ is almost surely in general position. We readily obtain the following lemma and its obvious corollary.

Lemma 1 *Let c_1 and c_2 be two cells of $\boxtimes(\mathcal{S})$ that are incident at a free facet. Then for any $p_1 \in c_1$ and $p_2 \in c_2$, the line segment between p_1 and p_2 is disjoint from all $s \in \mathcal{S}$.*

Corollary 2 *If a milestone is placed in each cell of $\boxtimes(\mathcal{S})$ then any two points that can be connected by a path in $\mathcal{C}_{\text{free}}$ can also be connected by a piecewise linear path whose only internal vertices are milestones.*

Volume bound Corollary 2 implies that it suffices to place a milestone in every cell of $\boxtimes(\mathcal{S})$. To show that this can be accomplished with a polynomial number of samples we prove a high-probability lower bound on the volume of each cell of $\boxtimes(\mathcal{S})$. This is achieved with the help of the following simple lemma.

Lemma 3 *Let $\mathcal{A}(\mathcal{H})$ be the arrangement induced by a set of hyperplanes $\mathcal{H}$. If every vertex v of $\mathcal{A}(\mathcal{H})$*

is ε-distant from every hyperplane $H \in \mathcal{H}$ for which $v \notin H$, then the volume (k-dimensional measure) of any k-face of the arrangement is at least $\varepsilon^k/k!$, for $1 \leq k \leq d$.

Lemma 3 implies that volume bounds can be proved through vertex-hyperplane separation bounds. Accordingly, Section 3 is devoted to proving the following theorem:

Theorem 4 *Consider a vertex v and a hyperplane H of $\boxtimes(\mathcal{S})$ such that $v \notin H$, and let $\Delta := \min\{1, D_{\text{in}}\}$. Given $\varepsilon \in [0, \Delta)$, v is ε-close to H with probability at most*

$$O\left(\varepsilon^{1-\alpha} \max\{\gamma, \gamma^{d^2}\}\right)$$

for any $\alpha > 0$.

Note that all terms involving only the constants d and D are subsumed into the $O(.)$ notation. The number of hyperplanes in $\boxtimes(\mathcal{S})$ is $O(n)$ and the number of vertices of $\boxtimes(\mathcal{S})$ is $O(n^d)$. A union bound and an application of Lemma 3 thus yield the following corollary to Theorem 4.

Corollary 5 *Each cell of $\boxtimes(\mathcal{S})$ has volume at least ε with probability at least $1 - \omega$ if*

$$\varepsilon \;\leq\; \min\left\{K\, \omega^{\frac{d}{1-\alpha}}\; n^{-\frac{d(d+1)}{1-\alpha}} \left(\max\{\gamma, \gamma^{d^2}\}\right)^{-\frac{d}{1-\alpha}}, \frac{\Delta^d}{d!}\right\}$$

for any $\alpha > 0$ and an appropriate constant K.

If each cell of $\boxtimes(\mathcal{S})$ has volume at least ε, standard probability theory implies that the expected number of samples sufficient for placing a milestone in every cell is $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$. Applying Corollary 5, we conclude that with high probability, a set of $\text{Poly}(n, \gamma)$ samples from Σ is expected to place a milestone in every cell of $\boxtimes(\mathcal{S})$. This yields our main theorem, which we state in the special case of Gaussian perturbations.

Theorem 6 *Let a free configuration space be defined by n $(d-1)$-simplices in $\mathbb{R}^d$ within a fixed domain. If independent Gaussian perturbations of variance σ^2 are applied to the simplex vertices then the expected number of uniformly chosen random samples required to construct an accurate probabilistic roadmap is polynomial in n and $\frac{1}{\sigma}$.*

3 Distance Bounds

This section forms the technical bulk of the analysis and is devoted to proving Theorem 4. The one-dimensional case admits a simple proof, so we assume $d \geq 2$ in the balance of this paper. The hyperplane H can be of three types, which we analyse separately:

1. The affine span of $s \in \mathcal{S}$.
2. A wall spanned by a facet of $s \in \mathcal{S}$.
3. A hyperplane defining the boundary of Σ.

3.1 Affine Spans of Simplices

Theorem 7 *Consider a fixed point p in $\mathbb{R}^d$. Given $0 \leq k < d$, let $k+1$ points $U = \{u_1, u_2, \ldots, u_{k+1}\}$ be distributed independently and γ-smoothly in Σ. The probability that the affine span of U is ε-close to p is at most*

$$K\varepsilon^{d-k}\gamma^{k+1}$$

for $\varepsilon \geq 0$ and a constant K depending on k, d and D.

Proof. (Sketch) For $k = 0$ the result is trivial. Assume $1 \leq k \leq \frac{d}{2}$. We will integrate over all k-flats formed by $(k+1)$-tuples of points. For a given u_1, the k-subspace $F - u_1$ of $\mathbb{R}^d$ can be represented as the span of k orthogonal unit vectors $v_1, v_2, \ldots, v_k$. We define a particular onto mapping ϕ between k-tuples of points drawn from S_{d-k} and orthonormal bases for k-flats in $\mathbb{R}^d$ that ensures that the localization of a k-tuple to a differential element ensures a corresponding spatial localization of the image k-flat. With this localization, we can assume that the k-flat F has the same orientation as a fixed subspace F^0, and it can be entirely parametrized by a single point on it, say u_1. The point p is ε-close to the the k-flat iff u_1 lies in $(p + F^0) \oplus B_{d-k}(\varepsilon)$ — the probability of this happening is at most γ times the volume of the latter region. The result follows by integration over k-tuples of points from S_{d-k}. We omit the mathematical details of the mapping ϕ and of the integration. The case $k > \frac{d}{2}$ is handled by considering the orthogonal complement of $F - u_1$. □

The following corollary is immediate.

Corollary 8 *For nonnegative integers k, k' that satisfy $k + k' < d$, consider an arbitrarily distributed k'-flat F in $\mathbb{R}^d$, as well as a set $U = \{u_1, \ldots, u_{k+1}\}$ of γ-smoothly distributed points in Σ, independent of F and of each other. The shortest distance between F and the affine span of U is at most ε with probability at most*

$$K\varepsilon^{d-k-k'}\gamma^{k+1}$$

for $\varepsilon \geq 0$ and a constant K depending on k, d and D.

From Theorem 7 we see that a hyperplane-vertex pair of $\boxtimes(\mathcal{S})$, in which the hyperplane is the affine span of a simplex s, and the vertex v is defined entirely by hyperplanes not associated with s, is ε-close with probability at most polynomial in ε and γ. Specifically, the bound is $K\varepsilon\gamma^d$ for a constant K depending on d and D. The "local orthogonality" of $\boxtimes(\mathcal{S})$ allows us to extend the use of Corollary 8 to the case when the vertex is formed by the intersection of one or more walls supporting s with hyperplanes not associated with s.

3.2 Walls Supporting Simplices

When the hyperplane is a wall spanned by a simplex facet, the analysis is trickier. We divide our work into three cases based on the interdependence of the wall and the vertex. These cases may be summarised as:

1. The wall and the vertex are entirely independent.
2. The wall and the vertex depend on the same simplex but the vertex does not lie in the affine span of that simplex.
3. The wall and the vertex depend on the same simplex and the vertex lies in the affine span of that simplex.

The mathematical details are too involved for this abstract and the reader is encouraged to refer to the full version. We will merely comment that the techniques used are similar to those used to prove Theorem 7, and involve, in the last two cases, a study of the angle between two faces of a simplex, which is where we obtain the mysterious constant α (from the inequality $\log x \leq K_\alpha x^\alpha$, for any $x > 1$, $\alpha > 0$ and a constant K_α depending only on α).

3.3 The Boundary of the Domain

Again, the analysis of this final case that deals with hyperplanes constituting the boundary of Σ is omitted due to space limitations. In brief, we use results from Sections 3.1 and 3.2 to show that every vertex of $\boxtimes(\mathcal{S})$ (other than those of the bounding box) follows a smooth distribution and thus prove that the probability of the vertex being ε-close to a boundary hyperplane is at most $K\varepsilon \max\{\gamma, \gamma^{d^2}\}$.

This concludes the proof of Theorem 4.

Acknowledgements

We are grateful to Dan Halperin for crucial early discussions and to Jean-Claude Latombe for his support throughout this project.

References

[1] B. Aronov and M. Sharir. Castles in the air revisited. *Disc. and Comp. Geom.*, 12:119–150, 1994.

[2] J. Barraquand, L. E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Int. J. of Robotics Research*, 16:759–774, 1997.

[3] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. *Th. Comp. Sci.*, 84:77–105, 1991.

[4] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In *Proc. 5th W. on Alg. Founds. of Robotics*, pages 43–59, 2002.

[5] D. Hsu, J.-C. Latombe, and H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *Int. J. of Robotics Research*, 25(7):627-643, 2006.

[6] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. of Comp. Geom. and Apps.*, 9:495–512, 1999.

[7] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Trans. on Robotics and Automation*, 14:166–171, 1998.

[8] L. E. Kavraki and J.-C. Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and A. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 33–53, 1998.

[9] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *Proc. 27th ACM Symp. on Th. of Computing*, pages 353–362, 1995.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 11:566–580, 1996.

[11] V. Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. of the ACM*, 51:699–730, 2004.

[12] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. E. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital Michelangelo project: 3D scanning of large statues. In *SIGGRAPH*, pages 131–144, 2000.

[13] G. Song, S. L. Thomas, and N. M. Amato. A general framework for PRM motion planning. In *Proc. Int. Conf. on Robotics and Automation*, pages 4445–4450, 2003.

[14] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms. In *Proc. of the Int. Congress of Mathematicians*, 2002.

[15] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. of the ACM*, 51:385–463, 2004.

Polygon Exploration with Discrete Vision

Sándor P. Fekete[*] Christiane Schmidt[*] [†]

Abstract

With the advent of autonomous robots with two- and three-dimensional scanning capabilities, classical visibility-based exploration methods from computational geometry have gained in practical importance. However, real-life 3D laser scanning of useful accuracy does not allow the robot to scan continuously while in motion; instead, it has to stop each time it surveys its environment. This requirement was studied by Fekete, Klein, and Nüchter for the subproblem of looking around a corner, but until now has not been considered for whole polygonal regions.

We give the first comprehensive algorithmic study for this important algorithmic problem that combines stationary art gallery-type aspects with watchman-type issues in an online scenario. We show that there is a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio in an orthogonal polygon with holes; we also demonstrate that even for orthoconvex polygons, a competitive strategy can only be achieved for limited aspect ratio A, i.e., for a given lower bound on the size of an edge. Our main result is an $O(\log A)$-competitive strategy for simple rectilinear polygons, which is best possible up to constants.

1 Introduction

Visibility Problems: Old and New. The study of geometric problems that are based on visibility is a well-established field within computational geometry.

In recent years, the development of real-world autonomous robots has progressed to the point where actual visibility-based guarding, searching, and exploring become very serious practical challenges, offering new perspectives for the application of algorithmic solutions. However, some of the technical constraints that are present in real life have been ignored in theory; taking them into account gives rise to new algorithmic challenges, necessitating further research on the theoretical side, and also triggering closer interaction between theory and practice.

One technical novelty that has lead to new possibilities and demands is the development of high-resolution 3D laser scanners that are now being used in robotics. By merging several 3D scans, the robot Kurt3D builds a virtual 3D environment that allows it to navigate, avoid obstacles, and detect objects [8]; this makes visibility problems quite practical, as actually using good trajectories is now possible and desirable. However, while human mobile guards are generally assumed to have full vision at all times, Kurt3D has to stop each time it scans the environment, taking in the order of several seconds for doing so; the typical travel time between scans is in the same order of magnitude, making it necessary to balance the number of scans with the length of travel, and requiring a combination of aspects of stationary art gallery problems with the dynamic challenge of finding a short tour.

In this paper, we give the first comprehensive algorithmic study of visibility-based online exploration in the presence of scan cost, i.e., discrete vision, by considering an unknown polygonal environment, i.e., we consider the resulting *Online Watchman Problem with Discrete Vision* (OWPDV). This is not only interesting and novel in theory, it is also an important step in making algorithmic methods from computational geometry more useful in practice. We focus on the case of rectilinear polygons, which is particularly relevant for practical applications, as it includes almost all real-life buildings.

Classical Related Work. Using a fixed set of positions for guarding a known polygonal region is the classical *art gallery problem* [9]. Note that Schuchardt and Hecker [10] showed that finding a minimum cardinality set of guards is NP-hard, even for a simple rectilinear region; quite easily, this also implies that the offline version of our problem (minimum watchman problem with discrete vision) is also NP-hard, even in simple rectilinear polygons.

Finding a short tour along which one mobile guard can see a given region in its entirety is the *watchman problem*; see Mitchell [7] for a survey. For simple rectilinear polygons and distance traveled being measured according to the Manhattan metric, Deng et al. [2] gave an online algorithm for finding an optimum watchman route (i.e. $c = 1$); note that our approach for the problem with discrete vision is partly based on this GREEDY-ONLINE algorithm, but needs considerable additional work.

Searching with Discrete Penalties. In the presence of a cost for each scan, any optimal tour consists of a polygonal path, with the total cost being a lin-

[*]Abteilung für Mathematische Optimierung, TU Braunschweig, D-38106 Braunschweig, Germany, Email: {s.fekete,c.schmidt}@tu-bs.de, http://www.math.tu-bs.de/mo

[†]Supported by DFG Focus Program “Organic Computing” (SPP 1183) project “AutoNomos” (Fe 407/11-1).

ear combination of the path length and the number of vertices in the path.

Somewhat surprisingly, scan cost (however small it may be) causes a crucial difference to the well-studied case without scan cost, even in the limit of infinitesimally small scan times: quite recently, Fekete et al. [3] have established an asymptotically optimal competitive ratio of 2 for the problem of looking around a corner with scan cost, as opposed to the optimal ratio of 1.2121... without scan cost [6].

Our Results. Our mathematical results are as follows, cf. [4]:

- We show that there is a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio in a rectilinear polygon with holes; this is markedly higher than in the case of continuous vision, where the best lower bound is $\Omega(\log n)$. Note that this lower bound is purely combinatorial, as it only requires coordinates that are strongly polynomial (even linear) in n.
- We demonstrate that even in extremely simple cases, a competitive strategy is only possible if maximum and minimum edge length in the polygon are bounded, i.e., for limited resolution of the scanning device; more precisely, we give an $\Omega(\log A)$ lower bound on the competitive ratio for the case of orthoconvex polygons that depends logarithmically on the aspect ratio A of the region that is to be searched; if the input size of coordinates is not taken into account, we get an $\Omega(n)$ lower bound on the competitive factor.
- For the natural special case of simple rectilinear polygons (which includes almost all real-life buildings), we provide a matching competitive strategy with performance $O(\log A)$.

2 Polygons with Holes

Theorem 1 *Let P be a polygonal region with $n+2$ holes and $O(n)$ edges whose lengths are multiples of 1/10 not exceeding $O(n)$. Then no deterministic strategy can achieve a competitive ratio better than $\Omega(\sqrt{n})$, even if P is rectilinear.*

Sketch. We construct P as a polygon and add a number of obstacles as holes to its interior; see Figure 1 for the overall layout. In-between these holes, we add small modifications whose exact shape depends on the tour the robot chooses for exploration. They are bounded by the green rectangles in Figure 1.

3 Why the Aspect Ratio Matters

We state that an $O(\log A)$-competitive strategy for a simple orthogonal polygon with aspect ratio A is best possible, cf. [4].

Theorem 2 *Let P be an orthoconvex polygon region with n edges and aspect ratio A. Then no deterministic strategy can achieve a competitive ratio better than $\Omega(\log A)$.*

4 Simple Rectilinear Polygons

In the following we will develop our strategy SCANSEARCH for simple rectilinear polygons. We start by reviewing the strategy GREEDY-ONLINE for the online watchman problem in simple rectilinear polygons by Deng, Kameda, and Papadimitriou [2] that is optimal for continuous vision.

We will deal with a limited aspect ratio by assuming a minimum edge length of a; for simplicity, we assume that the cost of a scan is equal to the time a robot needs for traveling a distance of 1.

GREEDY-ONLINE. A central idea of polygon exploration is the use of extensions (see [2].) The basic idea of the GREEDY-ONLINE algorithm is to identify the clockwise bound of the currently visible boundary; this is followed by considering a necessary extension that is defined either by the corner incident with this bound, or by a sight-blocking corner. This is based on the proposition of Chin and Ntafos [1] that there always exists a non-crossing shortest path, i.e., a path that visits the critical extensions in the same circular order as the edges on the boundary that induce them. This property is what we need to establish for the case of discrete scans.

A definition of a *visibility path* (a path of a robot with discrete vision along which the same area is visible as it would be for a robot with continouos vision) and modifications for discrete vision to the results by Chin and Ntafos allow us to reformulate their Lemma 3:

Lemma 3 *Any optimum watchman route of a robot with discrete vision in P will have to visit the essential edges in the order in which they appear on the boundary of P', the new polygon obtained by removing the "non-essential" portions of the polygon.*

Developing a Competitive Strategy for a Robot with Discrete Vision. Just like in the

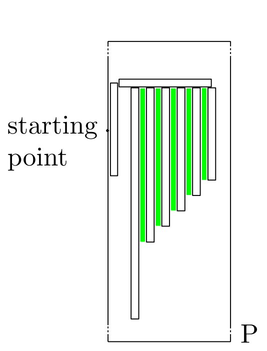


Figure 1: An overview of a construction for $n = 6$.

GREEDY-ONLINE strategy by Deng et al. [2], we start with identifying the next extension, which is either defined by f, the bound of the contiguous visible part of the boundary, or by b, a sight-blocking corner; then the boundary is in clockwise order completely visible up to the extension. Now we merely know that the identified extension needs to be visited; with discrete vision, the optimum does not necessarily need to perform a scan on the extension, instead it may also run beyond it. We distinguish two cases (depending on the distance to the next extension): If we search for visibility on the path to the extension, we refer to the *interval case*, as we only have an unknown interval on one (the counterclockwise) side. If we run beyond the extension, we call this the *extension case*: beyond the extension the clockwise side is also unknown. For simplicity, we compare to an optimal tour in the Manhattan metric.

Even situations that are trivial for a robot with continuous vision may lead to serious difficulties in the case of discrete vision. This leads to the definition of *non-visible regions* (NVRs), as illustrated in Figure 2: without entering the gray area a watchman with continuous vision is able to see the bold sides completely. A robot with discrete vision is only able to see these bold parts of the boundary if he chooses a scan point under the northernmost part of the boundary. Such an area where not (yet) all sides which would be completely visible with continuous vision (the bold sides) are visible for a robot with discrete vision is called a *non-visible region* (NVR).

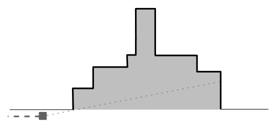


Figure 2: If the dark gray point represent the scan position, a robot with discrete vision cannot see the entire bold sides, resulting in a non-visible region (NVR), shown in gray; an NVR is dealt with by performing a binary search.

Now we assume that without loss of generality, the known parts of the boundary run north-south and east-west, and the extension runs north-south. Thus, we distinguish cases depending on whether we run to or over an extension, and, furthermore, whether we reach the extension on an axis-parallel path without a change of direction.

In case we move beyond an extension, two subcases may occur: either we are able to cover the total planned length, or a boundary keeps us from doing so. Our strategy differs in case of a shortened travel distance, depending on whether the boundary is closed to the south of our path, or not. If we may cover the total distance, we draw an imaginary line parallel to the extension. Then we observe whether the entire boundary on the opposite side of this line is visible. If this is the case we say that the line creation is *positive*. A positive line creation implies that between the extension and the imaginary line there is an essential extension (which may be the extension or the line itself). Otherwise we refer to it as a *negative* line creation.

Both in the interval case and the extension case, our strategy may force the robot to pass some non-visible regions, which we are going to discover with a binary search strategy: we use binary search over a maximal distance as an upper bound (we can show that the robot needs at most k searches ($2k$ if we have NVRs on both sides) if the optimum uses k scans). This yields a reference point for computing the cost of the optimum to determine an upper bound for the competitive ratio.

In some cases our robot needs to take a turn, but we do not know where the optimum turns. Therefore, we consider the maximum possible corridor for turning, move in its center and make adjustments to the new center whenever a width reduction appears. As soon as it is again required to turn, we adjust to the best possible position.

The strategy SCANSEARCH. The full details of the resulting strategy are quite involved, cf. [4]. For an example of our strategy, see Figure 3, with $a = 0.5$. Because of limited space, we just note the following.

Theorem 4 *A simple rectilinear polygon allows an $O(\log A)$-competitive strategy.*

5 Conclusions

We have considered the online problem of exploring a polygon with a robot that has discrete vision. In case of a rectilinear multi-connected polygon we have seen that no strategy may have a constant competitive ratio; even for orthoconvex polygons, it has turned out that any bound on the competitive factor must involve the aspect ratio. Finally, we have developed a competitive strategy for simple rectilinear polygons. For this purpose it was important that we were able to order the extensions along the optimal route of a robot without continuous vision; this enabled us to compare the cost of the optimum with the cost of a robot that uses our strategy.

Another question is whether there exists a competitive strategy in case of a more general class of regions: simple polygons. In this context we face the difficulty that we do not know where the extensions lie. Thus, we are not able to give an a-priori lower bound on the length of the optimum; this is a serious obstacle to adapting the step length to the one of the optimum; extending the highly complex method of Hoffmann et al. [5] for continuous vision to the case of discrete vision is an intriguing and challenging problem.

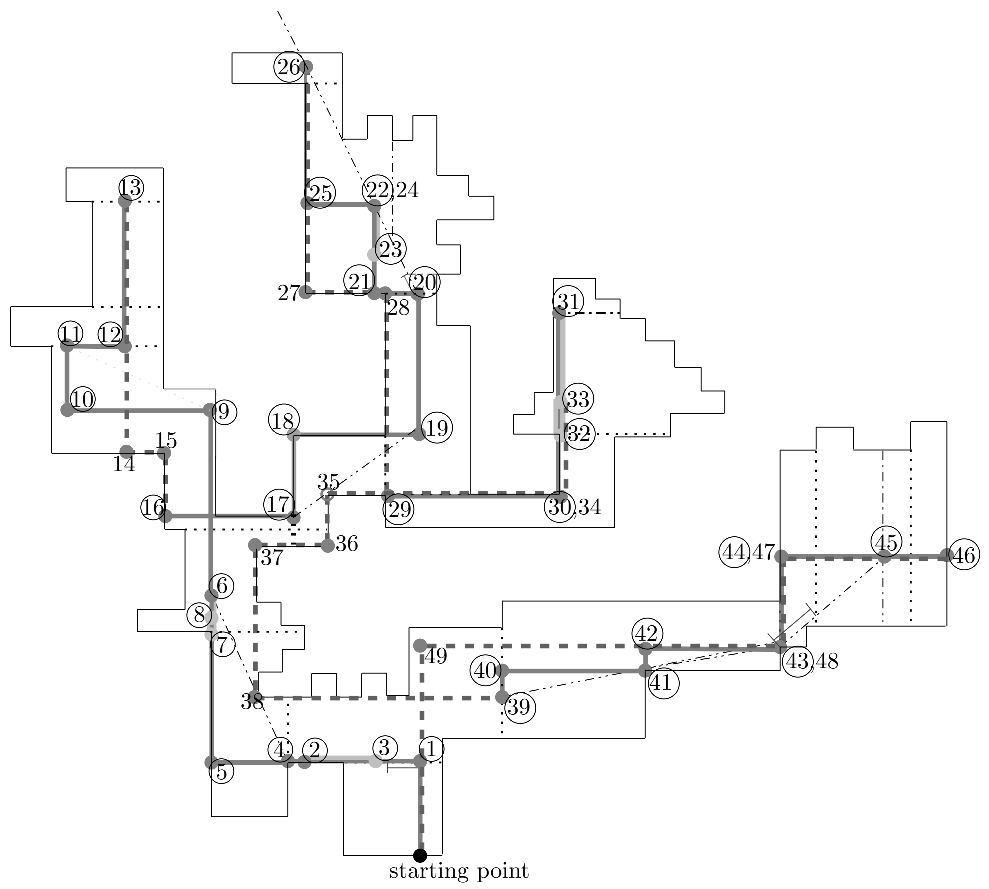


Figure 3: An example for strategy SCANSEARCH. The path of the robot is plotted in dark gray, we use light gray for binary searches, and dashed dark gray for parts of the path where it improves clarity. Extensions are dotted in black, some straight connections are dash dot dotted and some line creations are dash dotted. (For clarity, some lines are slightly offset from their actual position.) Numbered points correspond to turns in the tour, scan points are circled; uncircled turn points arise when navigating back from fully explored terrain. The first extension may be achieved on a straight, axis-parallel line. As we face the interval case and no non-visible regions appear on the counterclockwise side up to E, the robot moves directly to E and takes a scan.

Finally, it is interesting to consider the offline problem for various classes of polygons. As stated in the introduction, even the case of simple rectilinear polygons is NP-hard; developing reasonable approximation methods and heuristics would be both interesting in theory as well as useful in practice.

References

[1] W.-P. Chin and S. Ntafos. Optimum watchman routes. *Proceedings on the Second Annual ACM Symposium on Computational Geometry*, 28(1):39–44, 1988.

[2] X. Deng, T. Kameda, and C. H. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *Journal of the ACM*, 45(2):215–245, 1998.

[3] S. P. Fekete, R. Klein, and A. Nüchter. Online searching with an autonomous robot. *Computational Geometry: Theory and Applications*, 34:102–115, 2006.

[4] S. P. Fekete and C. Schmidt. Polygon exploration with discrete vision, 2006. Submitted for publication.

[5] F.Hoffmann, C.Icking, R.Klein, and K.Kriegel. The polygon exploration problem. *SIAM J. Comp.*, 31:577–600, 2001.

[6] C. Icking, R. Klein, and L. Ma. An optimal competitive strategy for looking around a corner. Technical Report 167, Department of Computer Science, FernUniversität Hagen, Germany, 1994.

[7] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, pages 633–702. Elsevier Science, 2000.

[8] A. Nüchter, H. Surmann, and J. Hertzberg. Automatic classification of objects in 3D laser range scans. In *Proc. 8th Conf. Intelligent Autonomous Systems*, pages 963–970, March 2004.

[9] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Internat. Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[10] D. Schuchardt and H.-D. Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41:261–267, 1995.

Leaving an Unknown Maze with One-Way Roads

Bernd Brüggemann* Tom Kamphans* Elmar Langetepe*

Abstract

We consider the problem of escaping from an unknown polygonal maze under limited resources. The maze may have passages that can be traversed in only one direction. It is well-known that in a setting without 'one-way roads', the Pledge algorithm always finds a path out of an unknown maze provided that such a path exists. We extend the Pledge algorithm for our type of environments and show the correctness of our solutions.

Apart from the maze-leaving application, we introduce a new type of scenes that combines the advantages of both polygonal scenes (i.e., modelling the geometric shape of the environment) and directed graphs (i.e., modelling the connectivity of several parts in the environment). This type might be interesting to consider in other motion-planning tasks such as exploration and search.

Keywords: Online algorithms, motion planning, autonomous robots, Pledge algorithm, one-way roads.

1 Introduction

Imagine you want to leave the old town of a large city. The city is surrounded by some ring roads. As soon as you reach this ring, there are traffic signs that lead you to your destination, but you have no clue how to get to the ring roads. To make things worse, there are many one-way roads in the old town [8].

Usually, we model environments like this by *polygonal scenes*; see, for example, [12, 14, 3]. If we have limited ressources and, particularly, cannot build a map of the environment, the task of leaving an unknown maze can be solved using the well-known Pledge algorithm, see Algorithm 1.[1] The Pledge algorithm assumes that the searcher is able to recognize and follow a wall in a specified direction while counting the *turning angles* (w.l.o.g. we assume that the searcher uses the *left-hand rule*; that is, the searcher keeps the obstacle boundary on its left side). We assume that the searcher has no vision and knows—somehow or other—when it leaved the maze.

It was shown by Abelson and diSessa [1] and Hemmerling [7] that a searcher will escape from a polygonal maze using the Pledge algorithm, provided that there is such a solution, and provided that the agent is error free. For sufficient conditions on the searcher's errors to ensure a successful application of the Pledge algorithm see [9, 8].

Algorithm 1: Pledge [1]

```
φ := 0.
REPEAT
    REPEAT
        Move in direction φ in the free space.
    UNTIL Searcher hits an obstacle.
    REPEAT
        Follow the wall using the left-hand rule.
        Count the overall turning angle in φ.
    UNTIL Angle counter φ is equal to 0.
UNTIL Searcher is outside the maze.
```

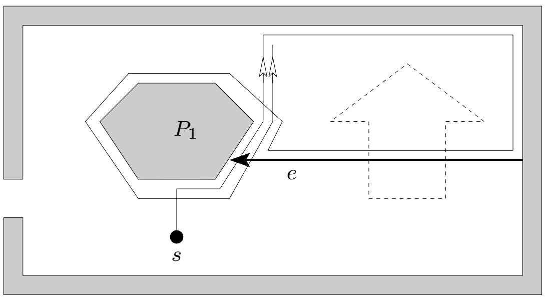


Figure 1: Applying the Pledge algorithm to environments with one-way roads does not work.

We consider the case that there are one-way roads in the surrounding. We can model this problem by adding directed edges between obstacle boundaries. The searcher is allowed to cross these edges only from the left to the right—seen in the direction of the edge—but never the opposite way. It is easy to see that we cannot simply apply the Pledge algorithm considering one-way roads as obstacle edges if we encounter them from the wrong side and otherwise pass them. Figure 1 shows an example with one one-way road e. A searcher starting in s hits the obstacle P_1, passes e, and leaves P_1. Following the second obstacle, the searcher meets the exit side of e. Thus, it follows e and circles P_1 again. The angle counter gets zero in the same vertex of P_1 as in the first visit and the searcher is trapped in an endless loop. Therefore, the

*University of Bonn, Institute of Computer Science I, 53117 Bonn, Germany.

[1]See also [1, 7, 10]. For an implementation of the Pledge algorithm see [6].

simple consideration of one-way roads does not work and we have to use a more sophisticated strategy.

2 Preliminaries

We are given a scene, $\mathcal{M} = (\mathcal{P}, \mathcal{E})$, where $\mathcal{P}$ is a set of simple polygons and $\mathcal{E}$ is a set of *directed* edges, whose start- and endpoints lie on the boundary of a polygon. We call the left side of an edge $e \in \mathcal{E}$ the *entrance* of the one-way road marked by e, the right side the *exit*. The searcher is allowed to pass e only from the entrance side; passing e from the exit side is forbidden. We assume that the searcher perceives exits as walls. Nevertheless, we assume that the searcher does not have sufficient memory to store a map of the whole scene. We allow space only in $|\mathcal{E}|$, not in $|\mathcal{P}|$.

We consider two different models: First, the searcher is not able to distinguish entrances; that is, when meeting an entrance, the searcher is not able to determine whether this entrance is met for the first time or has been met before. In the second setting we assume that the searcher is able to distinguish entrances. Either every entrance has a unique identifier that the searcher can read or the searcher is able to mark a discovered entrance.

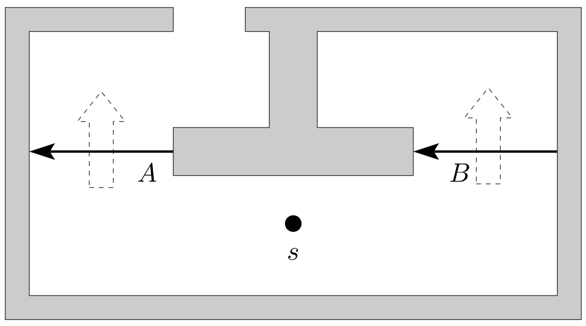


Figure 2: An unfair maze: The searcher cannot tell whether A or B leads to the exit, and it is trapped if it chooses to pass B.

Further, we assume that for every point in the free space, $\mathcal{C}_{\text{free}} := \mathbb{R}^2 \setminus \bigcup_{P \in \mathcal{P}} \overset{\circ}{P}$, there exists a path to an exit; we call such an environment a *fair* maze. In an unfair maze, the searcher may get stuck. See, for example, Figure 2: Starting in s, the searcher cannot determine whether the exit is behind one-way road A or B. Once the searcher passed the wrong road, B, it is trapped!

Definition 1 *Given a scene, $\mathcal{M} = (\mathcal{P}, \mathcal{E})$, we can consider every edge in $\mathcal{E}$ as a wall. Now, the free space, $\mathcal{C}_{\text{free}}$, divides into several path-connected components.*[2] *We call these components the* regions *of $\mathcal{M}$.*

[2]A set $S \subseteq \mathbb{R}^2$ is path connected, if for every $a, b \in S$ there is a path from a to b that is completely inside S.

3 Leaving a Maze with One-Way Roads

The difference between usual polygonal scenes and our type of environments is that we have edges that mark the entrance to a one-way road. Now, when the searcher reaches an entrance, it has the choice to enter the one-way road or to consider the entrance edge as a wall and follow the edge.

It is easy to see that any strategy with periodic choices (e.g., 'enter every second one-way road' or 'enter a one-way road every second time that its entrance is met') fails: For such a strategy, we can construct a maze where the given periodic choice ends up in an endless loop. However, any fair maze is solvable:

Lemma 1 *For every fair maze, $\mathcal{M} = (\mathcal{P}, \mathcal{E})$, there is a function $\beta : \{1, \ldots, |\mathcal{E}|\} \longrightarrow \{\text{enter}, \text{bypass}\}$ such that a searcher using the Pledge algorithm can leave $\mathcal{M}$ if it enters a one-way road, e_i, iff $\beta(i) = \text{enter}$.*

Proof. We start with $\beta(i) = \text{bypass}\ \forall i$. In every maze there exists a region, R_1, from which the searcher using the Pledge algorithm can escape without crossing a one-way road. Remark that R_1 is the only unbounded region in $\mathcal{M}$. Now, we remove every obstacle and one-way road in R_1 and proceed recursively until we removed every obstacle. For a given start point, there is a sequence $R_k, \ldots, R_1$ of regions that the searcher has to pass to move from s to R_1. Now, we define $\beta(i) := \text{enter}$ for every e_i that leads from a region R_j to R_{j-1}, $1 < j \leq k$. $\square$

3.1 Indistinguishable One-Way Roads

In this section, we assume that the searcher is not able to distinguish one-way roads. That is, if the searcher meets an entrance it cannot tell whether this is a new entrance or one that has been met before. Algorithm 2 solves the problem in this setting: We store a control word $w \in \{'\text{r}', '\text{p}', '\text{b}'\}^*$ (see Algorithm 2) where every character determines the searcher's behavior when meeting an entrance. We evaluate this word character by character and generate the next word in lexicographical order when every character is evaluated. Between two entrances, the searcher moves using the Pledge algorithm.

Theorem 2 *Algorithm 2 finds the exit from every fair maze.*

Proof. We use a proof idea similar to [7]. First, we show that there is a universal control word, w_{uni}, that allows the searcher to escape from every start point. Let the searcher start in s_1 and let e_i be the first one-way road that the searcher meets. By Lemma 1, there is a control word w_i that directs the searcher to the exit. Now, let the searcher start in s_2 and met another one-way road $e_j \neq e_i$. If the searcher applys w_i

Algorithm 2: Pledge with indistinguishable one-way roads

```
• $w := ''\text{p}''$, $i := 1$.
• Use the Pledge algorithm, until a one-way
  entrance is met. If the $i$th character in $w$ is
    'p': enter the one-way road
    'b': do not enter
    'r': angle counter $\varphi := \varphi \bmod 2\pi$
  Increment $i$.
  If $i > |w|$ generate the next word and $i := 1$.
  Continue the Pledge algorithm.
```

starting in e_j, it either escapes or ends up on another one-way road e_k. Now, there is a word w_k that leads to the exit, and the concatenation $w_1 := w_k \circ {}'\text{r}' \circ w_i$ finds the exit from two one-way roads. Note that it is necessary to 'reset' the angle counter between two control words to ensure that w_k leads to the exit for every angle-counter value that the searcher may have when it meets e_k, because two different angle-counter values may cause different paths even for the same control word.

This way we continue, until our control word, w_{uni}, finds the exit from every one-way road. Algorithm 2 enumerates all words in $\{ {}'\text{r}', {}'\text{p}', {}'\text{b}' \}^*$ and, thus, eventually finds w_{uni}. □

Needless to mention that Algorithm 2 may try a lot of words until it finds the exit and uses $O(|\mathcal{E}|^2)$ space, so this algorithm is more of theoretical value.

3.2 Distinguishable One-Way Roads

Now, we assume that the searcher is able to uniquely identify entrances to one-way roads. Clearly, this model is more powerful than the one used in the preceding section: We have the advantage of using the entrances of one-way roads as landmarks. Thus, we can uniquely identify regions by the entrances on their outer boundary. Our algorithm builds a graph of the environment with one node per region and directed edges representing the one-way roads leading from one region to another. We traverse this graph using an online strategy for the exploration of directed graphs, see Papadimitriou [13], Albers and Henzinger [2], Kwek [11], or Fleischer and Trippen [5]. The graph exploration strategy is the framework for our maze leaving strategy. Algorithm 3 is a subroutine that is called by the graph explorer when the search begins and every time the searcher has just passed a one-way road that has never been passed before.

Algorithm 3, in turn, uses basically the Pledge algorithm to explore a region. If the searcher arrives at an entrance, the Pledge algorithm is interrupted to gather some information about the region. Algorithm 3 uses a marker for every one-way roads either to store a pair (from, to) of regions or to mark the one-way road as 'unknown so far', 'open forever', or 'closed forever'. The latter is used for one-way roads that are part of the inner boundary of a region that is already visited (i.e., lead to regions that are completely surrounded by this region), see Figure 3(i). The exit of the maze cannot be inside such a region, so we never have to visit it. We use 'open forever' for one-way roads that do not define a region, see Figure 1. For every other one-way road, the decision whether or not to enter it is based on the online exploration strategy for directed graphs.

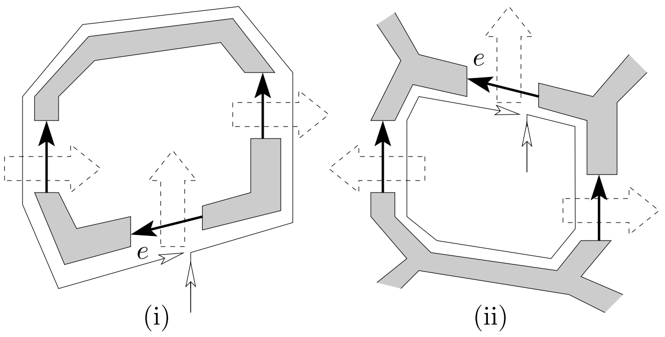


Figure 3: (i) The searcher surrounded a set of inner obstacles, (ii) the searcher moved on the outer boundary of a region.

Theorem 3 *A graph explorer that calls Algorithm 3 from start point s and after a one-way road is entered for the first time, finds the exit from every fair maze.*

Proof. The Pledge algorithm ensures that we eventually reach the outer boundary of a region that we enter either from the start point or after passing a one-way road: Inside a region we interrupt the Pledge algorithm only when we meet an entrance on the inner boundary. After surrounding this part of the inner boundary, we consider this boundary as one obstacle by 'closing' every one-way road that leads inside. Then, we continue the Pledge algorithm with the same angle-counter value as before the interruption. From the correctness of the Pledge algorithm follows that it reaches the bounding box of a maze in a setting without one-way roads; thus, in our setting the Pledge algorithm reaches the outer boundary of a region.

When we reach the first one-way road on the outer boundary of a region, we interrupt the Pledge algorithm again. We completely circle the outer boundary discovering every one-way road that leaves the current region. So we build by and by a graph that contains a node for every region that we enter and an edge for every one-way road between regions. The online graph exploration strategy ensures that this graph is completely traversed until we find the exit. □

Algorithm 3: Pledge with distinguishable one-way roads

- Use the Pledge algorithm, until an entrance to a one-way road e is met.
- If e is 'open forever' or 'closed forever' continue the Pledge algorithm.
- If the 'from'-marker of e is set, update the 'to'-marker of the most recently passed one-way road. (The searcher is inside a known region). If both markers are the same, set the one-way road to 'open forever' and remove it from the graph. Continue the graph explorer.
- If e was never met before, store the current angle-counter value, ω_e, and follow the wall using the *left-hand rule* until e is met again. On this path, store all discovered entrances in a list, ℓ, and count the turning angles. Compare the current angle-counter value to ω_e:
  - If the difference is 2π, the searcher has surrounded an inner obstacle (or a set of inner obstacles), see Figure 3(i). Mark every one-way road in ℓ as 'closed forever'. Set the angle-counter to ω_e and continue the Pledge algorithm.
  - If the difference is -2π, the searcher moved on the outer boundary of a region, see Figure 3(ii). If no entrance on the outer boundary has been met before, we have found a new region: Add a new vertex to the graph. Set the 'from' markers of the one-way roads in ℓ and the 'to'-marker of the most recently passed one-way road to this region. Continue the graph explorer.

Both the graph explorer and Algorithm 3 use $O(|\mathcal{E}|)$ space. The simple strategy by Kwek [11] uses $O(\min\{\, r|\mathcal{E}|, dr^2 + |\mathcal{E}|\,\})$ edge traversals, where r is the number of vertices (regions) and d is the number of edges that have to be added to make the graph Eulerian. The more elaborated strategy by Fleischer and Trippen [5] uses $O(d^8\,|\mathcal{E}|)$ edge traversals. Altogether, the strategy presented in this section is more applicable than the one shown in the preceding section. But it assumes, that the searcher is able to distinguish one-way roads.

4 Conclusion

We introduced polygonal scenes with passages that can be traversed in only one direction, and considered the problem of leaving such a scene. This problem was solved for two different settings—indistinguishable and distinguishable one-way entrances—by combining the Pledge algorithm with other techniques that make the decision whether or not to enter a one-way road: enumerating all control words and exploring directed graphs online. Two other approaches are presented in [4] and in the forthcoming technical report. A strategy that leaves an unknown maze with one-way roads was implemented on a Khepera II robot [4].

The next step may be to ask, if and how other algorithms known for polygonal scenes such as searching, navigation, exploration/covering have to be modified in the presence of one-way roads.

References

[1] H. Abelson and A. A. diSessa. *Turtle Geometry.* MIT Press, Cambridge, 1980.

[2] S. Albers and M. Henzinger. Exploring unknown environments. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 416–425, 1997.

[3] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.*, 26(1):110–137, Feb. 1997.

[4] B. Brüggemann. Entkommen aus unbekannten Labyrinthen mit Einbahnstraßen. Diplomarbeit, University of Bonn, November 2006.

[5] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. In *Proc. 13th Annu. European Sympos. Algorithms*, volume 3669 of *Lecture Notes Comput. Sci.*, pages 11–22. Springer-Verlag, 2005.

[6] U. Handel, T. Kamphans, E. Langetepe, and W. Meiswinkel. Polyrobot — an environment for simulating strategies for robot navigation in polygonal scenes. Java Applet, 2002. http://www.geometrylab.de/Polyrobot/.

[7] A. Hemmerling. *Labyrinth Problems: Labyrinth-Searching Abilities of Automata.* B. G. Teubner, Leipzig, 1989.

[8] T. Kamphans. *Models and Algorithms for Online Exploration and Search.* Dissertation, University of Bonn, 2005. http://www.kamphans.de/k-maole-05.pdf.

[9] T. Kamphans and E. Langetepe. The Pledge algorithm reconsidered under errors in sensors and motion. In *Proc. of the 1th Workshop on Approximation and Online Algorithms*, volume 2909 of *Lecture Notes Comput. Sci.*, pages 165–178. Springer, 2003.

[10] R. Klein. *Algorithmische Geometrie - Grundlagen, Methoden, Anwendungen.* Springer, Heidelberg, 2nd edition, 2005.

[11] S. Kwek. On a simple depth-first search stratey for exploring unknown graphs. In *Proc. 5th Workshop Algorithms Data Struct.*, pages 345–353, 1997.

[12] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[13] C. H. Papadimitriou. On the complexity of edge traversing. *J. ACM*, 23:544–554, 1976.

[14] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoret. Comput. Sci.*, 84(1):127–150, 1991.

A Simple Solution To Two-Guard Street Search Problem

John Z. Zhang*

Abstract

Given a simple polygon P with two prespecified vertices s and g on its boundary and two guards, the two-guard street search problem asks whether two guards can move on the boundary of P from s to g in the opposite directions such that they always maintain their mutual visibility. We revisit this problem and present a simple solution to it.

1 Introduction

A *simple polygon* is defined as a geometric entity consisting of a set of vertices and a set of closed non-intersecting segments connecting adjacent vertices. Those segments compose the boundary of the polygon, which can have extremely complex structures.

The *two-guard street search problem* was first introduced by Icking and Klein [4]. A street is a simple polygon P with two prespecified vertices s (the *entrance*) and g (the *exit*) on its boundary. (In the sequel, we denote a street as $P(s, g)$.) s and g divide the boundary of P into two sides, which are required to be mutually visible, i.e., any point on one side is visible to at least one point on the other, and vice versa. The two-guard street search problem involves determining whether the two guards can move along the boundary of P from s to g, one clockwise and the other counter-clockwise, and maintain their mutual visibility. If this can be accomplished, $P(s, g)$ is said to be *walkable* or *searchable*.

The problem was further investigated. Heffernan [3] showed that linear time was sufficient to check the searchability of a polygon. Tseng *et al.* [9] studied a related problem, obtaining, for given a simple polygon P of n vertices, an $O(n \log n)$-time algorithm to find all the pairs s and g such that $P(s, g)$ is searchable by two guards. Bhattacharya *et al.* [1] proposed a linear-time algorithm to the same problem. For some other related work, interested readers are referred to [2, 6, 5, 8].

The rest of the paper is organized as follows. Sec. 2 introduces the notation used throughout the paper. We will discuss the visibility space diagram of a street in Sec. 3. In Sec. 4, we will present our simple characterization of searchable streets. We conclude the paper by discussing some related results in Sec. 5.

*Department of Mathmatics and Computer Science, University of Lethbridge, zhang@cs.uleth.ca

2 Preliminaries

A simple polygon P is defined by a clockwise sequence of n distinct *vertices* ($n \geq 3$) and *edges* that connect adjacent vertices. The edges form the boundary of P, which is denoted as ∂P. We assume that $\partial P \subseteq P$. The vertices immediately preceding and succeeding vertex v clockwise are denoted by $Pred(v)$ and $Succ(v)$, respectively. For any two points $a, b \in \partial P$, the *polygonal chain* of ∂P clockwise from a to b is denoted by $\partial P_{cw}(a, b)$.

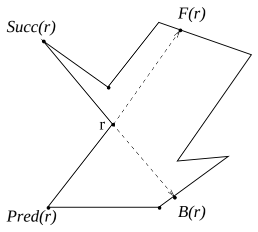


Figure 1: Extensions from a reflex vertex.

We distinguish the *reflex vertices* of P, whose interior angles inside P are larger than 180^o. For a reflex vertex r, we generate two extensions from it. The *backward extension* (resp. *forward extension*) extends $\overline{rSucc(r)}$ (resp. $\overline{rPred(r)}$) within P. The *backward extension point* $B(r)$ (resp. *forward extension point* $F(r)$) is the first intersection between the backward extension (resp. forward extension) and ∂P. See Fig. 1. Two points $u, v \in P$ are said to be *mutually visible* if the line segment $\overline{uv} \subseteq P$.

In street $P(s, g)$, s and g divide ∂P into two sides. We call $\partial P_{cw}(s, g)$ (resp. $\partial P_{cw}(g, s)$) L (resp. R). L and R are mutually weakly visible, i.e., any point on L is visible to at least one point on R, and vice versa. The two guards move along L and R, respectively. Let us use functions $l(t)$ and $r(t)$, where $t \in [0, 1]$, to represent the positions of the two guards on L and R, respectively, i.e. $l : [0, 1] \to L$ and $r : [0, 1] \to R$. The two-guard search problem (adapted from [4]) asks whether $P(s, g)$ is searchable, i.e., $l(0) = r(0) = s$, $l(1) = r(1) = g$, and $r(t)$ and $l(t)$ are mutually visible at any time $t \in [0, 1]$.

It should be noted that in [4], three types of searches - *general*, *straight* and *counter straight* - were discussed. For the sake of space, we only report our result related to the general search in this paper.

3 The visibility space diagram of a street

Imagine that there were intruders sneaking into street $P(s, g)$. The two guards' task is to push them from s to g. The requirement of the mutual visibility between the guards is equivalent to that the portion of the street (containing s) below $\overline{l(t)r(t)}$ at any time t is clear of intruders, while the invisibility between them would allow the intruders to escape the street through s.

We define a *visibility configuration* or just *configuration* to be a pair of boundary points q and p such that $q \in L$ and $p \in R$. The *visibility space* for street $P(s, g)$ is defined as $L \times R = \{\langle q, p\rangle | \ q \in L, p \in R\}$. $\langle p, q\rangle$ is a *visible configuration* if p and q are mutually visible. Otherwise, it is an *invisible configuration*.

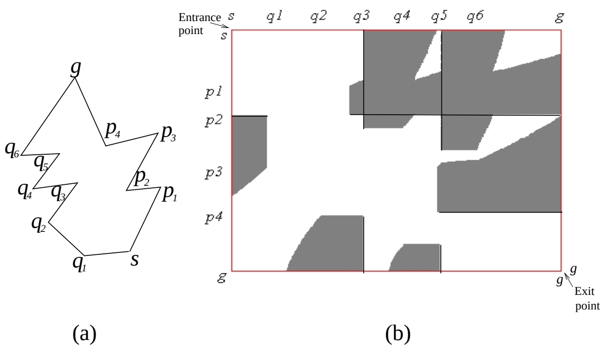


Figure 2: A street and its VSD.

We visualize the visibility space in a rectangle whose sides are horizontal and vertical. The points on L are mapped clockwise onto the horizontal sides (whose length is $|L|$) from left to right while the points on R are mapped counter-clockwise onto the vertical sides (whose length is $|R|$) from top to bottom. A configuration can be uniquely mapped onto a point in this rectangle. Now let us gray the points which correspond to invisible configurations while leaving white those corresponding to visible configurations. We call the rectangle after this coloring the *visibility space diagram (VSD)* of $P(s, g)$. We call its top-left corner the *entrance point* and its bottom-right corner the *exit point.* As an example, for the street shown in Fig. 2 (a), its VSD is shown in Fig. 2 (b). The VSD of a street is essentially the same as the *visibility obstruction diagram* of a polygon proposed by LaValle *et al.* [7]. However, we have adapted it extensively, making it possible for us to conduct searchability analysis related to a street.

Before proceeding, we make some comments on the VSD. The gray areas in a VSD are caused by reflex vertices. For instance, for reflex vertex $q_3 \in L$ in Fig. 2 (a), it is obvious that any point on $\overline{q_3 Succ(q_3)}$ is invisible to any point on $\partial P_{cw}(B(q_3), s)$. This invisibility information, when mapped into the VSD, corresponds to a gray area attaching at q_3 on the top side, whose height is $|\partial P_{cw}(B(q_3), s)|$, as shown in Fig. 2 (b). Similarly, a gray area attaching on the bottom side is due to a reflex vertex on L, whose forward extension point is on R. The gray areas from the left and right sides are due to the reflex vertices on R, whose forward extension points and/or backward extension points are on L.

The two guards start at time $t = 0$, where $\langle l(0), r(0)\rangle$ corresponds to the entrance point. If $P(s, g)$ is searchable, at any time $t \in (0, 1)$, $\langle l(t), r(t)\rangle$ is a visible configuration, which corresponds to a white point in the VSD. The search ends at time $t = 1$ at g, where $\langle l(1), r(1)\rangle$ is at the exit point.

Imagine that, as time t changes from 0 to 1, the points corresponding to $\langle l(t), r(t)\rangle$ form a continuous path within the white area in the VSD, which starts at the entrance point and ends at the exit point. We call it the *legal path.* Conversely, if there exists a legal path in the VSD, $P(s, g)$ is searchable by two guards, since any point on the path is corresponding to a visible configuration, which can represent the current positions of the two guards on L and R, respectively.

Proposition 1 *Street $P(s, g)$ is searchable by two guards if and only if there exists a legal path inside the VSD of $P(s, g)$.*

The following is obvious due to the requirement of the mutual visibility between the two guards.

Proposition 2 *No point on a legal path can be in a gray area and a legal path never crosses through a gray area.*

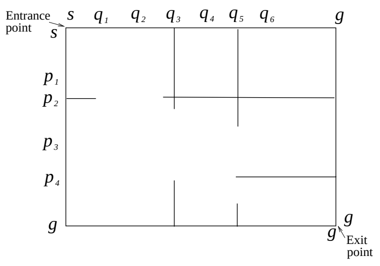


Figure 3: The SVSD of the street shown in Fig. 2.

When we use the VSD of a street to help conduct the searchability analysis, we can focus on the skeletal version of the diagram. Consider, for instance, a reflex vertex $r \in L$, where $B(r) \in R$. The height of the gray area (due to it) tells us the degree of invisibility that r causes on R. Therefore, instead of using the whole gray area in our analysis, we use the line segment that sustain the gray area, whose length is equal to $|\partial P_{cw}(B(r), s)|$. For example, we highlight the sustaining segment for each gray area in Fig. 2 (b). The diagram thus obtained is called the *skeletal visibility space diagram (SVSD)* of $P(s, g)$. For the street in Fig. 2 (a), its corresponding SVSD is shown in Fig. 3.

As commented in [7], the visibility space diagram and its skeleton counterpart are topologically equivalent. In the SVSD, abusing the notation, we still call the top-left corner the *entrance point* and the bottom-right corner the *exit point*. For convenience, we call the segments attaching on the top side *north segments (NS)*, the ones on the right side *east segments (ES)*, the ones on the bottom side *south segments (SS)*, and the ones on the left side *west segments (WS)*. Each segment originates from one side of the SVSD and ends at its *tip*.

4 A simple characterization of seachable streets

The characterization of searchable streets was discussed in [4]. In this section, we will derive an equivalent one using the SVSD of a street. The proof of our characterization is simple, and in addition, we believe that our approach is applicable to other related problems as well.

Clearly, due to the relationship between the VSD and SVSD, in the VSD there is a legal path if and only if in the counterpart SVSD we can construct a path which starts at the entrance point and ends at the exit point. We still call such a path in the SVSD *legal path*. According to Proposition 2, such a path never crosses a segment.

Proposition 3 *Street $P(s, g)$ is searchable by two guards if and only if there exists a legal path inside the SVSD of $P(s, g)$.*

Therefore, in order to obtain the characterization of searchable streets by two guards, we need to analyze what "patterns", in terms of the relationship among the segments in the SVSD, could make it impossible to construct a legal path.

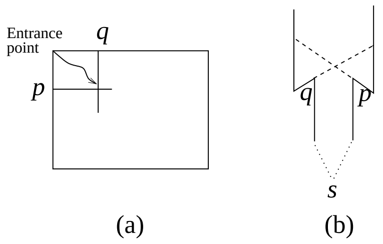


Figure 4: The situation where the entrance point is trapped.

Consider a north segment and a west segment, as shown in Fig. 4 (a), which intersect within the SVSD. The entrance point is said *trapped*. Under this situation, no legal path is possible, since, no matter how we extend it, it is confined within the rectangle formed by the top side of the SVSD, the left side, and the two segments. The corresponding street is thus not searchable. It is easy to see that the pattern corresponds to some geometric entity called *deadlock* [4] in the street itself, as shown in Fig. 4 (b).

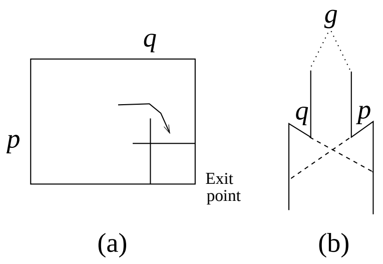


Figure 5: The situation where the exit point is trapped.

Lemma 4 *For street $P(s, g)$, if the entrance point is trapped in the SVSD, it is not searchable by two guards.*

On the other hand, consider the intersection between a south segment and a west segment in Fig. 5 (a). We say that the exit point is *trapped*. Even though we can extend a path from the entrance point, there is no way that the path can be extended into the rectangle shown in the figure and reach the exit point. Such a pattern corresponds to a deadlock containing exit g in the street, as shown in Fig. 5 (b).

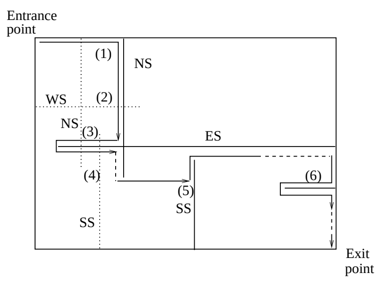


Figure 6: The illustration for the proof of Lemma 6.

Lemma 5 *For street $P(s, g)$, if the exit point is trapped in the SVSD, it is not searchable by two guards.*

Lemma 6 *For street $P(s, g)$, if neither the entrance point nor the exit point is trapped, $P(s, g)$ is searchable by two guards.*

Proof. We use Fig. 6 as a reference.[1] Within the SVSD of $P(s, g)$, we attempt to construct a path from the entrance point. We divide the construction into two stages. In the first stage, we extend the path eastwards along the top side. There are two cases. The path is about to either hit the right side of the

[1]We omit the irrelevant segments in the figure.

SVSD, in which case we switch to the second stage, or to hit an NS, in which case the path is extended along the NS downwards (1).

Such an extension should be able to reach the tip of the NS. We need to consider two cases here. (a) It is about to hit a WS (2). However, this is impossible, since this means that the NS and WS make the entrance point trapped. (b) It is about to hit a ES. We can then continue the extension along the ES westwards. We do not need to worry about any NS that intersects with the ES (3), since this means that the path has been extended into the rectangle formed by the top side, the right side, the NS and the ES. This is impossible since the path cannot cross any segment. Going around the tip of the ES, we extend the path along the ES eastwards, attempting to reach the NS (from (1)). No SS can stop our extension (4), since if so, the exit point is trapped. When the path is about to hit the NS, we extend it downwards again.

We may need to repeat the above process, extending the path around the tips of ESs. But eventually, the path should be at the tip of the NS (from (1)). After that, we extend the path eastwards, until the path hits the next NS, in which case we repeat the process. If the path hits an SS (5), it is extended upwards along it. Any ES stopping this extension would mean that the exit point is trapped. On the other hand, any WS stopping this extension would mean that the path has been extended into the rectangle formed by the SS, the WS and the two sides of the SVSD. After the tip of the SS is reached, the path is extended again eastwards until the next NS and SS. If it hits the right side of the SVSD, we go to the second stage.

At the beginning of the second stage, the path is about to hit the right side. We extend the path downwards, attempting to reach the exit point. The extension may encounter an ES (6). But it can always go around it, by the similar reasoning as above, and go back to the right side. It is impossible for it to come across any WS, since this means that the two sides of the street are not mutually weakly visible. Therefore the path should be able to reach the exit point. □

Theorem 7 *Street $P(s, g)$ is searchable by two guards if and only if in its SVSD neither the entrance point nor the exit point is trapped.*

As discussed after Lemmas 4 and 5, the fact that the entrance point or the exit point is trapped corresponds deadlocks in the street. Therefore, the characterization stated in Theorem 7 is equivalent to the one in [4]. However, its proof is simple, avoiding the previous tedious and complex analysis.

5 Discussions

We reported in this paper our initial attempts to simplify the street search problem. By extracting the visibility information in a street and using it to guide our analysis, we derived a simple characterization of searchable streets, in terms of its understandability and interpretation. Actually the same merit applies to our solutions to the other related problems in streets as well. Due to limited space, we do not present the relevant results here.

In addition, the simplicity of our characterizations has shed light on our efforts to simplify the previous approach to achieve the optimal-time complexity for checking the searchability and generating a search schedule for a street.

The next step of our work is naturally to find all the entrance and exit pairs in a polygon that can form a searchable street. We believe that the same approach reported here can be harnessed and further explored.

Acknowledgments

We would like to thank the anonymous readers for their comments on the paper.

References

[1] B.K. Bhattacharya, A. Mukhopadhyay, and G. Narasimhan. Optimal algorithms for two-guard walkability of simple polygons. In *Proc. 7th Int'l Workshop on Algorithms and Data Structures*, pages 438–449, 2001.

[2] D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor: the open edge variant of the polygon search problem. *Int'l J. of Computational Geometry and Applications*, 5(4):397–412, 1995.

[3] P. Heffernan. An optimal algorithm for the two-guard problem. *Int'l J. of Computational Geometry and Applications*, 6:15–44, 1996.

[4] C. Icking and R. Klein. The two guards problem. *Int'l J. of Computational Geometry and Applications*, 2(3):257–285, 1992.

[5] C. Icking, R. Klein, and E. Langetepe. An optimal competitive strategy for walking in streets. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science 1999*, number 1563 in Lecture Notes in Computer Science, pages 110–120. Springer-Verlag, 1999.

[6] J. M. Kleinberg. On-line search in a simple polygon. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 8–15, 1994.

[7] S. M. LaValle, B. Simov, and G. Slutzki. An algorithm for searching a polygonal region with a flashlight. *Int'l J. of Computational Geometry and Applications*, 12(1-2):87–113, 2002.

[8] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. on Computing*, 21(5):863–888, October 1992.

[9] L. H. Tseng, P. J. Heffernan, and D. T. Lee. Two-guard walkability of simple polygons. *Int'l J. of Computational Geometry and Applications*, 8(1):85–116, 1998.

Maximizing Maximal Angles for Plane Straight Line Graphs

Oswin Aichholzer[*] Thomas Hackl[*] Michael Hoffmann[†] Clemens Huemer[‡] Francisco Santos[§]
Bettina Speckmann[¶] Birgit Vogtenhuber[*]

Abstract

Let $G = (S, E)$ be a plane straight line graph on a finite point set $S \subset \mathbb{R}^2$ in general position. For a point $p \in S$ let the *maximum incident angle* of p in G be the maximum angle between any two edges of G that appear consecutively in the circular order of the edges incident to p. A plane straight line graph is called φ-*open* if each vertex has an incident angle of size at least φ. In this paper we study the following type of question: What is the maximum angle φ such that for any finite set $S \subset \mathbb{R}^2$ of points in general position we can find a graph from a certain class of graphs on S that is φ-open? In particular, we consider the classes of triangulations, spanning trees, and paths on S and give tight bounds in all but one cases.

1 Introduction

Conditions on angles in plane straight-line graphs have been studied extensively in discrete and computational geometry. It is well known that Delaunay triangulations maximize the minimum angle over all triangulations, and that in a (Euclidean) minimum weight spanning tree each angle is at least $\frac{\pi}{3}$. In this paper we address the fundamental combinatorial question, what is the maximum value α such that for each finite point set in general position there exists a plane straight-line graph (of a certain type) where each vertex has an incident angle of size at least α. We present bounds on this value for three classes of graphs: spanning paths (general and bounded degree), spanning trees, and triangulations. Most of the bounds we give are tight. In order to show that, we describe families of point sets for which no graph from the respective class can achieve a greater incident angle at all vertices.

Background. Our motivation for this research stems from the investigation of "pseudo-triangulations", a straight-line framework which, apart from deep combinatorial properties, has applications in motion planning, collision detection, ray shooting and visibility; see [1, 9, 10, 12, 13] and references therein. Pseudo-triangulations with a minimum number of pseudo-triangles (among all pseudo-triangulations for a given point set) are called *minimum* (or *pointed*) pseudo-triangulations. They can be characterized as plane straight-line graphs where each vertex has an incident angle greater than π. Furthermore, the number of edges in a minimum pseudo-triangulation is maximal, in the sense that the addition of any edge produces an edge-crossing or negates the angle condition.

In comparison to these properties, we consider connected plane straight-line graphs where each vertex has an incident angle α – to be maximized – and the number of edges is minimal (spanning trees) and the vertex degree is bounded (spanning trees of bounded degree and spanning paths, respectively). We further show that any planar point set has a triangulation in which each vertex has an incident angle of at least $\frac{2\pi}{3}$. Observe that perfect matchings can be described as plane straight-line graphs where each vertex has an incident angle of 2π and the number of edges is maximal.

Related Work. There is a vast literature on triangulations that are optimal according to certain criteria. Similar to Delaunay triangulations which maximize the smallest angle over all triangulations for a point set, farthest point Delaunay triangulations minimize the smallest angle over all triangulations for a convex polygon [6]. If all angles in a triangulation are $\geq \frac{\pi}{6}$, then it contains the relative neighborhood graph as a subgraph [11]. The relative neighborhood graph for a point set connects any pair of points which are mutually closest to each other (among all points from the set). Edelsbrunner et al. [7] showed how to construct a triangulation that minimizes the maximum angle among all triangulations for a set of n points in $O(n^2 \log n)$ time.

[*]Institute for Software Technology, Graz University of Technology, [oaich|thackl|bvogt]@ist.tugraz.at. Supported by the Austrian FWF Joint Research Project 'Industrial Geometry' S9205-N12.

[†]Institute for Theoretical Computer Science, ETH Zürich, hoffmann@inf.ethz.ch

[‡]Departament de Matemática Aplicada II, Universitat Politécnica de Catalunya, Huemer.Clemens@upc.edu. Partially supported by projects MEC MTM2006-01267 and Gen. Cat. 2005SGR00692.

[§]Departamento de Matemáticas, Estadística y Computación, Universidad de Cantabria, francisco.santos @unican.es. Partially supported by grant MTM2005-08618-C02-02 of the Spanish Ministry of Education and Science.

[¶]Department of Mathematics and Computer Science, TU Eindhoven, speckman@win.tue.nl

In applications where small angles have to be avoided by all means, a Delaunay triangulation may not be sufficient in spite of its optimality because even there arbitrarily small angles can occur. By adding so-called Steiner points one can construct a triangulation on a superset of the original points in which there is some absolute lower bound on the size of the smallest angle [4]. Dai et al. [5] describe several heuristics to construct minimum weight triangulations (triangulations which minimize the total sum of edge lengths) subject to absolute lower or upper bounds on the occurring angles.

Spanning cycles with angle constraints can be regarded as a variation of the traveling salesman problem. Fekete and Woeginger [8] showed that if the cycle may cross itself then any set of at least five points admits a locally convex tour, that is, a tour in which the angle between any three consecutive points is positive. Aggarwal et al. [2] prove that finding a spanning cycle for a point set which has minimal total angle cost is NP-hard, where the angle cost is defined as the sum of direction changes at the points.

Regarding spanning paths, it has been conjectured that each planar point set admits a spanning path with minimum angle at least $\frac{\pi}{6}$ [8]; recently, a lower bound of $\frac{\pi}{9}$ has been presented [3].

Definitions and Notation. Let $S \subset \mathbb{R}^2$ be a finite set of points in general position, that is, no three points of S are collinear. In this paper we consider plane straight line graphs $G = (S, E)$ on S. The vertices of G are precisely the points in S, the edges of G are straight line segments that connect two points in S, and two edges of G do not intersect except possibly at their endpoints. For a point $p \in S$ the *maximum incident angle* $\mathrm{op}_G(p)$ of p in G is the maximum angle between any two edges of G that appear consecutively in the circular order of the edges incident to p. For a vertex $p \in S$ of degree at most one we set $\mathrm{op}_G(p) = 2\pi$. We also refer to $\mathrm{op}_G(p)$ as the *openness* of p in G and call $p \in S$ *φ-open* in G for some angle φ if $\mathrm{op}_G(p) \geq \varphi$. Consider, for example, the graph depicted in Figure 1. The point p has four incident edges in G and, therefore, four incident angles. Its openness is $\mathrm{op}_G(p) = \alpha$. The point q has only one incident angle and correspondingly $\mathrm{op}_G(q) = 2\pi$.

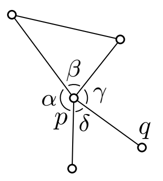


Figure 1: Incident angles of p.

Similarly we define the *openness* of a plane straight line graph $G = (S, E)$ as $\mathrm{op}(G) = \min_{p \in S} \mathrm{op}_G(p)$ and call G *φ-open* for some angle φ if $\mathrm{op}(G) \geq \varphi$. In other words, a graph is φ-open if and only if every vertex has an incident angle of size at least φ. The *openness* of a class $\mathcal{G}$ of graphs is the supremum over all angles φ such that for every finite point set $S \subset \mathbb{R}^2$ in general position there exists a φ-open connected plane straight line graph G on S and G is an embedding of some graph from $\mathcal{G}$. For example, the openness of minimum pseudo-triangulations is π.

Observe that without the general position assumption many of the questions become trivial because for a set of collinear points the non-crossing spanning tree is unique – the path that connects them along the line – and its interior points have no incident angle greater than π.

Let a, b, and c be three points in the plane that are not collinear. With $\angle abc$ we denote the counterclockwise angle between the segment (b, a) and the segment (b, c) at b.

Results. In this paper we study the openness of several well known classes of plane straight line graphs, such as triangulations ($\frac{2\pi}{3}$, Section 2), spanning trees (Section 3) in general ($\frac{5\pi}{3}$) and with maximum degree three ($\frac{3\pi}{2}$), and spanning paths ($\frac{3\pi}{2}$ for sets in convex position, Section 4).

2 Triangulations

Theorem 1 *Every finite point set in general position in the plane has a triangulation that is $\frac{2\pi}{3}$-open and this is the best possible bound.*

Proof. Consider a point set $S \subset \mathbb{R}^2$ in general position. Clearly, $\mathrm{op}_G(p) > \pi$ for every point $p \in \mathrm{CH}(S)$ and every plane straight line graph G on S. We recursively construct a $\frac{2\pi}{3}$-open triangulation T of S by first triangulating $\mathrm{CH}(S)$; every recursive subproblem consists of a point set with a triangular convex hull.

Let S be a point set with a triangular convex hull and denote the three points of $\mathrm{CH}(S)$ with a, b, and c. If S has no interior points, then we are done. Otherwise, let a', b' and c' be (not necessarily distinct) interior points of S such that the triangles $\Delta a'bc$, $\Delta ab'c$ and $\Delta abc'$ are empty (see Figure 2). Since the sum of the six exterior angles of the hexagon $bc'ab'ca'$ equals 8π, the sum of the three angels $\angle ac'b$, $\angle ba'c$, and $\angle cb'a$ is at least 2π. In particular, one of them, say $\angle cb'a$, is at least $2\pi/3$. We then recurse on the two subsets of S that have $\Delta bb'c$ and $\Delta ab'b$ as their respective convex hulls.

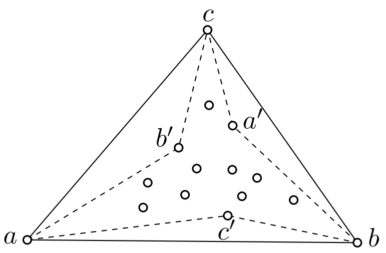


Figure 2: Constructing a $\frac{2\pi}{3}$-open triangulation.

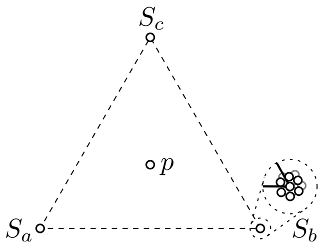


Figure 3: The openness of triangulations of this point set approaches $\frac{2\pi}{3}$.

The upper bound is attained by a set S of n points as depicted in Figure 3. S consists of a point p and of three sets S_a, S_b, and S_c that each contain $\frac{n-1}{3}$ points. S_a, S_b, and S_c are placed at the vertices of an equilateral triangle Δ and p is placed at the barycenter of Δ. Any triangulation T of S must connect p with at least one point of each of S_a, S_b, and S_c and hence $\mathrm{op}_T(p)$ approaches $\frac{2\pi}{3}$ arbitrarily close. $\square$

3 Spanning Trees

In this section we give tight bounds on the φ-openness of two basic types of spanning trees, namely general spanning trees and spanning trees with bounded vertex degree. Consider a point set $S \subset \mathbb{R}^2$ in general position and let p and q be two arbitrary points of S. Assume w.l.o.g. that p has smaller x-coordinate than q. Let l_p and l_q denote the lines through p and q that are perpendicular to the edge (p, q). We refer to the region bounded by l_p and l_q as the *orthogonal slab* of (p, q).

Observation 1 *Assume that $r \in S \setminus \{p, q\}$ lies in the orthogonal slab of (p, q) and above (p, q). Then $\angle qpr \leq \frac{\pi}{2}$ and $\angle rqp \leq \frac{\pi}{2}$. A symmetric observation holds if r lies below (p, q).*

Recall that the diameter of a point set is a pair of points that are furthest away from each other. Let a and b define the diameter of S and assume w.l.o.g. that a has a smaller x-coordinate than b. Clearly, all points in $S \setminus \{a, b\}$ lie in the orthogonal slab of (a, b).

Observation 2 *Assume that $r \in S \setminus \{a, b\}$ lies above a diametrical segment (a, b) for S. Then $\angle arb \geq \frac{\pi}{3}$ and hence at least one of the angles $\angle bar$ and $\angle rba$ is at most $\frac{\pi}{3}$. A symmetric observation holds if r lies below (a, b).*

These two simple observations can be used to obtain the following results on spanning trees.

Theorem 2 *Every finite point set in general position in the plane has a spanning tree that is $\frac{5\pi}{3}$-open, and this bound is tight.*

Theorem 3 *Let $S \subset \mathbb{R}^2$ be a set of n points in general position. There exists a $\frac{3\pi}{2}$-open spanning tree T of S such that every point from S has vertex degree at most 3 in T. The angle bound is best possible, even for the much broader class of spanning trees of vertex degree at most $n - 2$.*

Both proofs for the above theorems are based on an extensive case analysis. Therefore we omit them in this extended abstract. The interested reader can find all details in the full version of the paper or in [14].

4 Spanning Paths

For spanning paths, the upper bound for trees with bounded vertex degree can be applied as well. The resulting bound of $\frac{3\pi}{2}$ is tight for points in convex position, even in a very strong sense: There exists a $\frac{3\pi}{2}$-open spanning path starting from any point. We also give examples showing that our construction cannot be extended to general point sets.

4.1 Point Sets in Convex Position

Consider a set $S \subset \mathbb{R}^2$ of n points in convex position. We can construct a spanning path for S by starting at an arbitrary point $p \in S$ and recursively taking one of the tangents from p to $\mathrm{CH}(S \setminus \{p\})$. As long as $|S| > 2$, there are two tangents from p to $\mathrm{CH}(S \setminus \{p\})$: the left tangent is the oriented line t_ℓ through p and a point from $p_\ell \in S \setminus \{p\}$ (oriented in direction from p to p_ℓ) such that no point from S is to the left of t_ℓ. Similarly, the right tangent is the oriented line t_r through p and a point from $p_r \in S \setminus \{p\}$ (oriented in direction from p to p_r) such that no point from S is to the right of t_r. If we take the left and the right tangent alternately, we call the resulting path a *zigzag* path for S.

Theorem 4 *Every finite point set in convex position in the plane admits a spanning path that is $\frac{3\pi}{2}$-open, and this bound is best possible.*

In the full version of the paper, we present two different proofs for this theorem, an existential proof using counting arguments and a constructive proof. In addition, the latter provides the following stronger statement.

Corollary 5 *For any finite set $S \subset \mathbb{R}^2$ of points in convex position and any $p \in S$ there exists a $\frac{3\pi}{2}$-open spanning path for S which has p as an endpoint.*

4.2 General Point Sets

So far we have not been able to generalize the results of Theorem 4 and Corollary 5 to general point sets. In this section we present a few examples to indicate where the difficulties lie.

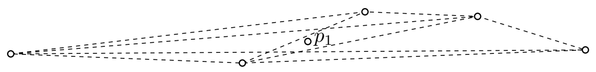


Figure 4: Starting at interior point p_1 results in an at most $(\pi + \varepsilon)$-open spanning path.

Figure 4 depicts a configuration where any spanning path starting at the interior point p_1 is at most $(\pi + \varepsilon)$-open. Figure 5 shows a configuration that has a similar property. Here point p_5 is positioned arbitrarily far to the left and $\beta = \frac{\pi}{3}$. If we require the edge (p_1, p_2) to be part of the spanning path, then we can construct at most a $\left(\frac{4\pi}{3} + \varepsilon\right)$-open spanning path.

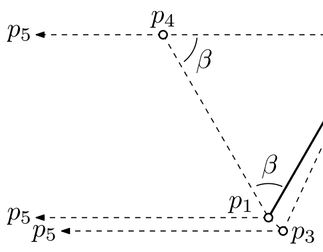


Figure 5: If edge (p_1, p_2) is forced we get at most a $\left(\frac{4\pi}{3} + \varepsilon\right)$-open spanning path.

Both examples show that, whatever approach is used to generate a spanning path, we have to be careful when forcing points or edges to play a specific role in the construction. Especially starting at a fixed interior point has to be avoided.

A direct generalization of the constructive approach for convex sets would be a path which starts at a given extreme point and recursively continues only along tangents to the remaining point set. But there exist examples where this approach generates an at most $(\pi + \varepsilon)$-open spanning path. Details on this construction and the examples presented above, as well as a large variety of much more involved approaches can be found in [14].

On the other hand, and despite the above presented constructions, we have not been able to provide a single point set, which does not contain a $\frac{3\pi}{2}$-open spanning path. To the contrary, computational investigations on several billion random point sets (in the range of $4 \leq n \leq 20$ points) provided for each set a $\frac{3\pi}{2}$-open spanning path, even if we required the path to start with a prefixed extreme point. Thus we conclude this section with the following conjecture.

Conjecture 1 *Spanning paths for general point sets are $\frac{3\pi}{2}$-open.*

Acknowledgments. Research on this topic was initiated at the third European Pseudo-Triangulation working week in Berlin, organized by Günter Rote and André Schulz. We thank Sarah Kappes, Hannes Krasser, David Orden, Günter Rote, André Schulz, Ileana Streinu, and Louis Theran for many valuable discussions. We also thank Sonja Čukić and Günter Rote for helpful comments on the manuscript.

References

[1] O. Aichholzer, F. Aurenhammer, H. Krasser, and P. Brass. *Pseudo-Triangulations from Surfaces and a Novel Type of Edge Flip.* SIAM J. Comput. 32, 6 (2003), 1621–1653.

[2] A. Aggarwal, D. Coppersmith, S. Khanna, R. Motwani, and B. Schieber. *The Angular-Metric Traveling Salesman Problem.* SIAM J. Comput. 29, 3 (1999), 697–711.

[3] I. Bárány, A. Pór, and P. Valtr. *Paths with no Small Angles.* Manuscript in preparation, 2006.

[4] M. Bern, D. Eppstein, and J. Gilbert. *Provably Good Mesh Generation.* J. Comput. Syst. Sci. 48, 3 (1994), 384–409.

[5] Y. Dai, N. Katoh, and S.-W. Cheng. *LMT-Skeleton Heuristics for Several New Classes of Optimal Triangulations.* Comput. Geom. Theory Appl. 17, 1–2 (2000), 51–68.

[6] D. Eppstein. *The Farthest Point Delaunay Triangulation Minimizes Angles.* Comput. Geom. Theory Appl. 1, 3 (1992), 143–148.

[7] H. Edelsbrunner, T. S. Tan, and R. Waupotitsch. *An $O(n^2 \log n)$ Time Algorithm for the Minmax Angle Triangulation.* SIAM J. Sci. Stat. Comput. 13, 4 (1992), 994–1008.

[8] S. P. Fekete and G. J. Woeginger. *Angle-Restricted Tours in the Plane.* Comput. Geom. Theory Appl. 8, 4 (1997), 195–218.

[9] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, and W. Whiteley. *Planar Minimally Rigid Graphs and Pseudo-Triangulations.* Comput. Geom. Theory Appl. 31, 1–2 (2005), 31–61.

[10] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. *Kinetic Collision Detection for Simple Polygons.* Internat. J. Comput. Geom. Appl. 12, 1–2 (2002), 3–27.

[11] J. M. Keil and T. S. Vassilev. *The Relative Neighbourhood Graph is a Part of Every 30°-Triangulation.* Abstracts 21st European Workshop Comput. Geom., 2005, 9–12.

[12] G. Rote, F. Santos, and I. Streinu. *Pseudo-Triangulations — a Survey.* Manuscript, 2006.

[13] I. Streinu. Pseudo-Triangulations, *Rigidity and Motion Planning.* Discrete Comput. Geom. 34, 4 (2005), 587–635.

[14] B. Vogtenhuber. *On Plane Straight Line Graphs.* Master Thesis, Graz University of Technology, Graz, Austria, 2006.

Triple-loop networks with an arbitrarily big number of associated minimum distance diagrams [*]

Pilar Sabariego[†] Francisco Santos[‡]

Abstract

We say that a *minimum distance diagram* (MDD, for short) for a multi-loop network is *coherent* if, whenever the shortest path from v_1 to v_2 in the MDD passes through a third vertex v_3, then the two subpaths induced are the shortest paths that the diagram gives from v_1 to v_3 and from v_3 to v_2. In contrast to the (known) fact that every double loop network has at most two coherent MDD's, we here show that there are triple-loop networks with arbitrarily many.

Our methods exploit the relations between MDD's and monomial ideals, introduced in [3]. Our result shows that characterizing MDD's of triple-loop networks combinatorially, which has been a topic of study in recent years, is going to be a difficult task.

1 Introduction

Multi-loop networks, that is, directed circulant graphs, were proposed in 1974 by Wong and Coppersmith for organizing multi-module memory services [5]. The double-loop case has been widely studied, but many problems remain in the general case.

In the double-loop case, a very useful tool are the so-called *L-shapes* which are a special case of minimum distance diagrams. *Minimum distance diagrams*, or *MDD* for short, are arrays that give (one of) the shortest paths form every node to every other node. In particular, from an MDD one can easily compute the diameter or the average distance of the corresponding network.

By contrast, MDD's for triple-loop networks do not have a uniform nice shape like the *L-shapes* in dimension two, and this fact has made the study of the properties of the triple-loop networks difficult. For example, [1] proposed the study of a particular type of tiles that they called *hyper-L tiles*, but it was shown in [2] that only triple-loop networks with very special parameters possess these type of MDD's.

*Research partially supported by the Spanish Ministry of Education and Science, grant number MTM2005-08618-C02-02.

†Departamento de Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain, pilar.sabariego@unican.es

‡Departamento de Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain, francisco.santos@unican.es

Here, we show that there are triple-loop networks with an arbitrarily big number of coherent MDD's associated (see Figure 6).We use the fact that a coherent MDD is the same as (the set of standard monomials of) one of the monomial ideals associated to the network (see [3]). For double-loop networks, L-shapes are precisely the coherent MDD's.

2 Circulant digraphs and monomial ideals

In this section, we recall some results from Gómez et al. in [3], and extend them by giving in Theorem 4 a simpler characterization of the binomial ideal associated to a multi-loop network.

Definition 1 *The* multi-loop network *or* circulant digraph *of size N and with steps $s_1, ...s_r$ is a directed graph which has nodes $0, 1, \dots, N-1$ and rN links:*

$$i \to i + s_j \mod N,$$

for all $i = 0, 1, \dots, N-1$ and $j = 0, 1, \dots, r$.

We denote it $C_N(s_1, ...s_r)$ and call it "double" or "triple" if $r = 2$ or $r = 3$, respectively.

In other words, $C_N(s_1, ...s_r)$ is the Cayley digraph of the cyclic group $\mathbb{Z}_N$ with respect to $\{s_1, \dots, s_r\}$. Clearly, $C_N(s_1, ...s_r)$ is vertex-transitive and of degree r. It is connected if and only if $\gcd(N, s_1, ..., s_r) = 1$, which we always assume. Figure 1 shows the triple-loop network $C_{10}(1, 2, 3)$.

The *routing map* associated to the circulant digraph $C_N(s_1, ...s_r)$ is the map:

$$\begin{array}{rcl} R := \mathbb{N}^r & \longrightarrow & \mathbb{Z}_N \\ \mathbf{a} & \longmapsto & a_1 s_1 + \cdots + a_r s_r \end{array}$$

That is, $R(a_1, \dots, a_r)$ tells where in the graph you end up if you start at zero and do $a_1, \dots, a_r$ steps via the edges of type $1, \dots, r$. Observe that the order in which different steps are performed is irrelevant.

A minimum distance diagram is a right inverse of the routing map that point-wise minimizes the L_1 norm:

Definition 2 *A* minimum distance diagram *(MDD) of $C_N(s_1, ...s_r)$ is a map $D : \mathbb{Z}_N \longrightarrow \mathbb{N}^r$ such that $R(D(c)) = c, \forall c \in \mathbb{Z}_N$ and*

$$\|D(c)\|_1 = \min\{\|\mathbf{x}\|_1 : \mathbf{x} \in R^{-1}(c)\}$$

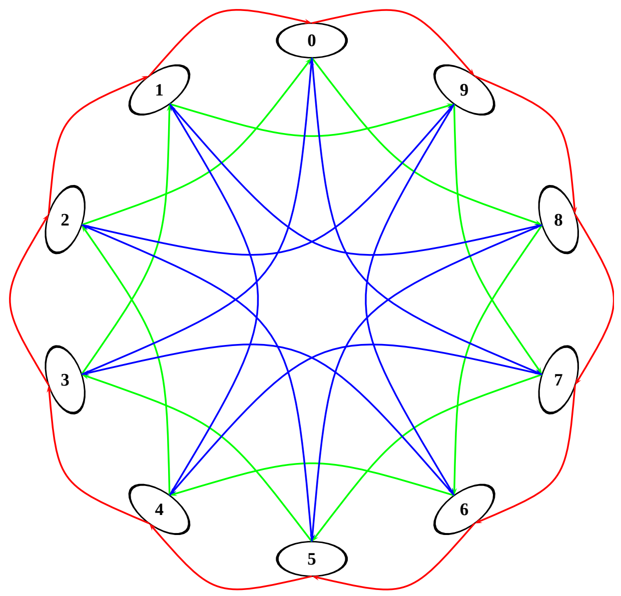


Figure 1: Graphic representation of the circulant graph $C_{10}(1, 2, 3)$

Here and in what follows, $\mathbb{N} = \{0, 1, 2, 3, \dots\}$. MDD's give minimum paths from vertex 0 to any other vertex and, by transitivity, between any two vertices in the network. They admit (for small r) a nice graphical representation as a "stack of labeled boxes": boxes represent elements of $\mathbb{N}^r$ and they are labeled by the numbers $0, \dots, N-1$. See two MDD's for the network $C_6(1, 2, 3)$ in Figure 2. They differ in the choice of shortest path from 0 to 4. To understand the concept of "coherent" MDD, think of the Figure as representing six of the seven blocks of certain MDD's for the network $C_8(1, 2, 3)$. To complete them we need to add the box $(0, 0, 2)$ labeled "6" and either of the boxes $(1, 0, 2)$ or $(0, 2, 1)$ labeled "7" (since $1 + 3 + 3 = 2 + 2 + 3 = 7$). But only one of the choices for "7" is coherent with each choice for "4".

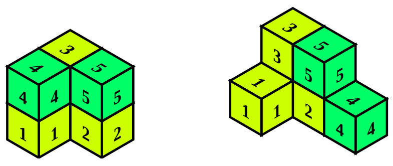


Figure 2: Two MDD's for the network $C_6(1, 2, 3)$.

We now switch to a more algebraic language. Let $\mathbb{K}$ be an arbitrary field and $\mathbb{K}[x_1, ..., x_r]$ the polynomial ring in the variables $x_1, ..., x_r$. As customary, we identify monomials of $\mathbb{K}[x_1, ..., x_r]$ with vectors of $\mathbb{N}^r$ in the following way:

$$\mathbf{x}^{\mathbf{a}} = x_1^{a_1} \cdots x_r^{a_r} \longleftrightarrow \mathbf{a} = (a_1, \dots, a_r).$$

A monomial ideal is any ideal generated by monomials. It can be thought of as a set I of monomials (i.e., a subset of $\mathbb{N}^r$) with the property that $v \in I$ implies $v + w \in I$ for every $w \in \mathbb{N}^r$. Equivalently, $I \subset \mathbb{N}$ is an ideal if its complement (a.k.a. "its set of standard monomials") satisfies

$$v + w \notin I \Rightarrow v, w \notin I.$$

Hence:

Lemma 1 *An MDD is coherent if and only if it is the complement of an ideal.*

A monomial ordering is a total ordering on $\mathbb{N}^r$ that is invariant under addition and in which 0 is the unique minimum. We say a monomial ordering is *graded* if it extends the (partial) ordering given by the L_1 norm (or total degree) of monomials.

Lemma 2 *Every graded monomial ordering $\prec$ induces a* coherent MDD, *the map:*

$$\begin{array}{rcl} D_\prec : \mathbb{Z}_N & \longrightarrow & \mathbb{N}^r \\ c & \longmapsto & \min{}_\prec(R^{-1}(c)) \end{array}$$

The main result in [3] is that coherent MDD's can be obtained as the initial ideals of a certain *lattice ideal.* Recall that every integral lattice $\mathcal{L} \subset \mathbb{Z}^r$ has an associated binomial ideal:

$$I_\mathcal{L} := \left\langle \mathbf{x}^{\mathbf{a}^+} - \mathbf{x}^{\mathbf{a}^-} : \mathbf{a} \in \mathcal{L} \right\rangle \subset \mathbb{K}[x_1, \dots, x_r],$$

where $\mathbf{a} = \mathbf{a}^+ - \mathbf{a}^-$ is the unique decomposition of $\mathbf{a}$ with $\mathbf{a}^+, \mathbf{a}^- \in \mathbb{N}^r$. Moreover (see [4]):

$$\mathbf{x}^\mathbf{a} - \mathbf{x}^\mathbf{b} \in I_\mathcal{L} \Longleftrightarrow \mathbf{a} - \mathbf{b} \in \mathcal{L}$$

Theorem 3 (Gómez-Pérez et al. 2006) *The coherent MDD's of a circulant digraph $C_N(s_1, \dots, s_r)$ are "the same" as the (complements of) initial ideals of the binomial ideal I_C of the lattice* $\ker(R)$, *where $R : \mathbb{Z}^r \to \mathbb{Z}_N$ is (the natural extension of) the routing map.*

Our result in this respect is a more explicit characterization of the ideal I_C:

Theorem 4 *Let $\widetilde{\mathcal{L}}_G$ be the lattice associated to the circulant digraph $C_N(s_1, \dots, s_r)$:*

$$\widetilde{\mathcal{L}}_G := \langle N e_t, s_1 e_t - e_1, \dots, s_r e_t - e_r \rangle$$

Then the ideal I_C equals the elimination ideal of the variable t in the binomial ideal of the lattice. That is,

$$I_C = \operatorname{elim}\left(\langle t^N - 1, t^{s_1} - x_1, ..., t^{s_r} - x_r \rangle; t\right)$$

In particular, if $\mathcal{G}$ is a reduced Gröbner basis, with respect to the elimination ordering, of the ideal $I_{\widetilde{\mathcal{L}}_G}$ then the set of the leading monomials of the elements of $\mathcal{G}$, constitutes a minimal system of generators of (the complement of) an MDD.

3 Many minimum distance diagrams

In this section we consider a triple-loop network $C_N(s_1, s_2, s_3)$. Its associated lattice $\mathcal{L} = \ker R$ equals:

$$\mathcal{L} := \{(x, y, z) \in \mathbb{Z}^3 : s_1x + s_2y + s_3z \equiv 0 \mod N\}.$$

Choices in the construction of an MDD correspond to different path-vectors $\mathbf{a}, \mathbf{b} \in \mathbb{N}^r$ with $R(\mathbf{a}) = R(\mathbf{b})$ and $\|\mathbf{a}\|_1 = \|\mathbf{b}\|_1$. This makes us interested in the following *auxiliary lattice*

$$\mathcal{L}_0 := \mathcal{L} \cap \{(x, y, z) \in \mathbb{Z}^3 : x + y + z = 0\}.$$

See again Figure 2, where the lattice $\mathcal{L} := \{(x, y, z) \in \mathbb{Z}^3 : x + 2y + 3z \equiv 0 \mod 6\}$ gives an $\mathcal{L}_0$ generated by the vectors $(1, -2, 1)$ and (for example) $(3, 0, -3)$. The second vector is irrelevant in this example, but the first is the cause of non-uniqueness of the MDD.

We now state and proof or main result:

Theorem 5 *Let a and $q \in \mathbb{N}$. Let $N = 1 + q + q^2$ and let $k \geq N$ be such that $\gcd(k, N) = 1$. Then, the triple-loop network $C_{Nk}(a + k, a + qk, a + q^2k)$ has at least $\mathbf{3(q + 2)}$ coherent MDD's.*

Proof. We have to prove that there are $\mathbf{3(q + 2)}$ different initial ideals for the lattice $\mathcal{L}$. To simplify the exposition we will work out the case $a = 1$ and $q = 5$. The general case is similar.

The equation for the lattice $\mathcal{L}$

$$(1 + k)x + (1 + 5k)y + (1 + 25k)z \equiv 0 \mod 31,$$

together with $x + y + z = 0$, reduces to:

$$\mathcal{L}_0 = \begin{cases} y + 6z \equiv 0 \mod 31 \\ x + y + z = 0 \end{cases}$$

Consider the octant with sings $(-, +, +)$ intersected with $\mathcal{L}_0$. This is a semigroup generated by the points $(-31, 0, 31)$, $(-6, 1, 5)$, $(-11, 7, 4)$, $(-16, 13, 3)$, $(-21, 19, 2)$, $(-26, 25, 1)$ and $(-31, 31, 0)$ (in general, we obtain $q + 2$ generators).

Now, for each b_i in this minimal sistem of generators the normal to $\vec{Ob_i}$ in the direction exterior to the octant gives us a ray which separates two regions in the fan of coherent MDDs (the so-called Gröbner fan), since if $b_i = b_i^+ - b_i^-$, with $b_i^+ = \max\{b_{i_j}, 0\}$ and $b_i^- = \max\{-b_{i_j}, 0\}$, then:

- The exterior normal vector gives equal weight to the two monomials $x^{b_i^+}$ and $x^{b_i^-}$, which are in the same orbit modulo $\mathcal{L}$ (they represent the same monomial in the ideal).
- Since $x^{b_i^+}$ and $x^{b_i^-}$ have less degree than $31 < k$ and because $\mathcal{L}$ contains only vectors whose coordinates sum a number multiple of k there are no monomials with less degree equivalent to them modulo $\mathcal{L}$.

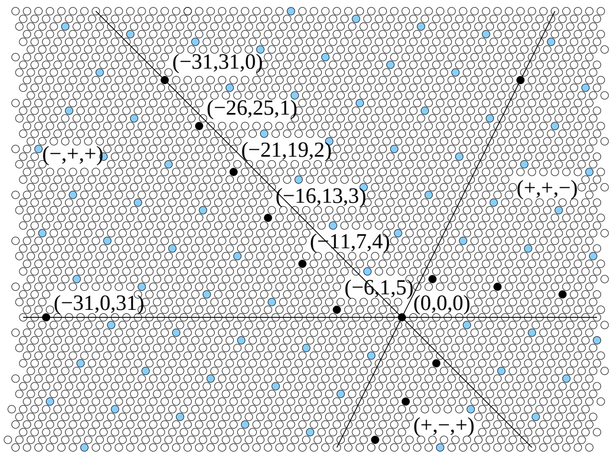


Figure 3: Lattice $\mathcal{L}_0$ of $C_{31k}(1 + k, 1 + 5k, 1 + 25k)$

- Between the monomials equivalent to $x^{b_i^+}$ and $x^{b_i^-}$ and with the same degree, the exterior normal vector gives less wieght to these one than to any one, since $\vec{Ob_i}$ is a segment in the convex hull of the points equivalent to b_i^+ and b_i^- modulo $\mathcal{L}$ that are in the octant intersected with the plane $x + y + z = 0$ considered at the beginning, then the exterior normal vector gives less degree to these two points than to any others.

So $x^{b_i^+}$ and $x^{b_i^-}$ are two monomials of the initial ideal obtained by perturbing the graded monomial ordering with a vector produced by a perturbation of the exterior normal vector in one side or in another, respectively. See Figure 4

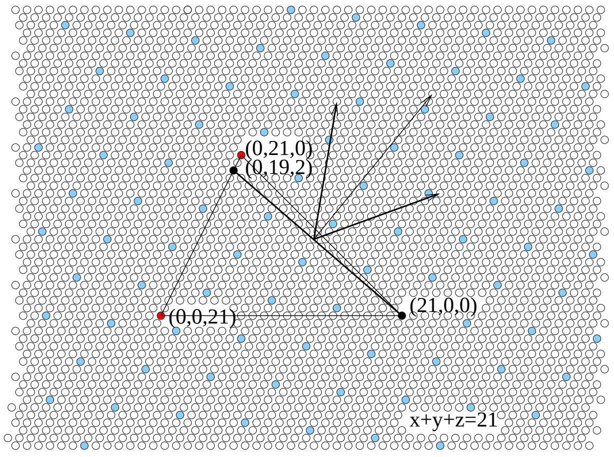


Figure 4: Perturbed exterior normal vector.

Repeating the same reasoning over the octants with signs $(+, -, +)$ and $(+, +, -)$, the numbers of coherent MDD's will be the sum of the number of elements in the three mininal systems of generators. So, there are $\mathbf{21} = 3 \times 7$ coherent MDD's associated with the circulant digraph $C_{31k}(1 + k, 1 + 5k, 1 + 25k)$ under the conditions given at the beginning. □

The crucial object used in this group is the systems of generators of the semigroups obtained intersecting the lattice $\mathcal{L}_0$ with different orthants. We recall that the (unique) minimal system of generators of a semigroup is called its *Hilbert basis*. It seems logical to think that the number of elements in the Hilbert bases of the three semigroups is going to equal the number of MDD's, but we have a counterexample of this assertion:

Let $\mathcal{L}_0$ be the lattice generated by $(-4,4,0)$ and $(-4,3,1)$. (See Figure 5)

The Hilbert basis in each cone is shown in Table 1, but the only elements susceptible to generate rays in the fan of MDD's are: $(-4,4,0)$, $(4,0,-4)$, $(0,1,-1)$ and $(0,-1,1)$

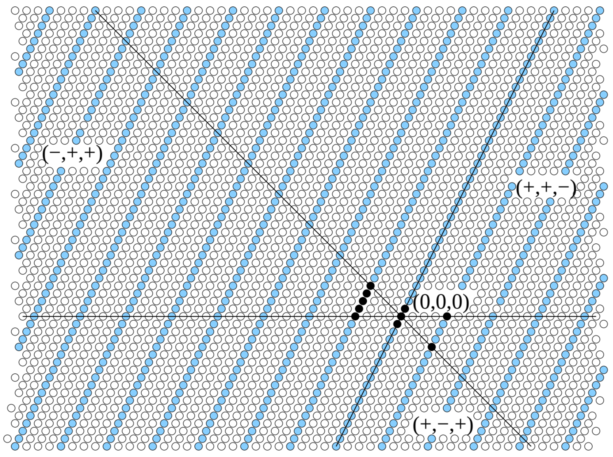


Figure 5: Lattice $\mathcal{L}_0$ generated by $(-4,4,0)$ and $(-4,3,1)$

Cone	Hilbert Basis
(+,+,-)	(4,0,-4), (0,1,-1)
(-,+,+)	(-4,4,0), (-4,3,1), (-4,2,2), (-4,1,3), (-4,0,4)
(+,-,+)	(0,-1,1), (4,-4,0)

Table 1: Hilbert Bases for the auxiliary lattice $\mathcal{L}_0$ generated by $(-4,4,0)$ and $(-4,3,1)$.

The circulant digraph which corresponds with this lattice, $\mathcal{L}_0$, is $C_{16}(13,5,5)$ and the four MDD's associated to it are shown in Figure 5.

In fact, we have a rult to prescribe which elements in the Hilbert bases are susceptible of producing rays in the fan of MDD's:

Let $v_1,\ldots,v_k$ be the vectors in the Hilbert basis of a cone, *from left to right* (see figure 5). Let $v_i, v_{i+1},\ldots,v_j$ be the vectors in the basis with the minimum norm L_1. Then:

- The *left exterior normal vector* of each of the vectors $v_1,\ldots,v_i$ is susceptible to produce a ray in the fan of MDD's.
- The *right exterior normal vector* of each of the vectors $v_j,\ldots,v_k$ is susceptible to produce a ray in the fan of MDD's.

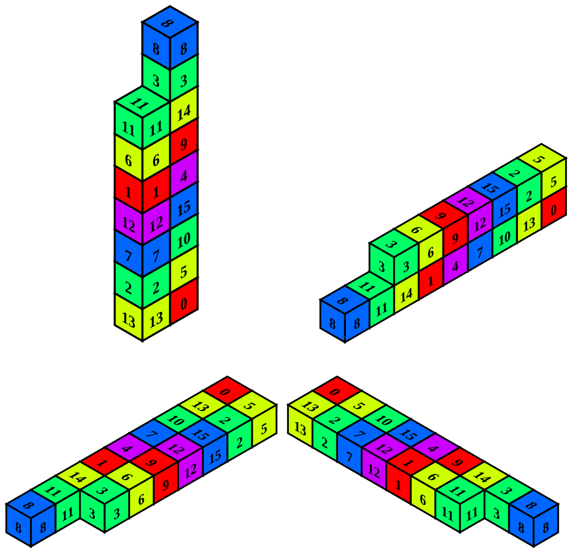


Figure 6: Coherent MDD's diagrams associated with $C_{16}(13,5,5)$

References

[1] F. Aguiló, M.A. Fiol, C. García, Triple-loop networks with small transmission delay. *Discrete Math.* ***167/168*** (1997) 3–16.

[2] C. Chen, F.K. Hwang, J.S. Lee, S.J. Shih, The existence of hyper-L triple-loop networks. *Discrete Math.* ***268*** (2003) 287–291.

[3] D. Gómez, J. Gutierrez, A. Ibeas, Cayley Digraphs of Finite Cyclic Groups and Monomial Ideals *Preprint.* University of Cantabria (2006).

[4] B. Sturmfels, R. Weismantel and G. M. Ziegler, Gröbner Bases of Lattices, Corner Polyhedra, and Integer Programming *Contribution to Algebra and Geometry.* ***Vol. 36*** (1995), No 2, 281–298.

[5] C.K. Wong, D. Coppersmith, A combinatorial problem related to multimodule organizations. *J. Assoc. Comput. Mach.* ***21*** (1974) 392–402.

Online conflict-free coloring for geometric hypergraphs

Amotz Bar-Noy[*] Panagiotis Cheilaris[†] Svetlana Olonetsky[‡] Shakhar Smorodinsky[§]

Abstract

(i) We provide a framework for online conflict-free coloring (CF-coloring) any hypergraph. We use this framework to obtain an efficient randomized online algorithm for CF-coloring any k-degenerate hypergraph. Our algorithm uses $O(k \log n)$ colors with high probability and this bound is asymptotically optimal for any constant k. Moreover, our algorithm uses $O(k \log k \log n)$ random bits with high probability. We obtain asymptotically optimal randomized algorithms for online CF-coloring some hypergraphs that arise in geometry and model an important version of the frequency assignment task for cellular networks. Our algorithms use exponentially fewer random bits compared to previous results for these special cases ($O(\log n)$ bits instead of $\Theta(n \log n)$ bits).
(ii) We initiate the study of deterministic online CF-coloring with recoloring. The goal is to use few colors, but also resort to recoloring as little as possible. We provide an algorithm for CF-coloring with respect to halfplanes using $O(\log n)$ colors and $O(n)$ recolorings.

1 Introduction

A *hypergraph* is a pair $(V, \mathcal{E})$, where V is a finite set and $\mathcal{E} \subset 2^V$. The set V is called the *ground set* or the *vertex set* and the elements of $\mathcal{E}$ are called *hyperedges*. A *proper* k-coloring of a hypergraph $H = (V, \mathcal{E})$, for some positive integer k, is a function $C : V \to \{1, 2, \ldots, k\}$ such that no $S \in \mathcal{E}$ with $|S| \geq 2$ is monochromatic. A *conflict-free* coloring (CF-coloring) of H is a coloring of V with the further restriction that for any hyperedge $S \in \mathcal{E}$ there exists a vertex $v \in S$ with a unique color (i.e., no other vertex of S has the same color as v).

The study of conflict-free colorings was originated in the work of Even et al. [5] and Smorodinsky [9] who were motivated by the problem of frequency assignment in cellular networks. Specifically, cellular networks are heterogeneous networks with two different types of nodes: *base stations* (that act as servers) and *clients*. Base stations are interconnected by an external fixed backbone network whereas clients are connected only to base stations. Connections between clients and base stations are implemented by radio links. Fixed frequencies are assigned to base stations to enable links to clients. Clients continuously scan frequencies in search of a base station with good reception. The fundamental problem of frequency assignment in such cellular networks is to assign frequencies to base stations so that every client, located within the receiving range of at least one station, can be served by some base station, in the sense that the client is located within the range of the station and no other station within its reception range has the same frequency (such a station would be in "conflict" with the given station due to mutual interference). The goal is to minimize the number of assigned frequencies ("colors") since the frequency spectrum is limited and costly. In addition to the practical motivation, this new coloring model has drawn much attention of researchers through its own theoretical interest and such colorings have been the focus of several recent papers (see, e.g., [2, 3, 4, 7, 8, 10]). To capture a dynamic scenario where antennas can be added to the network, Fiat et al. [4] initiated the study of online CF-coloring of hypergraphs.

In this paper, we study the most general form of online CF-coloring applied to arbitrary hypergraphs. Suppose the vertices of an underlying hypergraph $H = (V, \mathcal{E})$ are given online by an adversary. Namely, at every given time step t, a new vertex $v_t \in V$ is given and the algorithm must assign v_t a color such that the coloring is a valid conflict-free coloring of the hypergraph that is induced by the vertices $V_t = \{v_1, \ldots, v_t\}$ (see the exact definition in section 2). Once v_t is assigned a color, that color cannot be changed in the future. The goal is to find an algorithm that minimizes the maximum total number of colors used (where the maximum is taken over all permutations of the set V).

We present a general framework for online CF-coloring any hypergraph. Interestingly, this framework is a generalization of some known coloring algorithms. For example the Unique-Max Algorithm of [4] can be described as a special case of our framework. Also, when the underlying hypergraph is a

[*]Brooklyn College, City University of New York, amotz@sci.brooklyn.cuny.edu. Supported by the CUNY Collaborative Incentive Research Grants Program Round 11 (2004–2006).

[†]City University of New York and National Technical University of Athens, philaris@sci.brooklyn.cuny.edu. Supported by the European Social Fund (75%) and National Resources (25%), under the program EPEAEK II, 'Heraclitus'.

[‡]Tel-Aviv University, olonetsk@post.tau.ac.il

[§]Courant Institute for Mathematical Sciences, New York University, shakhar@cims.nyu.edu. Supported by the NSF Mathematical Sciences Postdoctoral Fellowship award 0402492.

simple graph then the First-Fit online algorithm is another special case of our framework.

Based on this framework, we introduce a randomized algorithm and show that the maximum number of colors used is a function of the 'degeneracy' of the hypergraph. We define the notion of a k-degenerate hypergraph as a generalization of the same notion for simple graphs. Specifically we show that if the hypergraph is k-degenerate, then our algorithm uses $O(k \log n)$ colors with high probability. This is asymptotically tight for any constant k.

As demonstrated in [4], the problem of online CF-coloring the very special hypergraph induced by points on the real line with respect to intervals is highly non-trivial. Kaplan and Sharir [7] studied the special hypergraph induced by points in the plane with respect to halfplanes and unit discs and obtained a randomized online CF-coloring with $O(\log^3 n)$ colors with high probability. Recently, the bound $\Theta(\log n)$ just for these two special cases was obtained independently by Chen [3]. Our algorithm is more general and uses only $\Theta(\log n)$ colors; an interesting evidence to our algorithm being fundamentally different from the ones in [3, 4, 7], when used for the special case of hypergraphs that arise in geometry, is that it uses exponentially fewer random bits. The algorithms of [3, 7] use $\Theta(n \log n)$ random coin flips and our algorithm uses $O(\log n)$ random coin flips. Another interesting corollary of our result is that we obtain a randomized online coloring for k-inductive graphs with $O(k \log n)$ colors with high probability. This case was studied by Irani [6] who showed that the first-fit greedy algorithm achieves the same bound deterministically.

Deterministic online CF-coloring with recoloring: We initiate the study of online CF-coloring where at each step, in addition to the assignment of a color to the newly inserted point, we allow some recoloring of other points. The bi-criteria goal is to minimize the total number of recoloring done by the algorithm and the total number of colors used by the algorithm. We provide an algorithm for CF-coloring with respect to halfplanes using $O(\log n)$ colors and $O(n)$ recolorings.

2 Preliminaries

Definition 1 *Let $H = (V, \mathcal{E})$ be a hypergraph. For a subset $V' \subset V$ let $H(V')$ be the hypergraph $(V', \mathcal{E}')$ where $\mathcal{E}' = \{e \cap V' | e \in \mathcal{E}\}$. $H(V')$ is called the* induced hypergraph *on V'.*

Definition 2 *For a hypergraph $H = (V, \mathcal{E})$, the* Delaunay graph *$G(H)$ is the simple graph $G = (V, E)$ where the edge set E is defined as $E = \{(x, y) \mid \{x, y\} \in \mathcal{E}\}$ (i.e., G is the graph on the vertex set V whose edges consist of all hyperedges in H of cardinality two).*

Definition 3 *A simple graph $G = (V, E)$ is called k-degenerate (or k-inductive) for some positive integer k, if every (vertex-induced) subgraph of G has a vertex of degree at most k.*

Definition 4 *Let $k > 0$ be a fixed integer and let $H = (V, \mathcal{E})$ be a hypergraph on n vertices. Fix a subset $V' \subset V$. For a permutation π of V' such that $V' = \{v_1, ..., v_i\}$ (where $i = |V'|$) let $C_\pi(V') = \sum_{j=1}^{i} d(v_j)$, where $d(v_j) = |\{l < j | (v_j, v_l) \in G(H(\{v_1, ..., v_j\}))\}|$, that is, $d(v_j)$ is the number of neighbors of v_j in the Delaunay graph of the hypergraph induced by $\{v_1, ..., v_j\}$. Assume that $\forall V' \subset V$ and for all permutations $\pi \in S_{|V'|}$ we have $C_\pi(V') \leq k\,|V'|$. Then we say that H is* k-degenerate.

It is not difficult to see that our definition of a k-degenerate hypergraph is a generalization of that of a k-degenerate graph.

3 An online CF-coloring framework

Let $H = (V, E)$ be any hypergraph. Our goal is to define a framework that colors the vertices V in an online fashion. That is, the vertices of V are revealed by an adversary one at a time. At each time step t, the algorithm must assign a color to the newly revealed vertex v_t. This color cannot be changed in the future. The coloring has to be conflict-free for all the induced hypergraphs $H(V_t)$ $t = 1, \ldots, n$, where $V_t \subset V$ is the set of vertices revealed by time t.

For a fixed positive integer h, let $A = \{a_1, \ldots, a_h\}$ be a set of h auxiliary colors (not to be confused with the set of 'real' colors used for the CF-coloring: $\{1, 2, \ldots\}$). Let $f : \mathbb{N} \to A$ be some fixed function. We now define the framework that depends on the choice of the function f and the parameter h.

A table (to be updated online) is maintained where each entry i at time t is associated with a subset $V_t^i \subset V_t$ in addition to an auxiliary proper coloring of $H(V_t^i)$ with at most h colors. We say that $f(i)$ is the color that represents entry i in the table. At the beginning all entries of the table are empty. Suppose all entries of the table are updated until time $t - 1$ and let v_t be the vertex revealed by the adversary at time t. The framework first checks if an auxiliary color can be assigned to v_t such that the auxiliary coloring of V_{t-1}^1 together with the color of v_t is a proper coloring of $H(V_{t-1}^1 \cup \{v_t\})$. Any (proper) coloring procedure can be used by the framework. For example a first-fit greedy one in which all colors in the order $a_1, \ldots, a_h$ are checked until one is found. If such a color cannot be found for v_t, then entry 1 is left with no changes and the process continues to the next entry. If however, such a color can be assigned, then v_t is added to the set V_{t-1}^1. Let c denote such an auxiliary color assigned to v_t. If this color is the same as $f(1)$

(the auxiliary color that is associated with entry 1), then the final color in the online CF-coloring of v_t is 1 and the updating process for the t-th vertex stops. Otherwise, if an auxiliary color cannot be found or if the assigned auxiliary color is not the same as the color associated with this entry, the updating process continues to the next entry. The updating process stops at the first entry i for which v_t is both added to V_t^i and the auxiliary color assigned to v_t is the same as $f(i)$. The color of v_t in the final conflict-free coloring is then set to i.

It is possible that v_t never gets a final color. In this case we say that the framework does not halt. However, termination can be guaranteed by imposing some restrictions on the auxiliary coloring method and the choice of the function f. For example, if first-fit is used for the auxiliary colorings at any entry and if f is the constant function $f(i) = a_1$, for all i, then the framework is guaranteed to halt for any time t. In section 4 we derive a randomized online algorithm based on this framework. It is not difficult to prove that the algorithm halts after a "small" number of entries with high probability (w.h.p.).

Lemma 1 *If the above framework halts for any vertex v_t then it produces a valid online CF-coloring.*

4 An online randomized CF-coloring algorithm

There is a randomized online CF-coloring in the oblivious adversary model that always produces a valid coloring and the number of colors used is related to the degeneracy of the underlying hypergraph in a manner described in theorem 2. Proofs will be included in a longer version of this paper.

Theorem 2 *Let $H = (V, \mathcal{E})$ be a k-degenerate hypergraph on n vertices. Then there exists a randomized online CF-coloring for H which uses at most $O(\log_{1+\frac{1}{4k+1}} n) = O(k \log n)$ colors with high probability.*

The algorithm is based on the framework of section 3. In order to define the algorithm, we need to choose: (a) the set of auxiliary colors of each entry, (b) the algorithm we use for the auxiliary coloring at each entry, and (c) the function f. We use: (a) auxiliary colors in $A = \{a_1, \ldots, a_{2k+1}\}$, (b) a first-fit algorithm for the auxiliary coloring, and (c) for each entry i, the representing color $f(i)$ is chosen uniformly at random from A. Our assumption on the hypergraph H (being k-degenerate) implies that at least half of the vertices up to time t that 'reached' entry i (but not necessarily added to entry i), and we denote by $X_{\geq i}^t$, have been actually given some auxiliary color at entry i (that is, $|V_t^i| \geq \frac{1}{2} |X_{\geq i}^t|$). This is easily implied by the fact that at least half of those vertices v_t have at most $2k$ neighbors in the Delaunay graph of the hypergraph induced by $X_{\geq i}^{t-1}$ (since the sum of these quantities is at most $k |X_{\geq i}^t|$ and since $V_t^i \subset X_{\geq i}^t$). Therefore since we have $2k+1$ colors available, there is always a free color to assign to such a vertex. The following lemma shows that if we use one of these 'free' colors then the updated coloring is indeed a proper coloring of the corresponding induced hypergraph as well.

Lemma 3 *Let $H = (V, \mathcal{E})$ be a k-degenerate hypergraph and let V_t^j be the subset of V at time t and at level j as produced by the above algorithm. Then for any j and t if v_t is assigned a color distinct from all its neighbors in the Delaunay graph $G(H(V_t^j))$ then this color together with the colors assigned to the vertices V_{t-1}^j is also a proper coloring of the hypergraph $H(V_t^j)$.*

Lemma 4 *Let $H = (V, \mathcal{E})$ be a hypergraph and let C be a coloring produced by the above algorithm on an online input $V = \{v_t\}$ for $t = 1, \ldots, n$. Let X_i (respectively $X_{\geq i}$) denote the random variable counting the number of points of V that were assigned a final color at entry i (respectively a final color at some entry $\geq i$). Let $\mathbf{E}_i = \mathbf{E}[X_i]$ and $\mathbf{E}_{\geq i} = \mathbf{E}[X_{\geq i}]$ (note that $X_{\geq i+1} = X_{\geq i} - X_i$). Then:*

$$\mathbf{E}_{\geq i} \leq \left(\frac{4k+1}{4k+2}\right)^{i-1} n$$

Lemma 5 *The expected number of colors used by the above algorithm is at most $\log_{\frac{4k+2}{4k+1}} n + 1$.*

Remark: In the above description of the algorithm, all the random bits are chosen in advance (by deciding the values of the function f in advance). However, one can be more efficient and calculate the entry $f(i)$ only at the first time we need to update entry i, for any i. Since at each entry we need to use $O(\log k)$ random bits and we showed that the number of entries used is $O(k \log n)$ w.h.p then the total number of random bits used by our algorithm is $O(k \log k \log n)$ w.h.p.

5 Application to Geometry

Our randomized algorithm has applications to CF colorings of certain geometric hypergraphs studied in [3, 4, 7]. We obtain the same asymptotic result as in [3], but with better constants of proportionality and much fewer random bits. An algorithm for intervals is given in [1]. When the hypergraph H is induced by points in the plane intersected by halfplanes or unit disks, we obtain online randomized algorithms that use $O(\log n)$ colors w.h.p. We summarize it as follows:

Lemma 6 *Let V be a finite set of n points in the plane and let $\mathcal{E}$ be all subsets of V that can be obtained by intersecting V with a halfplane. Then the hypergraph $H = (V, \mathcal{E})$ is 4-degenerate.*

Proof. The proof uses a few geometric lemmas. Details are omitted. □

Corollary 7 *Let H be the hypergraph as in lemma 6. Then the expected number of colors used by our randomized online CF-coloring applied to H is at most $\log_{\frac{18}{17}} n + 1$. Also the actual number of colors used is $O(\log_{\frac{18}{17}} n)$ with high probability. The number of random bits is $O(\log n)$ with high probability*

Proof. The proof follows immediately from lemma 6, lemma 5 and theorem 2. □

Proposition 8 *Let V be a finite set of n points in the plane and let $\mathcal{E}$ be all subsets of V that can be obtained by intersecting V with a unit disc. Then there exists a randomized online algorithm for CF-coloring H which uses $O(\log n)$ colors and $O(\log n)$ random bits with high probability.*

Proof. By a technique of Kaplan and Sharir [7] and Corollary 7. □

6 Deterministic online CF-coloring with recoloring

In this section we describe a deterministic algorithm for online CF-coloring points with respect to halfplanes that uses $O(\log n)$ colors and recolors $O(n)$ points. At every time instance t, the algorithm maintains the following invariant (V_t is the set of points that have appeared): All points (strictly) inside the convex hull of V_t are colored with the same special color, say '$\star$'. The set of points on the convex hull of V_t, denoted by $\mathrm{CH}(V_t)$, are colored with another set of colors, such that every set of consecutive points on the convex hull has a point with a unique color. The number of colors used in $\mathrm{CH}(V_t)$ must be logarithmic on t. It is not difficult to see that every subset of points of V_t induced by a halfplane contains a set of consecutive points, and thus the whole coloring is conflict-free. We describe how the algorithm maintains the above invariant. A new point v_{t+1} that appears at time $t + 1$ is colored as follows: If it is inside the convex hull of V_t, then it gets color '$\star$'. If however it is in $\mathrm{CH}(V_{t+1})$, it might force some points that where in $\mathrm{CH}(V_t)$ to get inside the convex hull of V_{t+1}. In order to maintain the invariant, if there exist such points, they are recolored to '$\star$', and v_{t+1} is colored greedily, so that the coloring of $\mathrm{CH}(V_{t+1})$ is conflict-free (it can be proved that no new color is introduced). If, on the other hand, no points of $\mathrm{CH}(V_t)$ are forced into the convex hull, then $v_{t+1} \in \mathrm{CH}(V_{t+1})$ is colored with the algorithm that is used for intervals, given in [1], with a slight adaptation to address the closed curve nature of the convex hull. In that last case, in order to maintain logarithmic number of colors on t, one recoloring of a point in $\mathrm{CH}(V_{t+1})$ might be needed. The number of recolorings is guaranteed to be $O(n)$, because for every insertion, at most one recoloring happens on the new convex hull, and every point colored with '$\star$' stays with that color, because the convex hull never shrinks.

References

[1] A. Bar-Noy, P. Cheilaris, S. Olonetsky, and S. Smorodinsky. Weakening the online adversary just enough to get optimal conflict-free colorings for intervals. Manuscript, 2006.

[2] A. Bar-Noy, P. Cheilaris, and S. Smorodinsky. Conflict-free coloring for intervals: from offline to online. In *Proceedings of the 18th annual ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 128–137, 2006.

[3] K. Chen. How to play a coloring game against a color-blind adversary. In *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (SoCG)*, pages 44–51, 2006.

[4] K. Chen, A. Fiat, H. Kaplan, M. Levy, J. Matoušek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, U. Wagner, and E. Welzl. Online conflict-free coloring for intervals. *SIAM Journal on Computing*, 36(5):956–973, 2006.

[5] G. Even, Z. Lotker, D. Ron, and S. Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33:94–136, 2003.

[6] S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.

[7] H. Kaplan and M. Sharir. Online CF coloring for halfplanes, congruent disks, and axis-parallel rectangles. Manuscript, 2004.

[8] J. Pach and G. Tóth. Conflict free colorings. In *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*, pages 665–671. Springer Verlag, 2003.

[9] S. Smorodinsky. *Combinatorial Problems in Computational Geometry.* PhD thesis, School of Computer Science, Tel-Aviv University, 2003.

[10] S. Smorodinsky. On the chromatic number of some geometric hypergraphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

On the Chromatic Numbers of Some Flip Graphs

Ruy Fabila-Monroy[†] David Flores Peñaloza [†] Clemens Huemer [‡] Ferran Hurtado [‡]
Jorge Urrutia [†] David R. Wood [‡]

Abstract

In this paper we study the chromatic number of the following four flip graphs: A graph on the perfect matchings of the complete graph on $2n$ vertices and three graphs on the triangulations, Hamiltonian geometric non-crossing paths, and triangles respectively of a point set in convex position in the plane. We give tight bounds for the latter two cases and upper bounds for the first two.

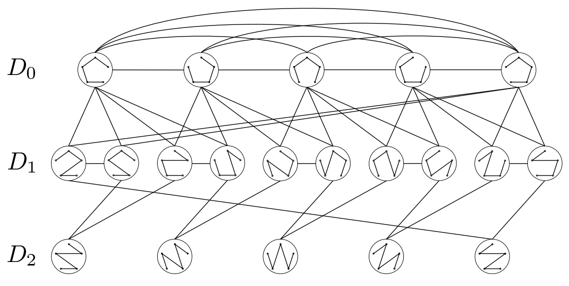


Figure 1: G_5

1 Introduction

Given a class $\mathcal{C}$ of combinatorial objects of a given kind and a transformation (flip) between these objects, a flip graph is defined as the graph whose vertex set is $\mathcal{C}$, where two vertices are adjacent whenever they differ by a flip. Flip graphs have received considerable attention in the past. Properties such as Hamiltonicity, connectivity and diameter have been widely studied [4, 6, 12, 15]. This interest is very likely due to the practical applications of these properties. For example, Hamiltonicity allows for rapid generation of the given combinatorial objects. We refer the interested reader to the survey [1].

The chromatic number $\chi(G)$ of a graph is the smallest integer such that it is possible to assign to each vertex of G an integer $i \leq \chi(G)$ such that adjacent vertices of G receive different integers. The chromatic number of flip graphs has received little attention, with only a few papers concentrating on this parameter (see for example [5]). In this paper we study the chromatic number of a flip graph on the perfect matchings of the complete graph and three flip graphs for sets of points in convex position, that is, they form the set of vertices of a convex polygon on the plane.

For flip graphs on convex point sets, we determine the exact chromatic number for geometric Hamiltonian paths (Section 2), we define a certain flip graph on its triangles of and determine its chromatic number up to a constant multiplicative factor (Section 3), and we give an upper bound on the chromatic number for triangulations (Section 5). We also consider a flip graph on the the perfect matchings of the complete graph on $2n$ vertices and give an upper bound on its chromatic number (Section 4). It should be stressed that in the case of matchings, no geometry is considered. We conclude with some open problems in Section 6.

Throughout the rest of the paper S will denote a set of n points in convex position in the plane.

2 Geometric Non-Crossing Hamiltonian Paths

Let G_n be the graph whose vertex set is the set of all non-crossing geometric paths with vertex set S. Two paths Γ_1 and Γ_2 in $V(G_n)$ are adjacent if and only if there exist edges e in Γ_1 and f in Γ_2 such that $\Gamma_2 = \Gamma_1 - e + f$. We say that Γ_2 is obtained from Γ_1 by *flipping* e and f. We point out that e and f may intersect. Note that Rivera-Campo and Urrutia-Galicia [14, 17] proved that G_n is Hamiltonian. We determine $\chi(G_n)$.

Theorem 1 *$\chi(G_n) = n$ for $n \geq 3$.*

Proof. Since $G_3 \simeq K_3$, assume $n \geq 4$. For $i = 0, \ldots, n-3$, let D_i be the set of paths in $V(G_n)$ that contain exactly i non-convex hull edges (see Figure 1). Note that the set D_0 consists of n paths, all of whose edges are on the convex hull of S. Each element of D_0 is obtained by removing one edge of the convex hull

[†]Instituto de Matemáticas, Universidad Nacional Autónoma de México (ruy@ciencias.unam.mx, colegadavid@gmail.com, urrutia@math.unam.mx). Supported by CONACYT of Mexico, Proyecto SEP-2004-Co1-45876, and PAPIIT (UNAM), Proyecto IN110802.

[‡]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain {(huemer.clemens,ferran.hurtado,david.wood}@upc.edu). Research of C. Huemer and F. Hurtado supported by projects MEC MTM2006-01267 and Gen. Cat. 2005SGR00692. Research of D. Wood supported by a Marie Curie Fellowship of the European Community under contract 023865.

of S. Clearly any two elements of D_0 are adjacent in G_n, and D_0 thus induces a complete subgraph of G_n. This proves that $\chi(G_n) \geq n$.

We now give a method to obtain an n-coloring of G_n. Note that each time we flip an edge of a path in D_i, we obtain a path in either D_{i-1}, D_i or D_{i+1}. Since D_0 induces a clique of size n in G_n, in any n-coloring of G_n we assign a different color to each element in D_0. We now show how to extend an n-coloring of the elements of D_0 to an n-coloring of G_n. Observe that every path Γ_1 in D_1 is adjacent to exactly two paths in D_0. Furthermore if Γ_1 and Γ_2 lie in D_1, then both are adjacent to the same two paths of D_0, or there is no path in D_0 adjacent to both of them. Since we are assuming $n \geq 4$, for each pair of adjacent paths in D_1 there are at least two colors we can assign to them, different from the colors assigned to their neighbors in D_0. Thus we can extend our coloring of D_0 to one of $D_0 \cup D_1$. Observe next that for for $i \geq 2$, there is no pair of adjacent paths in D_i, and that every path in D_i is adjacent to exactly two elements in D_{i-1}. Thus any n-coloring of $D_0 \cup \ldots \cup D_{i-1}$ can be extended to an n-coloring of $D_0 \cup \ldots \cup D_i$, $i = 2, \ldots, n-3$. Therefore $\chi(G_n) = n$. □

3 Triangle Graph

We introduce a new flip graph $G_\triangle(n)$, whose vertex set is the set of triangles with endpoints in S, two of which are adjacent if they share an edge and their interiors are disjoint. Assume that the elements of S are labeled $\{0, \ldots, n-1\}$ clockwise along the convex hull of S, and let $\triangle(i, j, k)$ denote the triangle with vertex set $\{i, j, k\}$.

Lemma 2 $\chi(G_\triangle(n)) \geq \log_2(n-1)$.

Proof. Let H be the subgraph of $G_\triangle(n)$ induced by all triangles containing 0 as a vertex. Suppose that H has a coloring with k colors. Let A_j be the set of colors assigned to the triangles $\triangle(0, i, j)$ with $i < j$. Observe that for $r \neq s$, $A_r \neq A_s$. This follows from the fact that if $r < s$, $\triangle(0, r, s)$ is adjacent to every triangle $\triangle(0, k, r)$ with $k < r$; thus the color assigned to $\triangle(0, r, s)$ is in A_s but not in A_r. The number of possible color sets must be at least $n-1$. Therefore $2^k \geq n-1$, $k \geq \log_2(n-1)$ and $\chi(G_\triangle(n)) \geq \log_2(n-1)$. □

The subgraph H of $G_\triangle$ considered in the proof of Lemma 2 is known as the *shift graph*, and its chromatic number is well known [16].

Given two graphs G and H, a homomorphism from G to H is a mapping from the vertex set of G to the vertex set of H such that adjacent vertices of G are mapped to adjacent vertices in H. It is well known and straightforward to see that if there is a homomorphism from G to H, then the chromatic number of G is less than or equal to the chromatic number of H.

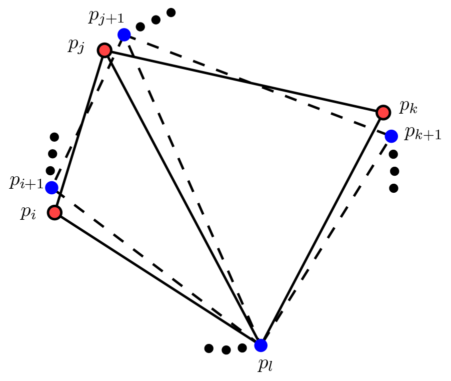


Figure 2: The homomorphism between G and $G_\triangle(m)$

Lemma 3 *Let G be the subgraph of $G_\triangle(2m)$ induced by all the triangles without edges on the convex hull of S. Then $\chi(G) \leq \chi(G_\triangle(m))$.*

Proof. Color the vertices of S red and blue such that no two consecutive vertices have the same color. Note that the subgraph H of G induced by the blue points is isomorphic to $G_\triangle(m)$. We now map every triangle t in $V(G)$ to a triangle such that all its vertices are blue as follows: if i is a red vertex of t, substitute it by the blue vertex $i+1$, addition taken $\mod n$. Observe that triangles whose vertices were already all blue are mapped onto themselves. Since adjacent triangles in G are mapped to adjacent triangles in H, this mapping induces a homomorphism from G to H (see Figure 2). The result follows. □

Lemma 4 $\chi(G_\triangle(2m)) \leq \chi(G_\triangle(m)) + 3$.

Proof. We use a similar technique to that used in Theorem 1 of Section 2. Let G' be the subgraph of $G_\triangle(2m)$ induced by all those triangles of S that have at least one edge on the convex hull of S. Let $\tau = \triangle(i, i+1, j)$ be any such triangle. $S - \{j, i, i+1\}$ can be partitioned into two maximal subsets of consecutive points, namely the points from $i+1$ to j and the points from j to i, which we denote by l_τ and r_τ respectively. We define the "order" of τ to be $\min\{|l_\tau|, |r_\tau|\}$. For $i = 0, \ldots \lceil \frac{n-2}{2} \rceil$, let D_i be the subset of $V(G')$ of all triangles of order i. Note that the subgraph of G' induced by $D_{\lceil \frac{n-2}{2} \rceil}$ has maximum degree 2 and therefore chromatic number at most 3. Note that in general every vertex of D_i is only adjacent to at most 2 vertices in $D_{i+1} \cup \ldots D_{\lceil \frac{n-2}{2} \rceil}$. Thus any 3-coloring of $D_{i+1} \cup \ldots D_{\lceil \frac{n-2}{2} \rceil}$ can be extended to a coloring of $D_i \cup D_{i+1} \cup \ldots D_{\lceil \frac{n-2}{2} \rceil}$. Therefore $\chi(G') = 3$. By Lemma 3, we can color G with $\chi(G_\triangle(m))$ colors and G' with four different colors. This produces a coloring of $G_\triangle(2m)$ with $\chi(G_\triangle(m)) + 3$ colors. □

We now have:

Theorem 5 $\log_2(n-1) \leq \chi(G_\triangle(n)) \leq 3\lceil\log_2(n)\rceil - 6$.

Proof. Let $F(k) = \chi(G_\triangle(2^k))$. By Lemma 4, $F(k) \leq F(k-1)+3$. Observe that $\chi(G_\triangle(4)) = 2$. Thus $F(k) \leq 3k-6$ and therefore $\chi(G_\triangle(n)) \leq 3\log_2(n) - 6$.

For $n \neq 2^k$, let m be the smallest power of 2 greater than n. Since in general $G_\triangle(n)$ is a subgraph of $G_\triangle(n+1)$, we can color the vertices of $G_\triangle(n)$ with $\chi(G_\triangle(m)) \leq 3\log_2(m) - 6 = 3\lceil\log_2(n)\rceil - 6$ colors. □

4 Perfect Matchings of K_{2n}

Given any graph G, we define $\mathcal{M}(G)$ to be the graph whose vertex set is the set of perfect matchings of G, where two vertices of $\mathcal{M}(G)$ are adjacent whenever the symmetric difference of the corresponding perfect matchings is a cycle of length 4. $\mathcal{M}(G)$ is known as the *flip graph* of the perfect matchings of G.

We now give an upper bound on $\chi(\mathcal{M}(K_{2n}))$. To do so, we need the fact that $\chi(\mathcal{M}(K_{n,n})) = 2$. In [7], the flip graph of the non-crossing geometric matchings of a set of $2n$ points in convex position is shown to be bipartite. This graph is actually a subgraph of $\mathcal{M}(K_{n,n})$. Using the same arguments as in [7] it is possible to show that $\mathcal{M}(K_{n,n})$ is bipartite. For details, the interested reader can see [7].

Theorem 6 $\chi(\mathcal{M}(K_{2n})) \leq 4n-4$ *for* $n \geq 2$.

Proof. Label the vertices of $V(K_{2n}) = \{1,\ldots,2n\}$. For every perfect matching M of K_{2n}, let $U_M = \{i \in V(K_{2n}) | (i,j) \in M \text{ and } i > j\}$ and $D_M = \{i \in V(K_{2n}) | (i,j) \in M \text{ and } i < j\}$. Assign to every partition U_M, D_M of $V(K_{2n})$ given by a matching M the number $i_M = \sum_{i\in D_M} i \mod 2n-2$. Given two sets U, D, let $M_{U,D}$ be the set of matchings M such that $U_M = U$ and $D_M = D$. The subgraph $H_{U,D}$ of $\mathcal{M}(K_{2n})$ induced by $M_{U,D}$ is a subgraph of $\mathcal{M}(K_{n,n})$, and is thus 2-colorable. Color the vertices of H with colors i_M and i'_M.

We show now that if two matchings M and M' differ by a flip, they receive different colors.

Two cases arise: $U_M = U_{M'}$ and $D_M = D_{M'}$ or for some $i<j$, $U_M = U_{M'} - i + j$ and $D_M = D_{M'} - j + i$. In the first case, M and M' belong to H_{U_M,D_M} and thus receive different colors.

In the second case, $U_M \neq U_{M'} - i + j$, and the colors assigned to M and M' are different. Thus we obtain a coloring of the vertices of $\chi(\mathcal{M}(K_{2n}))$ with $2(2n-2) = 4n-4$ colors. □

5 Triangulations of a Convex Polygon

Finally, we consider a flip graph on the triangulations of S. Let $G_T(n)$ be the graph whose vertex set is the set of triangulations of S, where two triangulations $\mathcal{T}_1$ and $\mathcal{T}_2$ are adjacent in $G_T(n)$ whenever they differ by one edge flip; that is, there exist edges $e \in \mathcal{T}_1$ and $f \in \mathcal{T}_2$ such that $\mathcal{T}_2 = \mathcal{T}_1 - e + f$ (see Figure 3).

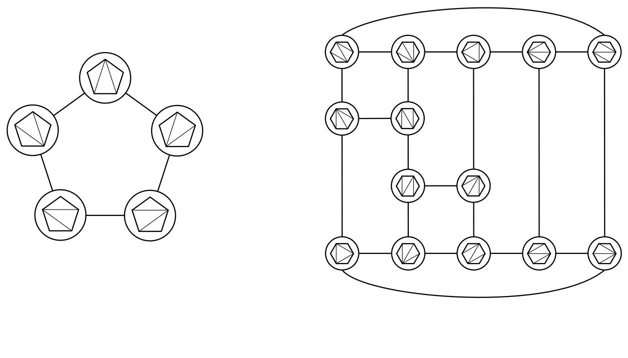


Figure 3: $G_T(5)$ and $G_T(6)$

Much is known about $G_T(n)$. This is probably due to the fact that there is a bijection between triangulations of the n-gon and binary trees with $n-2$ nodes. A flip in a triangulation corresponds to a rotation in its corresponding binary tree.

Lucas [12] proved that $G_T(n)$ is Hamiltonian. Sleator et al. [15] proved that the diameter of $G_T(n)$ is $2n-10$. Lee [10] proved that the automorphism group of $G_T(n)$ is the dihedral group of order $2n$ and can be realized as an $(n-3)$-dimensional convex polytope called the *associahedron*. Most of these results are proved again in [8] using a unifying framework called the *tree of triangulations*.

Of the flip graphs we have studied, $G_T(n)$ is the one for which we have made the least progress. We present our results as a starting point for further research in the area.

Theorem 7 $\chi(G_T(n)) \leq \lceil\frac{n}{2}\rceil$.

Proof. It is well known that for n even, the set of $\binom{n}{2}$ edges between the vertices of S can be partitioned into $\frac{n}{2}$ edge-disjoint geometric non-crossing graphs (see [11, 2] and Figure 4 for an example with $n=6$). Since $G_T(n)$ is a subgraph of $G_T(n+1)$, we will assume n to be even for the time being. Label these graphs $G_1,\ldots,G_{\frac{n}{2}}$. If an edge e belongs to G_i, assign it the weight $w(e) = i$. To every triangulation $\mathcal{T}$ of S, assign the number $(\sum_{e\in\mathcal{T}} w(e)) \mod \frac{n}{2}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two adjacent triangulations of $G_T(n)$, i.e. $\mathcal{T}_2 = \mathcal{T}_1 - e + f$ for some crossing edges e and f. Since e and f cross each other, $w(e) \neq w(f)$, and thus the numbers associated to $\mathcal{T}_1$ and $\mathcal{T}_2$ are different. This induces a coloring of $G_T(n)$ with $\frac{n}{2}$ colors for n even and $\lceil\frac{n}{2}\rceil$ in general. □

The upper bound on $\chi(G_T(n))$ in Theorem 7 is non-trivial since, for example, Brooks' Theorem [3] gives an upper bound of only $n-1$. However, $\chi(G_T(n))$ is in fact sublinear, as we now show. Johansson [9]

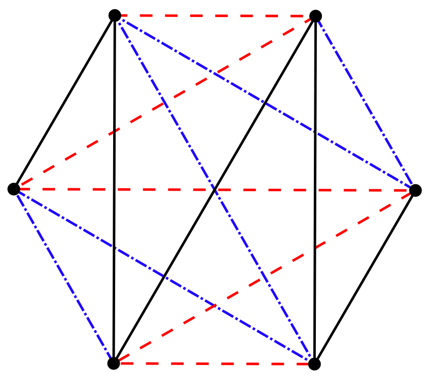


Figure 4: Partition of the edges into disjoint non-crossing geometric Hamiltonian paths

proved that for sufficiently large Δ, every triangle-free graph with maximum degree Δ is $O\left(\frac{\Delta}{\log \Delta}\right)$-colorable; see [13]. The next theorem follows since $G_T(n)$ is $(n-3)$-regular and triangle-free.

Theorem 8 $\chi(G_T(n)) \in O\left(\frac{n}{\log n}\right)$.

We remark that the proof of Theorem 7 is constructive, whereas Johansson's proof is probabilistic.

6 Open Problems

The most challenging open problems arising from this work are to determine the chromatic numbers of $G_T(n)$ and $\mathcal{M}(K_{2n})$. Despite our efforts, we have not been able to obtain non-trivial lower bounds for these graphs. We have made the following educated guess of the chromatic number of $\mathcal{M}(K_{2n})$.

Conjecture 1 $\chi(\mathcal{M}(K_{2n})) = n + 1$.

We have verified this conjecture with the aid of a computer for $n = 2, 3, 4$.

We have studied $\mathcal{M}(K_{2n})$, but it would be very interesting to study $\mathcal{M}(G)$ for other graphs.

There is another flip graph, $\mathcal{M}(G)$, related to $\mathcal{M}(G)$. $\mathcal{M}(G)$ has as its vertex set the perfect matchings of G, where two matchings are now adjacent if and only if their symmetric difference is a cycle of arbitrary length. We have yet to study its chromatic number.

With respect to $\chi(G_T(n))$, the problem seems far more intriguing since not only do we not have a non-trivial lower bound, we also believe that our upper bound is far from being tight.

Conjecture 2 $\chi(G_T(n)) = \Theta(\log(n))$.

It would be interesting to see what techniques will be capable of improving the lower bounds of both of these graphs. It is likely that such techniques will turn out to be useful in determining the chromatic number of other graphs.

References

[1] P. Bose and F. Hurtado. *Flips in planar graphs.* Technical Report TR-06-09, School of Computer Science, Carleton University, Ottawa, 2006.

[2] P. Bose, F. Hurtado, E. Rivera-Campo and D. R. Wood *Partitions of complete geometric graphs into plane trees.* Computational Geometry: Theory an Applications, 34(2):116–125,2006.

[3] R. L. Brooks. *On coloring the nodes of a network.* Cambridge Philos. Soc., 37:194–197, 1941.

[4] R. Cummins. *Hamilton circuits in tree graphs.* IEEE Transactions on Circuit Theory 13:82–90, 1966.

[5] V. Estivill-Castro, M. Noy and J. Urrutia. *On The chromatic number of tree graphs.* Discrete Math. 233:363–366, 2000.

[6] W. Goddard and H. C. Swart. *Distances between graphs under edge operations.* Discrete Math. 161:121–132, 1996.

[7] C. Hernando, F. Hurtado and M. Noy. *Graphs of non-crossing perfect matchings.* Graphs and Combinatorics, 18:517–532, 2002.

[8] F. Hurtado and M. Noy *Graph of triangulations of a convex polygon and tree of triangulations.* Computational Geometry: Theory and Applications, 13(3):179–188, 1999.

[9] A. Johansson. *Asymptotic choice number for triangle free graphs.* DIMACS Technical Report 91–95, 1996.

[10] C. Lee. *The associahedron and the triangulations of the n-gon.* European J. Combinatorics 10(4):551–560, 1989.

[11] D. E. Lucas. *Recreations Mathématiques*, Volume 2. Gauthiers Villars, Paris 1892.

[12] J. M. Lucas. *The rotation graph of binary trees is Hamiltonian.* J. Algorithms, 8(4):503–535, 1987.

[13] M. Molloy and B. Reed. *Graph colouring and the probabilistic method.*, Algorithms and Combinatorics, Vol. 23, Springer, 2002.

[14] E. Rivera-Campo and V. Urrutia-Galicia. *Hamilton cycles in the path graph of a set of points in convex position.* Computational Geometry: Theory and Applications, 18(2):65–72, 2001.

[15] D. Sleator, R. Tarjan and W. Thurston *Rotations distance, triangulations and hyperbolic geometry.* J. Amer. Math. Soc., 1:667–682, 1988.

[16] W. Trotter. *Ramsey theory and partially ordered sets.* Contemporary Trends in Discrete Mathmatics, R. L. Graham, et al., eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science 49:337–347, 1999.

[17] V. Urrutia-Galicia. *Algunas propiedades de gráficas geométricas.* Ph.D. thesis, Universidad Autónoma Metroploitana-Iztapalala, México D.F., 2000.

Computing multiple convex hulls of a simple polygonal chain in linear time

Lilian Buzer [*][†], buzerl@esiee.fr

Abstract

In 1987, Melkman published an online algorithm that can compute the convex hull of a simple polygonal chain in linear time [1]. Its short implementation based on a basic data structure leads to a very simple and efficient function. Twenty years later, we revisit his method and propose to solve a more complex problem. For a given polygonal chain $S = (v_1, v_2, \ldots, v_n)$, we want to determine the convex hulls of some subchains of the form $S_i^j = (v_i, v_{i+1}, \ldots, v_{j-1}, v_j)$. Consider two non-decreasing sequences $(i_u)_{1 \le u \le k}$ and $(j_u)_{1 \le u \le k}$ satisfying $1 \le i_u \le j_u \le n$ for all u, $1 \le u \le k$. We succeed in proving that the computation of the k convex hulls of the subchains $(S_{i_u}^{j_u})_{1 \le u \le k}$, can be processed online in optimal linear time.

1 Introduction

A polygonal chain S in the Euclidian plane, with n vertices, is defined as an ordered list of n vertices $(v_1, v_2, \ldots, v_n)$ such that any two consecutive vertices v_i, v_{i+1} are connected by a line segment. A polygonal chain is called *simple* when it is not self-intersecting. In this paper, we use the notation S_i^j to describe the subchain of S corresponding to the vertices $v_i, v_{i+1}, \ldots, v_{j-1}, v_j$.

In 1987, Melkman proposed an online algorithm that successively computes the convex hulls of the subchains $(S_1^j)_{1 \le j \le n}$ in optimal linear time. It only requires to use a sequence of vertices associated with a simple polygonal chain. We extend this problem to a more general case where we propose an online method that computes all the convex hulls of the subchains $(S_{i_u}^{j_u})_{1 \le u \le 2n}$ in optimal linear time. The two non-decreasing sequences $(i_u)_{1 \le u \le 2n}$ and $(j_u)_{1 \le u \le 2n}$ must satisfy $i_1 = j_1 = 1$, $i_{2n} = j_{2n} = n$ and $i_u \le j_u$ for any u. When we know the convex hull of the subchain $S_{i_u}^{j_u}$, we can choose like in Melkman's algorithm to insert the next point v_{j_u+1}. The two sequences also satisfy: $i_{u+1} = i_u$ and $j_{u+1} = j_u + 1$. But now, we are able to choose to withdraw the first point v_{i_u} of the current subchain and in this case : $i_{u+1} = i_u + 1$ and $j_{u+1} = j_u$.

Such a problem arises when we want to perform a geometrical simplification of a polygonal chain. When we use the general framework of Imai and Iri [2], the problem is formulated as follows: for a given polygonal chain S, the subchain S_i^j can be approximated by the straight segment $v_i v_j$ when the subchain S_i^j is contained in a tolerance region defined by $v_i v_j$. When the approximation criterion is linked to a property of the convex hull of S_i^j, our method can be reused to offer efficiency and a low complexity bound.

In the first Section, we recall Melkman's algorithm and its main properties. We show that the reverse case where only deletions are supported and where we also process the convex hulls of the subchains $(S_i^n)_{1 \le i \le n}$ can be simply deduced from the existing algorithm. In Section 2, we show that the convex hulls of two consecutive subchains of S intersect in at most two points. This main property allows us to set up our new algorithm in Section 3. In the last Section, we show that the resulting complexity is linear in the number of vertices.

2 Melkman's algorithm

Data structure representation. The algorithm uses a deque D (doubly ended queue) where the vertices of the current convex hull are stored in counterclockwise order. However, the last vertex created on the convex hull is stored twice at the top and at the bottom of the deque. You can see an example in Figure 1.a. At the beginning of the algorithm, the deque is initialized with three vertices associated with a counterclockwise triangle.

Insertion of a new point. We successively insert points corresponding to the vertices of a simple polygonal chain. When a new point is processed, Melkman proves that a simple test allows us to determine whether the new point is inside the convex hull. You can see an example in Figure 1.b. Let us suppose that v_i is the last vertex inserted on the convex hull. If the next point v_{i+1} lies inside the current convex hull, it can only be located in the area A and B on the Figure. In fact, if it was elsewhere inside the convex hull, the straight line segment $v_i v_{i+1}$ should cross the polygonal chain and that would contradict the hypothesis of simplicity. Thus, it is sufficient to test the location of the new point v_{i+1} relative to the two straight line segments: $D_{top-1} D_{top}$ and $D_{bottom} D_{bottom+1}$.

Convexifying. When the next point p lies outside the current convex hull, we compute the new resulting

[*]A2SI Laboratory, ESIEE, 2 boulevard Blaise Pascal, Cité Descartes, - BP 99, 93162 Noisy-Le-Grand Cedex, France

[†]Institut Gaspard Monge, Unité Mixte CNRS-ESIEE, UMR 8049, 77454 Marne-la-Valle, France

convex body. For this, classical approaches like phase two of "Graham's scan" or "three coins algorithm" can be used in order to fill the concave angles created by p and the neighboring points on the hull border. For each backtracking, one vertex is deleted and so the number of operations is bounded by the number of the vertices of the polygonal chain. You can see an example in Figure 1.c.

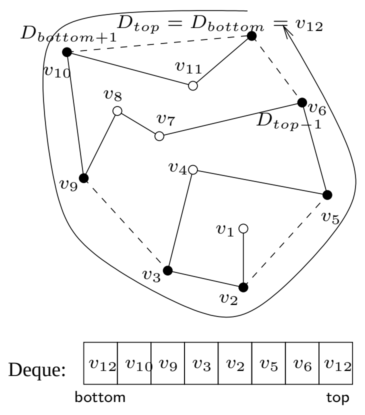
Source Code. We present the inner loop of Melkman's algorithm:

```
Get the next vertex $v$
    // Test if $v$ is inside the current convex hull
    If ($v$ is on the left of $D_{bottom}D_{bottom+1}$) and
        ($v$ is on the left of $D_{top-1}D_{top}$)
      skip $v$ and proceed to the next vertex

    // Update the tangent to the bottom
    While $v$ is on the right of $D_{bottom}D_{bottom+1}$
        Remove $D_{bottom}$ from the bottom of $D$
    Push $v$ at the bottom of $D$

    // Update the tangent to the top
    While $v$ is on the right of $D_{top-1}D_{top}$
        Remove $D_{top}$ from the top of $D$
    Push $v$ onto the top of $D$
```

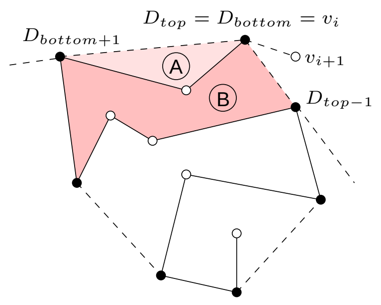
Reverse Melkman's algorithm. For a given polygonal chain S of n vertices, we propose to handle a similar problem where we want to successively compute the convex hulls of the subchains S_i^n from $i = 1$ to n. For this, we apply Melkman's algorithm as usual but we process the points in reverse order from the vertex v_n to the vertex v_1. Thus, we successively have access to the convex hulls of S_n^n up to the convex hull of S_1^n. To obtain the same results from S_1^n up to S_n^n, we keep in memory the successive deletions that appear during the insertion of each vertex. This list of events is used to build back the successive convex hulls of S_1^n up to S_n^n. For example, we store in memory that the insertion of the vertex v_i in the convex hull of S_{i-1}^n suppresses the vertices v_j and v_k from the bottom of the deque and the vertices v_j, v_l and v_m from the top of the deque. Then, when we want to deduce the convex hull of S_{i-1}^n from the convex hull of S_i^n, we only withdraw v_i from the top and the bottom of the deque, push back v_j and v_k on the bottom of deque and push back v_j, v_l and v_m on the top of the deque. Each operation is processed in constant time and the total number of operations is bounded by the number of vertices. So, the overall cost of this procedure is linear.

3 Main proposition

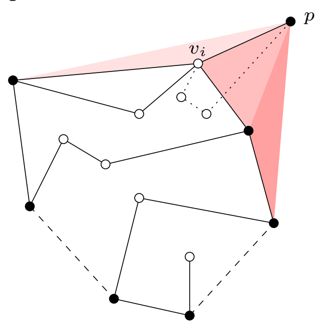
Proposition 1 *For a simple polygonal chain S of n vertices, the convex hulls of two consecutive subchains S_i^j and S_k^u with $j \leq k$ can intersect at most twice.*



(a) Representation of the convex hull



(b) Testing if the next vertex lies inside the hull



(c) Updating the current convex hull

Figure 1: Melkman's algorithm.

Proof. Let T and T' denote the convex hulls associated with the subchain S_i^j and the subchain S_k^l respectively. Suppose that T and T' intersect in two points I_1 and I_2. Let A_1 denote the vertex on the convex hull T that follows I_1 and that lies outside the intersections of $T \cap T'$. In the same way, we define A_2, A_1' and A_2'. Relative to the convex property of T and T', there exists two *forbidden areas* delimited by their tangents I_1A_1, I_1A_1', I_2A_2 and I_2A_2' where T and T' cannot lie. Only two configurations can appear depending on the possible locations of I_1A_1 and I_2A_2 relative to I_1I_2. Consider the second configuration presented in Figure 2. As A_1 and A_2 belongs to the same subchain there exists a polygonal path t that joins these two points. In the same way, there exists a polygonal path that joins A_1' and A_2'. Nevertheless, if

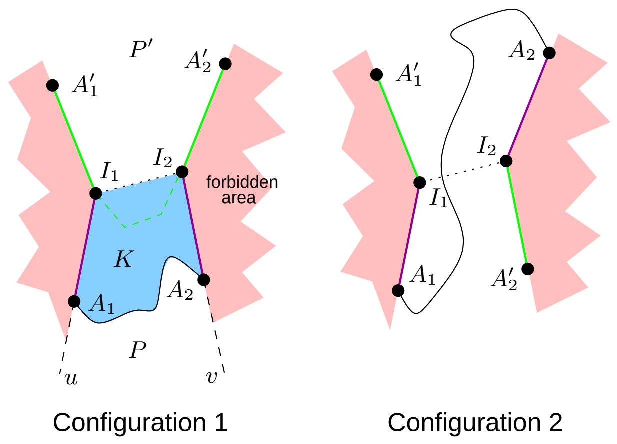


Figure 2: Intersections of two consecutive polygonal subchains.

this path exists, as it cannot traverse the two forbidden areas, it would have to cross the first path t and that would contradict the hypothesis of simplicity of S. Thus, only the first configuration can exist.

We consider the area P delimited by u, I_1, I_2 and v and the area K (blue colored) delimited by A_1, I_1, I_2, A_2 and the polygonal path that joins A_2 to A_1. Let C and C' denote $T \cap P$ and $T' \cap P$. Using the simplicity of S, the polygonal subchain associated with T' cannot cross the polygonal path that joins A_2 to A_1. Moreover, relative to the convex property of the convex hull T', T' is located on each side of its tangents A'_1I_1 and A'_2I_2. Thus, the convex body C' is included in the domain K and it follows that $C' \subset$ *convex hull*(K). As *convex hull*$(K) \subset C$, we finally obtain $C \subset C'$. Symmetrically, we obtain the same property on the other side of I_1I_2. As a consequence, no intersections between T and T' can lie in P, in P' and in the forbidden areas. So, T and T' intersect at most twice. □

Observation 1 *An immediate consequence for our problem is that there exist at most two bridges between the convex hull of S_i^k and S_k^l.*

4 Our algorithm

4.1 Online processing

We now explain how to handle the online insertions and deletions. Our algorithm produces a sequence of *event points*. Suppose that the procedure has just ended and let us have a look at two consecutive event points v_i and v_j with $i \leq j$. A decremental convex hull C^- has been managed from v_j to v_i and an incremental convex hull C^+ has been processed from v_j. The successive mergings of the two convex hulls C^- and C^+ gave the different results. When insertions are requested, points are inserted into C^+. When we ask to delete a point v_u with $i \leq u \leq j$, the operation

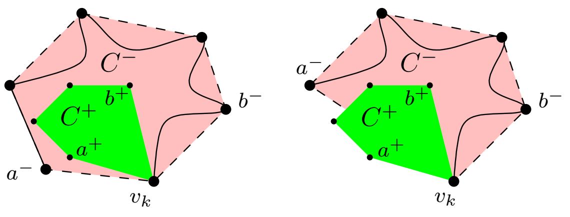


Figure 3: Testing whether C^+ is included in C^-.

is performed in C^-. When the last vertex of C^- disappears, this decremental convex hull is empty and it is now useless. Let v_k denote the last vertex created on the border of C^+ and let v_l denote the last inserted point. A new decremental convex hull C^- is initialized using S_j^k and a new incremental convex hull C^+ is initialized to S_k^l. The point v_k is the next event point after v_j. We apply the same process until the end. To start the algorithm, we choose the two event points to be equal to v_1.

4.2 First stage: $C^+ \subset C^-$

Consider that the couple of event points changes to v_j and v_k. As v_k is the last created vertex on the border of the previous C^+, the subchain S_k^l is also included in the new convex hull C^-. Thus, at the beginning of this new phase, we have: $C^+ \subset C^-$.

When a new point p is inserted in C^+, it is quite simple to check whether C^+ is included in C^-. The classical Melkman's condition guarantees that checking the location of p relative to two edges emerging from v_k on the border of C^- is sufficient.

We now explain how to check whether C^+ is included in C^- when a point is withdrawn from C^-. Since v_k is an extreme point of C^- and k is the greatest index of the points of C^-, v_k will always remain an extreme point of C^- (see Fig. 3). Let a^- and b^- (resp. a^+ and b^+) denote the two vertices neighboring v_k on the border of C^- (resp. C^+). Since v_k belongs to C^+ and v_ka^- and v_kb^- are tangent to C^+, v_k is an extreme point for C^+ too. When a^+ and b^+ are interior to v_ka^- and v_kb^-, C^+ is included in the domain defined by a^-, v_k, b^- and the polygonal subchain that joins a^- to b^-. As this domain is included in C^-, C^+ is inside C^-. Thus, testing if a^+ and b^+ are interior to a^-v_k and v_kb^- is sufficient.

When C^+ is no more included in C^-, we compute the merging C of C^+ and C^- in time linear in the number of vertices (using the rotating caliper algorithm for example). Thus, we determine which vertices of C^+ and of C^- contribute to the definition of C and which vertices support the two bridges. After this, we proceed to the second step.

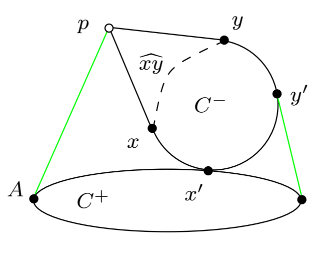
4.3 Second stage: $C^- \nsubseteq C^+$

Incremental case. When we insert a new point p in C^+, we concurrently insert it in C (the merging between C^+ and C^-) using Melkman's routine (see Fig. 4.b). To offer a virtual access to C, we test for each operation in $O(1)$ if we operate on C^+, on C^- or on one bridge. This simple approach automatically updates the current bridges.

Decremental case. Removing a point p in C^- that does not support one of the two bridges is a simple operation. Nevertheless, when this special case happens (see Fig. 4.a), the corresponding problem is equivalent to merging two convex hulls. We cannot launch a routine that updates the current bridges in time linear in the number of vertices of C^+ and C^- elsewhere we could not achieve a linear time complexity. Let x and y denote the two neighboring points of p on C^- and let $\widehat{xy}$ denote the border of C^- that appears after the deletion of p. We now prove that the new bridge that replaces the bridge supported by A and p is inevitably supported by $\widehat{xy}$. If the new bridge was supported by yy', one point of C^+ should lie above the straight line py. But py was tangent to C the step before and so C^+ lies under this line. In the same way, if the new bridge was supported by $\widehat{xx'}$, C^+ should lie on the right of the straight line px and this contradicts that A, the point of C^+ that supports the old bridge, lies on the left of px.

To find the new bridge, we use a rotating caliper approach. First, we find the vertex of $\widehat{xy}$ that supports a line parallel to Ap to obtain an antipodal pair between C^+ and C^-. Second, we rotate these lines until we find the new bridge. The previous remark guarantees that we traverse at most k vertices on C^- where k denote the number of vertices of $\widehat{xy}$.

Observation 2 *When C^- is included in C^+ we leave the second stage and enter the last stage. As deletions in C^- have no influence on C, we only perform operations when insertions are done in C^+ and $C = C^+$.*



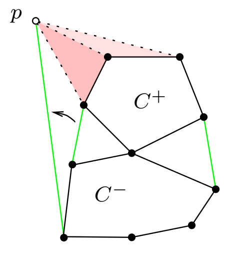


Figure 4: Updating C^+, C^- and C.

5 Complexity analysis

Let $(v_{\alpha_i})_{2\leq i\leq k-1}$ denote the subsequence of $(v_i)_{1\leq i\leq n}$ that describe the event points. We fix $\alpha_0 = \alpha_1 = 1$ and $\alpha_{k-1} = \alpha_k = n$. Between three event points v_{α_i}, $v_{\alpha_{i+1}}$ and $v_{\alpha_{i+2}}$, we manage a decreasing convex hull C^- from $v_{\alpha_{i+1}}$ to v_{α_i} and an increasing convex hull C^+ from $v_{\alpha_{i+1}}$ to $v_{\alpha_{i+2}}$. Let δ_i denote $\alpha_{i+1} - \alpha_i$. The number of vertices of C^- (resp. C^+) is bounded by δ_i (resp. δ_{i+1}). So, if the number of operations processed during the three stages is linear in $\delta_i + \delta_{i+1}$, the overall complexity $T(n)$ of our method satisfy: $T(n) = \sum_{i=0}^{k-2} O(\delta_i + \delta_{i+1}) \leq 2\sum_{i=0}^{k-1} O(\delta_i) = O(n)$

Between three event points v_{α_i}, $v_{\alpha_{i+1}}$ and $v_{\alpha_{i+2}}$, we estimate the operations processed during the three stages. Operations due to insertions in C^+ are bounded by $O(\delta_{i+1})$ because of Melkman's algorithm. Deletions in C^- produces $O(\delta_i)$ operations because of reverse Melkman's algorithm. Checking whether C^+ lies inside C^- is a constant time operation done at most $O(\delta_i + \delta_{i+1})$ times. Computing the merging of C^+ and C^- to enter the second stage is a linear operation. Updating C relative to insertions in C^+ is an equivalent work than Melkman's algorithm applied on $O(\delta_i+\delta_{i+1})$ vertices. The sole difficult operation is the deletion of vertices that support the bridges between C^+ and C^-. We soon prove that for each deletion, the number of operations relative to C^- is bounded by the number of vertices added in C^- when the vertex disappears. Thus, this quantity is bounded by $O(\delta_i)$. To bound the number of operations processed on C^+ (bridge displacement), we have to notice that C^+ increases and that C^- decreases. Thus, a bridge supported by a vertex of C^+ that moves to another vertex can never return to the previous point. So, the overall number of operations processed for the case described in Figure 4.a is bounded by $O(\delta_{i+1})$. The number of operations is then bounded $O(\delta_i + \delta_{i+1})$.

6 Conclusion

We revisit the classical online convex hull algorithm of a simple polygonal chain and extend it to a more general case where deletions from the queue are possible. As this functionality is quite interesting, many applications may emerge from this method. Moreover, this algorithm achieves an optimal linear time complexity unlike dynamic convex hull algorithms of the literature that are at least $O(n \log^2 n)$.

References

[1] A. Melkman. *On-line Construction of the Convex Hull of a Simple Polygon.* IPL. 25, p.11, 1987.

[2] H. Imai, M. Iri. *Computational-geometric methods for polygonal approximations of a curve.* Computer Vision, Graphics and Image Proc. pp:31-41, 36, 1986.

Cluster registration in 2D geometric constraint solving

David Podgorelec[*] Borut Žalik[†]

Abstract

We present two original features of our new 2D geometric constraint solver. Firstly, we introduce the cluster registration step, which reduces the number of clusters below the number of points and, consequently, importantly accelerates the cluster merging process. We also propose a heuristic search for a minimal mergeable subproblem, which further accelerates the recombination phase, extends the drawing scope of the method and improves generality.

1 Introduction

Geometric constraint solving was first introduced in CAD to suppress disadvantages of classical geometric modellers. Namely, a *geometric constraint problem (GCP)* consisting of a finite set of geometric elements and a finite set of geometric constraints (relations between the elements), captures a designer's intent more naturally than traditional geometric modellers. The solution of the GCP is a class of geometric element instantiations in a given coordinate system, such that all the constraints are satisfied [1]. Of course, we expect to obtain this solution automatically.

Geometric constraint solvers can be classified into two main groups: direct and constructive solvers. A *direct solver* transforms constraints into equations and, simultaneously, solves them by employing a general numerical [2] or symbolic technique [3]. The time complexity of direct solvers is exponential and, therefore, their use should be restricted to subproblems that are as small as possible. Isolation of such subproblems, whose solutions can be recombined by solving other small subproblems, represents the main task of the *constructive solvers*. These solvers typically represent a GCP by a constraint graph G = (N, E) where the set of nodes N consists of geometric elements and the edges from E represent constraints. The constructive solvers either use a set of construction rules to solve some predefined subgraph patterns (shape-recognition-based constraint solvers) [4, 5], or they rely on more general techniques [1, 6], usually slower but with the extended drawing scope.

The presented algorithm is a representative of the shape-recognition-based solvers. In Section 2, we outline its structure. Section 3 introduces an original concept of cluster registration, which importantly reduces the number of the initial clusters, created in the decomposition phase. In Section 4, we deal with the recombination phase. We first describe the strategy. Then we present a simple, but efficient heuristic algorithm for minimal mergeable subproblem search. Finally, we offer a brief summary in Section 5.

2 The method

We consider only GCPs in Euclidean plane. The geometric elements are points and lines, which form the so-called auxiliary geometry, but more complex geometric elements of the visible geometry as, for example, circular arcs and cubic Bézier curves, can also be handled by mapping them onto the control points and lines [7]. A user can use 26 predefined constraint types, but constraints of 15 types are decomposed into conjunctions of intuitively simpler constraints. To further simplify the GCP, the constraints of the remaining 11 types are transformed into constraints (equations) among the corresponding control points only. Transformations of distances are based on the formula for the Euclidean distance between a pair of points. Angles are transformed by employing the relation between the scalar product of two vectors and the angle between the vectors.

The decomposition phase represents a variant of the algorithm introduced by Sunde [5]. It operates with two types of point sets: clusters and CA sets.

Definition 1 *A cluster is a maximal structurally rigid geometric part.*

The cluster's shape and size are completely constrained, unlike its position and orientation. Namely, the required structural rigidity means that a cluster retains three degrees of freedom, two translations and a rotation. Definition 2 introduces the structural rigidity, and recalls some terms, closely related to it.

Definition 2 *The GCP represented by a constraint graph G = (N, E) is structurally:*

- *over-constrained if there is an induced subgraph $G' = (N', E')$ with $|E'| > 2|N'| - 3$,*
- *under-constrained if G is not structurally over-constrained and $|E| < 2|N| - 3$,*

[*]Faculty of Electrical Engineering and Computer Science, University of Maribor, david.podgorelec@uni-mb.si

[†]Faculty of Electrical Engineering and Computer Science, University of Maribor, zalik@uni-mb.si

- *well-constrained if G is not structurally over-constrained and $|E| = 2|N| - 3$,*
- *well over-constrained if it is structurally over-constrained, and if it can be transformed into a structurally well-constrained GCP by removing structurally redundant constraints [8],*
- *rigid if it is structurally well constrained or structurally well over-constrained.*

Different choices of initial clusters are possible. The decomposition phase is pre-processed by determination of redundant constraints. Solving triangles, determining the sums of adjacent angles, and elimination of the so-called simple conditional constraints are used to determine redundant distances and angles [7]. This pre-processing assures that point sets with known distances between all pairs of member points are typically large enough to represent a good choice for the initial clusters. This choice enables development of a simple and efficient cluster creation algorithm. Note that the required *maximality* in Definition 1 means that none of clusters may represent a subset of any other cluster. We also require that each point should be a member of at least one cluster. A cluster containing a single point is also acceptable.

Definition 3 *A cluster is called 1-cluster if it contains a single point, 2-cluster if it contains two points, 3-cluster if it contains three points, or k-cluster if it contains three or more points.*

Definition 4 *A constrained angle set (CA set) is a 2-cluster, a k-cluster or a part consisting of 2-clusters and/or k-clusters with known angles between them.*

The recombination phase merges clusters into larger clusters, and CA sets into larger CA sets. A model is completely constrained when all the points belong to the same cluster.

3 Cluster registration

Definition 5 *A constraint is called an inter-cluster constraint if it addresses points from different clusters. Constraints that are not inter-cluster will be called the intra-cluster constraints.*

Cluster creation treats all Distance constraints as intra-cluster constraints. Consequently, it usually produces a high number of initial 2-clusters. In Figure 1, each of 32 Distance constraints results in a separate 2-cluster. The recombination phase analytically solves only patterns with up to three clusters and, therefore, the obtained decomposition is not optimal for further processing. We introduce the *cluster registration* algorithm to prevent such situations. It registers all 1-clusters and k-clusters, and those 2-clusters, assuring that each point remains in at least one cluster. Only these registered clusters enter the recombination phase, and all other 2-clusters are replaced by inter-cluster Distance constraints. Only 8 of 32 clusters in Figure 1 are registered. Different implementations of cluster registration are possible, and there is also not a unique solution. Figure 1(b) represents only one of the possible solutions. Eight nearly horizontal or eight non-intersecting diagonal 2-clusters are also acceptable, and there are many other solutions with eight or more registered clusters. An optimal cluster registration algorithm would run in $O(k)$, where k is the number of initial clusters before the registration. The algorithm must also assure that each registered 2-cluster contains at least one point which is not present in any other registered cluster. This enables us to state the important Theorem 1.

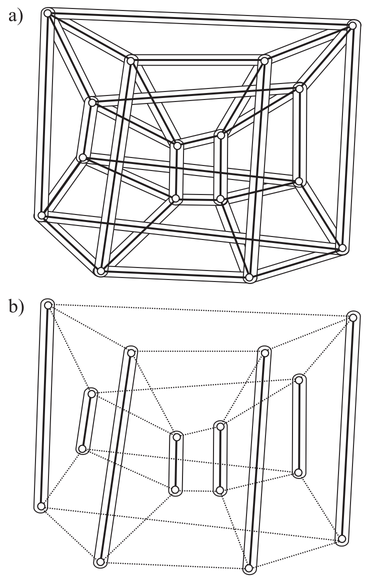


Figure 1: Cluster registration reduces the number of clusters.

Theorem 1 *The number of registered clusters cannot exceed the number of points n.*

Proof. Let us assume the opposite: a GCP with n points is decomposed into m registered clusters, where $0 < n < m$. Let $r \leq m$ represent the total number of 1-clusters and 2-clusters. If we remove these r registered clusters, we simultaneously remove at least r points. We obtain a subproblem with, at most, $p = n - r$ points and with exactly $q = m - r$ k-clusters. Our hypothesis $0 < n < m$ becomes $0 \leq p < q$. Then we reduce each k-cluster into a 3-cluster. This operation may also reduce the point set, but the number of 3-clusters (former k-clusters) remains q. Let N' and E' represent the point set and the set of the intra-cluster constraints Distance in all 3-clusters, respectively. There are three such constraints in each 3-cluster and, consequently, |E'| $= 3q$. (N', E') is certainly not structurally over-constrained because none of the triangles (3-clusters) constrained by three side lengths is not structurally over-constrained. Therefore, |E'| $\leq 2|N'| - 3$. We now use $0 \leq$ |N'| $\leq p < q$, derived from the initial hypothesis, and |E'| $= 3q$. We obtain $3q < 2q - 3$. The solution $q < -3$ proves that the initial hypothesis was incorrect since the number of k-clusters is not allowed to be negative. □

4 A heuristic determination of minimal mergeable subproblems

The recombination phase searches for pairs or triples of clusters that can be merged by employing predefined construction rules. We use 7 different construction rules, three of them addressing a pair of clusters, and four of them aimed to merge triples of clusters. Simultaneously with cluster merging, the process of merging CA sets is running. This process usually results in newly determined angles between pairs of clusters from the merged CA sets, and these angles may then encourage further cluster merging.

If the described process does not result in a single cluster then we must cope with larger k-tuples of clusters. Before we perform a cluster merging operation, we must be sure that the observed subproblem does not contain smaller mergeable subproblems. Our 7 construction rules are designed for patterns that automatically fulfill this requirement but, generally, we have to search for a so-called minimal mergeable subproblem. The brute force approach traverses all possible combinations. This strategy is reliable but it requires exponential time and is, therefore, applicable only when the number of clusters is considerably low. We could handle this problem by some general constructive solver [1, 6]. Here we introduce a simple heuristic approach that serves quite well and remarkably fast for this purpose.

Definition 6 *A subproblem is called minimal mergeable subproblem if it is structurally rigid and it does not contain smaller structurally rigid subproblems.*

Our heuristic algorithm sequentially eliminates clusters and checks the structural rigidity of the remaining subgraph. If a structurally rigid subgraph is identified then we extract it and deal with it independently. Let us suppose that the inter-cluster constraints link m points organized into k clusters. If a subgraph is structurally rigid then it must contain at least $2m - 3$ constraints. $2m - 3k$ intra-cluster constraints provide structural rigidity of the clusters, and the surplus of $3(k-1)$ constraints represents the inter-cluster constraints. This value is updated and compared to the *number of scalar equations NSE* after each cluster elimination. *NSE* nearly corresponds to the number of inter-cluster constraints, but three situations require special treatment.

1. The constraint Coincidence contributes 2 to *NSE*.
2. A point shared by k clusters contributes $2(k-1)$ to *NSE*.
3. In a group of k clusters from the same CA set, $k(k-1)/2$ angles between pairs of clusters contribute only $k-1$ to *NSE*.

The heuristic criterion for selection of a cluster for elimination is based on the clusters' *valences*. The valence generally means the number of scalar equations obtained from the inter-cluster constraints addressing the cluster, but here we also meet two exceptions.

1. A point shared by k clusters contributes $2(k-1)/k$ to the valence of each of the k clusters.
2. In a group of k clusters from the same CA set, angles between these clusters increase valences of each of the k clusters for $(k-1)/k$ when $k > 2$, or for 1 when $k = 2$.

Let us remark that we may ignore those clusters with the valence 3 or less. Since a cluster has three degrees of freedom, three equations are only sufficient to position a single cluster and cannot contribute to positioning of the remaining clusters.

The six rules listed below determine the cluster to be eliminated. If the first rule finds more candidates for elimination then the second rule is employed etc. The priorities of the rules have been experimentally established during some hundreds of tests:

1. the lowest valence in the updated subgraph,
2. the lowest number of neighbours in the updated subgraph,
3. the cluster updated the most recently,
4. the lowest initial valence,
5. the lowest initial number of neighbours, and
6. the lowest index.

The goal of cluster elimination is to isolate a minimal mergeable subproblem by removing structurally under-constrained subproblems. The heuristic criterion is based on the assumption that a structurally under-constrained subgraph is intuitively less dense than the minimal mergeable subgraph. A cluster with low valence and a low number of neighbours is, therefore, a good candidate for belonging to some structurally under-constrained part. The third rule is particularly interesting. It tries to hold the elimination process in a presumably structurally underconstrained part close to the preceding eliminations, instead of jumping chaotically from one end of the graph to another. However, the experimentally obtained heuristic criterion does not guarantee success in general. For this reason, we still separately consider patterns of two and three clusters.

Let us consider the example presented in Figure 2(a). It consists of 10 clusters and 27 inter-cluster constraints. The algorithm sequentially eliminates clusters C_5, C_2, C_1, C_3, and C_4. It determines equality $NSE = 3(k-1) = 12$ for the subproblem with $k = 5$ remaining clusters C_6 to C_{10}. The elimination process proceeds and determines that this subproblem does not contain smaller mergeable subproblems. The further steps are:

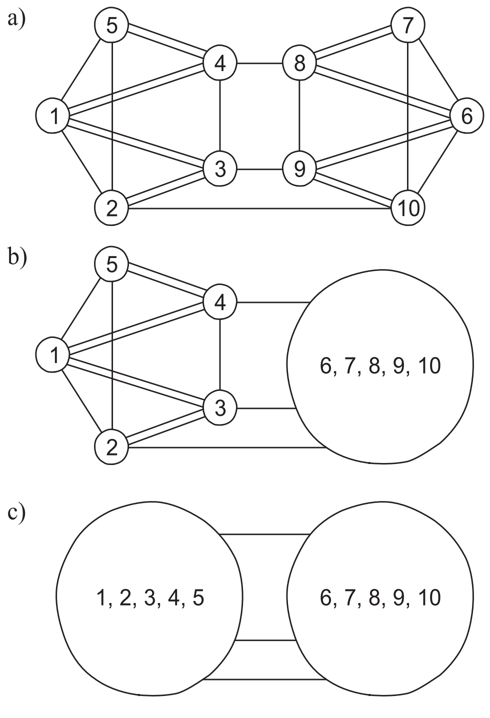


Figure 2: A decomposable structurally rigid problem.

1. We numerically merge the structurally rigid subproblem consisting of C_6 to C_{10} (Figure 2(b)).

2. The merged cluster C_{6-10} may be ignored since its valence is 3. We, therefore, deal with the subproblem consisting of C_1 to C_5.

3. This subproblem is recognised as mergeable. We numerically merge C_1 to C_5 (Figure 2(c)).

4. We merge both merged clusters C_{6-10} and C_{1-5} into a single cluster by employing one of the construction rules.

5 Conclusion

This paper presents two original features of our constructive geometric constraint solver. The cluster registration step assures that the number of clusters for further consideration does not exceed the number of points. Since the number of initial clusters is usually significantly reduced, the solver may often search for minimal mergeable subproblems by employing the most reliable brute-force approach. However, when the number of clusters remains too high, the method uses our original heuristic approach for determining minimal mergeable subproblems. This approach extends the drawing scope of the method and represents an important step towards generality. However, there are still problems open for future research. The heuristic algorithm requires further practical tests, theoretical confirmation or eventual correction of the selected heuristics. Multiple solutions of the cluster registration also require future investigation.

Acknowledgments

The authors are grateful to the Slovenian Research Agency for supporting this research under the project P2-0041 Computer systems, methodologies and intelligent services.

References

[1] C.M. Hoffmann, A. Lomonosov, M. Sitharam. *Decomposition Plans for Geometric Constraint Systems, Part II: New Algorithms.* J Symb Comp 2001;31:409-27.

[2] J.-X. Ge, S.-C. Chou, and X.-S. Gao. *Geometric constraint satisfaction using optimization methods.* Comput-Aided Des 1999;31:867-79.

[3] X.-S. Gao, S.-C. Chou. *Solving geometric constraint systems. II. A symbolic approach and decision of Rc-constructibility.* Comput-Aided Des 1998;30:115-22.

[4] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, J. Vilaplana-Pastó. *Revisiting Decomposition Analysis of Geometric Constraint Graphs.* ACM Solid Modeling 02, New York: ACM Press; 2003. p. 105-15.

[5] G. Sunde. *Eurographics workshop on Intelligent CAD Systems.* Noordwijkerhout, The Nederlands; 1987. p. 90-104.

[6] X.-S. Gao, G.-F. Zhang. *Geometric Constraint Solving via C-tree Decomposition.* ACM Solid Modeling 02, New York: ACM Press; 2003. p. 45-55.

[7] D. Podgorelec. *A new constructive approach to constraint-based geometric design.* Comput-Aided Des 2002;34:769-85.

[8] C.M. Hoffmann, M. Sitharam, B. Yuan. *Making constraint solvers more usable: overconstraint problem.* Comput-Aided Des 2004;36:377-99.

New Upper Bounds on the Quality of the PCA Bounding Boxes in $\mathbb{R}^2$ and $\mathbb{R}^3$

Darko Dimitrov* Christian Knauer* Klaus Kriegel* Günter Rote*

Abstract

Bounding boxes obtained by principal component analysis (PCA) usually quite well approximate the minimum-volume bounding boxes of a point set in $\mathbb{R}^d$. Here, we analize the worst case ratio of the volume of the PCA bounding box and the volume of the minimum-volume bounding box. Since there are examples of discrete points sets in the plane, where the worst case ratio tends to infinity, we consider PCA bounding boxes for continuous sets, especially for the convex hull of a point set. We contribute new upper bounds on the approximation factor of PCA bounding boxes of convex sets in $\mathbb{R}^2$ and $\mathbb{R}^3$.

1 Introduction

A frequently used heuristic for computing a bounding box of a set of points is based on *principal component analysis* (PCA) [3]. The principal components of the point set define the axes of the bounding box. Once the axis directions are given, the dimension of the bounding box is easily found by the extreme values of the projection of the points on the corresponding axis. Computing a bounding box of a set of points in $\mathbb{R}^2$ and $\mathbb{R}^3$ by PCA is simple and 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 the convex hull, or the boundary of the convex hull $CH(P)$ of the point set P. 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.

Given a point set $P \subseteq \mathbb{R}^d$ we denote by $BB_{pca}(P)$ the PCA bounding box of P and by $BB_{opt}(P)$ the bounding box of P with smallest possible volume. The ratio of the two volumes $\lambda_d(P) = \mathrm{Vol}(BB_{pca}(P))/\mathrm{Vol}(BB_{opt}(P))$ defines the approximation factor for P, and

$$\lambda_d = \sup\left\{\lambda_d(P) \mid P \subseteq \mathbb{R}^d, \mathrm{Vol}(CH(P)) > 0\right\}$$

defines the general PCA approximation factor.

Since bounding boxes of a point set P (with respect to any orthogonal coordinate system) depend only on the convex hull of $CH(P)$, the computation of the principal components should be based only on $CH(P)$ and not on the distribution of the points inside. Using the vertices, i.e., the 0-dimensional faces of $CH(P)$ to compute the principal components we obtain a bounding box $BB_{pca(d,0)}(P)$. We denote by $\lambda_{d,0}(P)$ the approximation factor for the given point set P and by

$$\lambda_{d,0} = \sup\left\{\lambda_{d,0}(P) \mid P \subseteq \mathbb{R}^d, \mathrm{Vol}(CH(P)) > 0\right\}$$

the approximation factor in general. In [1] it is shown that $\lambda_{d,0} = \infty$ for any $d \geq 2$. To overcome this problem, one can apply a continuous version of PCA taking into account (the dense set of) all points on the boundary of $CH(P)$, or even all points in $CH(P)$. Since the first case corresponds to the 1-dimensional faces of $CH(P)$ and the second case to the 2-dimensional face of $CH(P)$, the generalization to dimension $d > 2$ leads to a series of $d-1$ continuous PCA versions. For a point set $P \in \mathbb{R}^d$, $BB_{pca(d,i)}(P)$ denotes the PCA bounding box obtained by the i-dimensional faces of $CH(P)$. The approximation factors $\lambda_{d,i}(P)$ and $\lambda_{d,i}$ are defined as

$$\lambda_{d,i}(P) = \frac{\mathrm{Vol}(BB_{pca(d,i)}(P))}{\mathrm{Vol}(BB_{opt}(P))}, \quad \text{and}$$
$$\lambda_{d,i} = \sup\left\{\lambda_{d,i}(P) \mid P \subseteq \mathbb{R}^d, \mathrm{Vol}(CH(P)) > 0\right\}.$$

To the best of our knowledge, the only known results about the quality of the PCA bounding boxes were given in [1], where it was shown that $\lambda_{d,i} = \infty$ for any $d \geq 2$ and any $0 \leq i < d-1$. Additionally, lower bounds on $\lambda_{d,d}$ and $\lambda_{d,d-1}$ for arbitrary dimension d, and an upper bound on $\lambda_{2,1}$ were given. In what follows we present the first upper bounds on $\lambda_{2,2}$ and $\lambda_{3,3}$.

2 An Upper Bound for $\lambda_{2,2}$

Given a point set $P \subseteq \mathbb{R}^2$ and an arbitrary bounding box $BB(P)$, we will denote the two side lengths of $BB(P)$ by a and b, where $a \geq b$. We are interested in the side lengths $a_{opt}(P) \geq b_{opt}(P)$ and $a_{pca}(P) \geq b_{pca}(P)$ of $BB_{opt}(P)$ and $BB_{pca(2,2)}(P)$. The parameters $\alpha = \alpha(P) = a_{pca}(P)/a_{opt}(P)$ and $\beta = \beta(P) = b_{pca}(P)/b_{opt}(P)$ denote the ratios between the corresponding side lengths, so that $\lambda_{2,2}(P) = \alpha(P) \cdot \beta(P)$.

*Institut für Informatik, Freie Universität Berlin, Germany, {darko, knauer, kriegel, rote}@inf.fu-berlin.de

If the relation to P is clear, we will omit the reference to P in the notations introduced above.

Since the side lengths of any bounding box are bounded by the diameter of P, we can observe that in general $b_{pca}(P) \leq a_{pca}(P) \leq diam(P) \leq \sqrt{2}a_{opt}(P)$, and in the special case when the optimal bounding box is a square $\lambda_{2,2}(P) \leq 2$. This observation can be generalized, introducing an additional parameter $\eta(P) = a_{opt}(P)/b_{opt}(P)$.

Lemma 1 *$\lambda_{2,2}(P) \leq \eta + \frac{1}{\eta}$ for any point set P with aspect ratio $\eta(P) = \eta$.*

Proof. For both a_{pca} and b_{pca}, we have the upper bound $diam(P) \leq \sqrt{a_{opt}^2 + b_{opt}^2} = a_{opt}\sqrt{1 + \frac{1}{\eta^2}}$. Thus, $\alpha\beta = \frac{a_{pca}b_{pca}}{a_{opt}b_{opt}} \leq \frac{\left(a_{opt}\sqrt{1+\frac{1}{\eta^2}}\right)^2}{a_{opt}b_{opt}} = \frac{a_{opt}}{b_{opt}}(1 + \frac{1}{\eta^2})$. Replacing a_{opt} by $\eta \cdot b_{opt}$, we get $\alpha\beta = \eta\left(1 + \frac{1}{\eta^2}\right) = \eta + \frac{1}{\eta}$. $\square$

Unfortunately, this parametrized upper bound tends to infinity for $\eta \to \infty$. Therefore, we are going to derive another upper bound that is better for large values of η. We derive such a bound by finding a constant that bounds β from above. In this process we will make essential use of the properties of $BB_{pca(2,2)}(P)$. We denote by $d^2(CH(P), l)$ the integral of the squared distances of the points on $CH(P)$ to a line l, i.e., $d^2(CH(P), l) = \int_{s \in CH(P)} d^2(s, l)ds$. Let l_{pca} be the line going through the center of gravity, parallel to the longer side of $BB_{pca(2,2)}(P)$, and l_{opt} be the line going through the center of gravity, parallel to the longer side of $BB_{opt(P)}$. From principal component analysis [3], we know that l_{pca} is the best fitting line of P and therefore

$$d^2(CH(P), l_{pca}) \leq d^2(CH(P), l_{opt}). \qquad (1)$$

We obtain an estimate for β by determining a lower bound on $d^2(CH(P), l_{pca})$ that depends on b_{pca}, and an upper bound on $d^2(CH(P), l_{opt})$ that depends on b_{opt}. Having an arbitrary bounding box of $CH(P)$ (with side lengths a and b, $a \geq b$) the area of $CH(P)$ can be expressed as

$$\begin{aligned} A &= A(CH(P)) \\ &= \int_0^b \int_0^a \chi_{CH(P)}(x, y)dxdy \quad = \quad \int_0^b g(y)dy, \end{aligned}$$

where $\chi_{CH(P)}(x, y)$ is the *characteristic function* of $CH(P)$ defined as

$$\chi_{CH(P)}(x, y) = \begin{cases} 1 & (x, y) \in CH(P) \\ 0 & (x, y) \notin CH(P), \end{cases}$$

and $g(y) = \int_0^a \chi_{CH(P)}(x, y)dx$ is the length of the intersection of $CH(P)$ with a horizontal line at height y. In the following we call $g(y)$ the *density function* of $CH(P)$ for computing the area with the integral $\int_0^b g(y)dy$. Note that $g(y)$ is continuous and convex in the interval $[0, b]$ (see Figure 1 (a) for an illustration). Let b_1 denote the y-coordinate of the center of gravity of $CH(P)$. The line l_{b_1} $(y = b_1)$ divides the area of $CH(P)$ into A_1 and A_2. The following result,

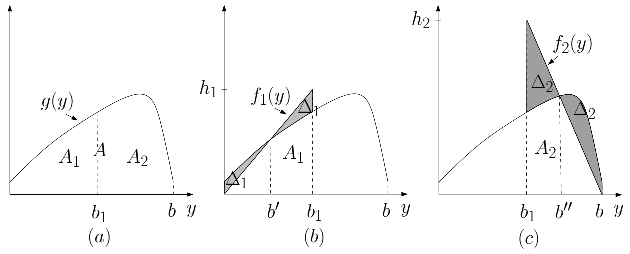


Figure 1: Construction of the lower bound for $d^2(CH(P), l_{b_1})$.

which is derived from the generalized first mean value theorem of integral calculus, is our central technical tool in derivation of the lower and the upper bound on $d^2(CH(P), l_{b_1})$:

Theorem 2 *Let $f(x)$ and $g(x)$ be positive continuous functions on the interval $[a, b]$ with $\int_a^b f(x)dx = \int_a^b g(x)dx$, and assume that there is some $c \in [a, b]$ such that $f(x) \leq g(x)$, for all $x \leq c$ and $f(x) \geq g(x)$, for all $x \geq c$. Then*

$$\int_a^b (x - b)^2 f(x)dx \leq \int_a^b (x - b)^2 g(x)dx \quad \text{and.}$$

$$\int_a^b (x - a)^2 f(x)dx \geq \int_a^b (x - a)^2 g(x)dx$$

Lemma 3 *The variance $d^2(CH(P), l_{b_1})$ is bounded from below by $\frac{10}{243}Ab^2$.*

Proof. We split the integral $\int_0^b (y - b_1)^2 g(y)dy$ at b_1, and prove lower bounds for both parts in the following way: For the left part consider the linear function $f_1(y) = \frac{h_1}{b_1}y$ such that $\int_0^{b_1} f_1(y)dy = \int_0^{b_1} g(y)dy = A_1$ (see Figure 1 (b) for an illustration). From $\int_0^{b_1} f_1(y)dy = A_1$, it follows that $f_1(y) = \frac{2A_1y}{b_1^2}$. Since $g(y)$ is convex, $g(y)$ and $f_1(y)$ intersect only once, at point $b' \in (0, b_1)$. By Theorem 2, we have

$$\begin{aligned} \int_0^{b_1} (y - b_1)^2 g(y)dy &\geq \int_0^{b_1} (y - b_1)^2 f_1(y)dy = \\ \int_0^{b_1} (y - b)^2 \frac{2A_1}{{b_1}^2} dy &= \frac{A_1 b_1^2}{6}. \end{aligned} \qquad (2)$$

Analogously, we obtain

$$\int_{b_1}^b (y - b_1)^2 g(y)dy \quad \geq \quad \frac{A_2 b_2^2}{6}. \qquad (3)$$

From (2) and (3) we obtain that

$d^2(CH(P), l_{b_1}) = \int_0^{b_1} (y-b_1)^2 g(y)dy + \int_{b_1}^{b} (y-b_1)^2 g(y)dy \geq \frac{A_1 b_1^2}{6} + \frac{A_2 b_2^2}{6}$.

From the Grünbaum-Hammer-Mityagin theorem [2], we know that $A_1, A_2 \in [\frac{4}{9}A, \frac{5}{9}A]$. Also, we know that $b_1, b_2 \in [\frac{1}{3}b, \frac{2}{3}b]$. It is not hard to show that, under these constrains, the expression $\frac{A_1 b_1^2}{6} + \frac{A_2 b_1^2}{6}$ achieves its minimum of $\frac{10}{243}Ab^2$ for $A_1 = \frac{4}{9}A, b_1 = \frac{5}{9}b$ or $A_1 = \frac{5}{9}A, b_1 = \frac{4}{9}b$. □

The derivation of the result in the following lemma is similar to that in Lemma 3.

Lemma 4 *The variance $d^2(CH(P), l_{b_1})$ is bounded from above by $\frac{29}{243}Ab^2$.*

Now, we are ready to derive an alternative parametrized upper bound on $\lambda_{2,2}(P)$ which is better than the bound from Lemma 1 for big values of η.

Lemma 5 $\lambda_{2,2}(P) \leq \sqrt{2.9\left(1+\frac{1}{\eta^2}\right)}$ *for any point set P with aspect ratio $\eta(P) = \eta$.*

Proof. Applying Lemma 3 and Lemma 4 in (1) we obtain

$$\frac{10}{243}Ab_{pca}^2 \leq d^2(\mathcal{P}, l_{pca}) \leq d^2(\mathcal{P}, l_{opt}) \leq \frac{29}{243}Ab_{opt}^2. \qquad (4)$$

From (4) it follows that $\beta = \frac{b_{pca}}{b_{opt}} \leq \sqrt{2.9}$. We have for a_{pca} the upper bound $diam(P) \leq \sqrt{a_{opt}^2 + b_{opt}^2} = a_{opt}\sqrt{1+\frac{1}{\eta^2}}$. From this, it follows that $\alpha \leq \sqrt{1+\frac{1}{\eta^2}}$. Putting this together, we obtain $\alpha\beta \leq \sqrt{2.9\left(1+\frac{1}{\eta^2}\right)}$. □

Theorem 6 *The PCA bounding box of a point set P in $\mathbb{R}^2$ computed over $CH(P)$ has a guaranteed approximation factor $\lambda_{2,2} \leq 2.104$.*

Proof. The theorem follows from the combination of the two parametrized bounds from Lemma 1 and Lemma 5:

$$\lambda_{2,2} \leq \sup_{\eta \geq 1}\left\{\min\left(\eta + \frac{1}{\eta}, \sqrt{2.9\left(1+\frac{1}{\eta^2}\right)}\right)\right\}.$$

It is easy to check that the supremum $s \approx 2.1038$ is obtained for $\eta \approx 1.3784$. □

3 An Upper Bound for $\lambda_{3,3}$

Some of the techniques used here are similar to those used in Subsection 2 where we derive an upper bound on $\lambda_{2,2}$. For this reason and due to space limitations, we omit some of the technical proofs (e.g., the derivation of the bounds for the ratios of the longest and the shortest sides of $BB_{pca(3,3)}(P)$ and $BB_{opt}(P)$).

Given a point set $P \subseteq \mathbb{R}^3$ and an arbitrary bounding box $BB(P)$, we will denote the three side lengths of $BB(P)$ by a,b and c, where $a \geq b \geq c$. We are interested in the side lengths $a_{opt} \geq b_{opt} \geq c_{opt}$ and $a_{pca} \geq b_{pca} \geq c_{pca}$ of $BB_{opt}(P)$ and $BB_{pca(3,3)}(P)$. The parameters $\alpha = \alpha(P) = a_{pca}/a_{opt}$, $\beta = \beta(P) = b_{pca}/b_{opt}$ and $\gamma = \gamma(P) = c_{pca}/c_{opt}$ denote the ratios between the corresponding side lengths. Hence, we have $\lambda_{3,3}(P) = \alpha \cdot \beta \cdot \gamma$. We introduce two additional parameters $\eta(P) = a_{opt}/b_{opt}$ and $\theta(P) = a_{opt}/c_{opt}$.

Lemma 7 $\lambda_{3,3}(P) \leq \eta\,\theta\left(1+\frac{1}{\eta^2}+\frac{1}{\theta^2}\right)^{\frac{3}{2}}$ *for any point set P with aspect ratios $\eta(P) = \eta$ and $\theta(P) = \theta$.*

Proof. We have for a_{pca}, b_{pca} and c_{pca} the upper bound $diam(P) \leq \sqrt{a_{opt}^2 + b_{opt}^2 + c_{opt}^2} = a_{opt}\sqrt{1+\frac{1}{\eta^2}+\frac{1}{\theta^2}}$. Thus, $\alpha\,\beta\,\gamma \leq \frac{a_{pca}\,b_{pca}\,c_{pca}}{a_{opt}\,b_{opt}\,c_{opt}} \leq \frac{a_{opt}^3\left(1+\frac{1}{\eta^2}\right)^{\frac{3}{2}}}{a_{opt}b_{opt}c_{opt}}$. Replacing a_{opt} in the nominator once by $\eta\,b_{opt}$ and once by $\theta\,c_{opt}$ we obtain $\lambda_{3,3}(P) \leq \eta\,\theta\left(1+\frac{1}{\eta^2}+\frac{1}{\theta^2}\right)^{\frac{3}{2}}$. □

Unfortunately, this parametrized upper bound tends to infinity for $\eta \to \infty$ or $\theta \to \infty$. Therefore we are going to derive another upper bound that is better for large values of η and θ. We derive such a bound by finding constants that bound β and γ from above. In this process we will make essential use of the properties of $BB_{pca(3,3)}(P)$. We denote by $d^2(CH(P), H)$ the integral of the squared distances of the points on $CH(P)$ to a plane H, i.e., $d^2(CH(P), H) = \int_{s \in CH(P)} d^2(s, H)ds$. Let H_{pca} be the plane going through the center of gravity, parallel to the side $a_{pca} \times b_{pca}$ of $BB_{pca(3,3)}(P)$, and H_{opt} be the bisector of $BB_{opt(P)}$ parallel to the side $a_{opt} \times b_{opt}$. From principal component analysis we know that H_{pca} is the best fitting plane of P and therefore

$$d^2(CH(P), H_{pca}) \leq d^2(CH(P), H_{opt}). \qquad (5)$$

We obtain an estimation for β by determining a lower bound on $d^2(CH(P), H_{pca})$ that depends on b_{pca}, and an upper bound on $d^2(CH(P), H_{opt})$ that depends on b_{opt}. Having an arbitrary bounding box of $CH(P)$ (with side lengths a, b, and c, $a \geq b \geq c$) the volume of $CH(P)$ can be expressed as

$$V = V(CH(P)) = \int_0^c \int_0^b \int_0^a \chi_{CH(P)}(x,y,z)dxdydz = \int_0^c g(z)dz,$$

where $\chi_{CH(P)}(x,y,z)$ is the *characteristic function* of $CH(P)$ defined as

$$\chi_{CH(P)}(x,y,z) = \begin{cases} 1 & (x,y,z) \in CH(P) \\ 0 & (x,y,z) \notin CH(P), \end{cases}$$

and $g(z) = \int_0^b \int_0^a \chi_{CH(P)}(x,y,z)dxdy$ is the area of the intersection of $CH(P)$ with the horizontal plane at height z. As before we call $g(z)$ the *density function* of $CH(P)$. Let c_1 denote the z-coordinate of the center of gravity of $CH(P)$. Note that $g(z)$ is continuous, but in general not convex in the interval $[0,b]$. Therefore, we can not use linear functions to derive a lower and an upper bound of the function $d^2(CH(P), H_{ab})$, as we did in Section 2, because a linear function can intersect $g(z)$ more than once, and we can not apply Theorem 2. Instead of linear functions, we use quadratic functions.

Proposition 8 *Let $g(z)$ be the density function of $CH(P)$ defined as above, and let $f(z) = kz^2$ be the parabola such that $\int_0^{c_1} f(z)dz = \int_0^{c_1} g(z)dz$. Then, $\exists c_0 \in [0, c_1]$ such that $f(z) \le g(x)$ for all $z \le c_0$ and $f(z) \ge g(z)$ for all $z \ge c_0$.*

Proof. We give a constructive proof. Let $c_0 := \inf\{d \,|\, \forall z \in [d, c_1]\ g(z) \le f(z)\}$. If $c_0 = 0$, then $f(z) = g(z)$, and the proposition holds. If $c_0 > 0$, then consider the polygon which is the intersection of $CH(P)$ with the plane $z = c_0$. We fix a point p_0 in $CH(P)$ with z-coordinate 0 and construct a pyramid Q by extending all rays from p_0 through the polygon up to the plane $z = c_1$ (see Figure 2 for an illustration). Since, $f(c_0) = g(c_0)$ the quadratic function

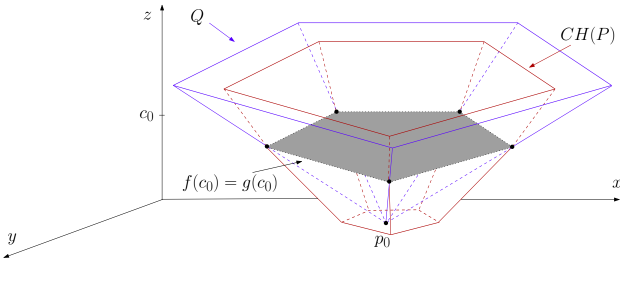


Figure 2: Construction of the intersection of $f(z)$ and $g(z)$.

$f(z)$ is the density function of Q. Therefore, since the part of Q below c_0 is completely included in the $CH(P)$, we can conclude that $f(z) \le g(z)$ for all $z \le c_0$. On the other side, $f(z) \ge g(x)$ for all $z \ge c_0$ by the definition of c_0. $\square$

Using Proposition 8, we can derive a lower and an upper bound on $d^2(CH(P), H_{ab})$ (Lemma 9 and Lemma 10), from which we can derive a new parametrized bound on $\lambda_{3,3}(P)$, (Lemma 11). We exploit similar ideas as those shown in Lemma 3 and Lemma 5, therefore we leave the proofs of Lemma 9, Lemma 10 and Lemma 11 to a full paper version.

Lemma 9 *The variance $d^2(CH(P), H_{ab})$ is bounded from below by $\frac{7}{256}Vc^2$.*

Lemma 10 *The variance $d^2(CH(P), H_{ab})$ is bounded from above by $\frac{12729}{71680}Vc^2$.*

Lemma 11 $\lambda_{3,3}(P) \le 6.43\sqrt{1 + \frac{1}{\eta^2} + \frac{1}{\theta^2}}$ *for any point set P with aspect ratios $\eta(P) = \eta$ and $\theta(P) = \theta$.*

Lemma 7 gives us a bound on $\lambda_{3,3}(P)$ which is good for small values of η and θ. In contrary, the bound from Lemma 11 behaves worse for small values of η and θ, but better for big values of η and θ. Therefore, we combine both of them to obtain the final upper bound.

Theorem 12 *The PCA bounding box of a point set P in $\mathbb{R}^3$ computed over $CH(P)$ has a guaranteed approximation factor $\lambda_{3,3} \le 7.72$.*

Proof. The theorem follows from the combination of the two parametrized bounds from Lemma 7 and Lemma 11:

$$\lambda_{3,3} \le \sup_{\eta \ge 1, \theta \ge 1} \left\{ \min\left(\eta\,\theta\left(1 + \frac{1}{\eta^2} + \frac{1}{\theta^2}\right)^{\frac{3}{2}}, 6.43\sqrt{1 + \frac{1}{\eta^2} + \frac{1}{\theta^2}} \right) \right\}.$$

By numerical verification we obtained that the supremum occurs at ≈ 7.72. $\square$

4 Future Work and Open Problems

Improving the upper bound for $\lambda_{3,3}$, $\lambda_{2,2}$ and $\lambda_{2,1}$, as well as obtaining an upper bound for $\lambda_{3,2}$ is of interest. A very demanding open problem is to get an approximation factor on the quality of PCA bounding boxes in higher dimensions.

References

[1] D. Dimitrov, C. Knauer, K. Kriegel and G. Rote. Upper and Lower Bounds on the Quality of the PCA Bounding Boxes. *Technical report B 06-10, Freie Universität Berlin*, September 2006.

[2] B. Grünbaum. Partitions of mass-distributions and convex bodies by hyperplanes. Pacific J. Math. 1960, vol. 10, pages 1257–1261.

[3] I. Jolliffe. Principal Component Analysis. *Springer-Verlag, New York, 2nd ed.*, 2002.

Kinetic kd-Trees

Mohammad Ali Abam* Mark de Berg* Bettina Speckmann*

Abstract

We propose a simple variant of kd-trees, called rank-based kd-trees, for sets of points in $\mathbb{R}^d$. We show that a rank-based kd-tree, like an ordinary kd-tree, supports range search queries in $O(n^{1-1/d} + k)$ time, where k is the output size. The main advantage of rank-based kd-trees is that they can be efficiently kinetized: the KDS processes $O(n^2)$ events in the worst case, assuming that the points follow constant-degree algebraic trajectories, each event can be handled in $O(\log n)$ time, and each point is involved in $O(1)$ certificates.

1 Introduction

Background. Due to the increased availability of GPS systems and to other technological advances, motion data is becoming more and more available in a variety of application areas: air-traffic control, mobile communication, geographic information systems, and so on. In many of these areas, the data are moving points in 2- or higher-dimensional space, and what is needed is to store these points in such a way that *range queries* ("Report all the points lying currently inside a query range") can be answered efficiently. Hence, there has been a lot of work on developing data structures for moving point data, both in the database community as well as in the computational-geometry community.

Within computational geometry, the standard model for designing and analyzing data structures for moving objects is the kinetic-data-structure framework introduced by Basch et al. [3]. A *kinetic data structure* (KDS) maintains a discrete attribute of a set of moving objects—the convex hull, for example, or the closest pair—where each object has a known motion trajectory. The basic idea is, that although all objects move continuously, there are only certain discrete moments in time when the combinatorial structure of the attribute—the ordered set of convex-hull vertices, or the pair that is closest—changes. A KDS contains a set of *certificates* that constitutes a proof that the maintained structure is correct. These certificates are inserted in a priority 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 an *event* happens, that is, when a certificate fails—see the surveys by Guibas [6, 7] for more details.

Related work. There are several papers that describe KDS's for the orthogonal range-searching problem, where the query range is an axis-parallel box. Basch et al. [4] kinetized d-dimensional range trees. Their KDS supports range queries in $O(\log^d n + k)$ time and uses $O(n \log^{d-1} n)$ storage. If the points follow constant-degree algebraic trajectories then their KDS processes $O(n^2)$ events and each event can be handled in $O(\log^{d-1} n)$ time. In the plane, Agarwal et al. [1] obtained an improved solution: their KDS supports range searching queries in $O(\log n + k)$ time, it uses $O(n \log n / \log\log n)$ storage, and the amortized cost of processing an event is $O(\log^2 n)$.

Although these results are nice from a theoretical perspective, their practical value is limited for several reasons. First of all, they use super-linear storage, which is often undesirable. Second, they can perform only orthogonal range queries; queries with other types of ranges or nearest-neighbor searches are not supported. Finally, especially the solution by Agarwal et al. [1] is rather complicated. Indeed, in the setting where the points do not move, the static counterparts of these structures are usually not used in practice. Instead, simpler structures such as quadtrees, kd-trees, or bounding-volume hierarchies (R-trees, for instance) are used. In this paper we consider one of these structures, namely the kd-tree.

Kd-trees were initially introduced by Bentley [5]. A kd-tree for a set of points in the plane is obtained recursively as follows. At each node of the tree, the current point set is split into two equal-sized subsets with a line. When the depth of the node is even the splitting line is orthogonal to the x-axis, and when it is odd the splitting line is orthogonal to the y-axis. In d-dimensional space, the orientations of the splitting planes cycle through the d axes in a similar manner. Kd-trees use $O(n)$ storage and support range searching queries in $O(n^{1-1/d} + k)$ time, where k is the number of reported points. Maintaining a standard kd-tree kinetically is not efficient. The problem is that a single event—two points swapping their order

*Department of Mathematics and Computer Science, TU Eindhoven, {mabam, mdberg, speckman}@win.tue.nl. M.A. was 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.

on x- or y-coordinate—can have a dramatic effect: a new point entering the region corresponding to a node could mean that almost the entire subtree must be restructured. Hence, a variant of the kd-tree is needed when the points are moving.

Agarwal et al. [2] proposed two such variants: the δ-pseudo kd-tree and the δ-overlapping kd-tree. In a δ-pseudo kd-tree each child of a node ν can be associated with at most $(1/2 + \delta)n_\nu$ points, where n_ν is the number of points in the subtree of ν. In a δ-overlapping kd-tree the regions corresponding to the children of ν can overlap as long as the overlapping region contains at most δn_ν points. Both kd-trees support range queries in time $O(n^{1/2+\varepsilon} + k)$, where k is the number of reported points. Here ε is a positive constant that can be made arbitrarily small by choosing δ appropriately. These KDS's process $O(n^2)$ events if the points follow constant-degree algebraic trajectories. Although it can take up to $O(n)$ time to handle a single event, the amortized cost is $O(\log n)$ time per event. Neither of these two solutions are completely satisfactory: their query time is worse by a factor $O(n^\varepsilon)$ than the query time in standard kd-trees, there is only a good amortized bound on the time to process events, and only a solution for the 2-dimensional case is given. The goal of our paper is to developed a kinetic kd-tree variant that does not have these drawbacks.

Our results. We present a new and simple variant of the standard kd-tree for a set of n points in d-dimensional space. Our *rank-based kd-tree* supports orthogonal range searching in time $O(n^{1-1/d}+k)$ and it uses $O(n)$ storage—just like the original. But additionally it can be kinetized easily and efficiently. The rank-based kd-tree processes $O(n^2)$ events in the worst case if the points follow constant-degree algebraic trajectories and each event can be handled in $O(\log n)$ worst-case time. Moreover, each point is involved only in a constant number of certificates. Thus we improve the both the query time and the event-handling time as compared to the planar kd-tree variants of Agarwal et al. [2], and in addition our results work in any fixed dimension.

2 Rank-based kd-trees

Let $\mathcal{P}$ be a set of n points in $\mathbb{R}^d$ and let us denote the coordinate-axes with $x_1, \cdots, x_d$. To simplify the discussion we assume that no two points share any coordinate, that is, no two points have the same x_1-coordinate, or the same x_2-coordinate, etc. (Of course coordinates will temporarily be equal when two points swap their order, but the description below refers to the time intervals in between such events.) In this section we describe a variant of a kd-tree for $\mathcal{P}$, the *rank-based kd-tree*. A rank-based kd-tree preserves all main properties of a kd-tree and, additionally, it can be kinetized efficiently.

Before we describe the actual rank-base kd-tree for $\mathcal{P}$, we first introduce another tree, namely the *skeleton* of a rank-base kd-tree, denoted by $\mathcal{S}(\mathcal{P})$. Like a standard kd-tree, $\mathcal{S}(\mathcal{P})$ uses axis-orthogonal splitting hyperplanes to divide the set of points associated with a node. As usual, the orientation of the axis-orthogonal splitting hyperplanes is alternated between the coordinate axes, that is, we first split with a hyperplane orthogonal to the x_1-axis, then with a hyperplane orthogonal to the x_2-axis, and so on. Let ν be node of $\mathcal{S}(\mathcal{P})$. $h(\nu)$ is the splitting hyperplane stored at ν, axis(ν) is the coordinate-axis to which $h(\nu)$ is orthogonal, and $\mathcal{P}(\nu)$ is the set of points stored in the subtree rooted at ν. A node ν is called an *x_i-node* if axis$(\nu) = x_i$ and a node ω is referred to as an *x_i-ancestor* of a node ν if ω is an ancestor of ν and axis$(\omega) = x_i$. The first x_i-ancestor of a node ν is the *x_i-parent(ν)* of ν.

A standard kd-tree chooses $h(\nu)$ such that $\mathcal{P}(\nu)$ is divided roughly in half. In contrast, $\mathcal{S}(\mathcal{P})$ chooses $h(\nu)$ based on a range of ranks associated with ν, which can have the effect that the sizes of the children of ν are completely unbalanced. We now explain this construction in detail. We use d arrays $\mathcal{A}_1, \cdots, \mathcal{A}_d$ to store the points of $\mathcal{P}$ in d sorted lists; the array $\mathcal{A}_i[1, n]$ stores the sorted list based on the x_i-coordinate. As mentioned above, we associate a range $[r, r']$ of ranks with each node ν, denoted by range(ν), with $1 \le r \le r' \le n$. Let ν be an x_i-node. If x_i-parent(ν) does not exist, then range(ν) is equal to $[1, n]$. Otherwise, if ν is contained in the left subtree of x_i-parent(ν), then range(ν) is equal to the first half of range(x_i-parent(ν)), and if ν is contained in the right subtree of x_i-parent(ν), then range(ν) is equal to the second half of range(x_i-parent(ν)). If range$(\nu) = [r, r']$ then $\mathcal{P}(\nu)$ contains at most $r' - r + 1$ points. We explicitly ignore all nodes (both internal as well as leaf nodes) that do not contain any points, they are not part of $\mathcal{S}(\mathcal{P})$, independent of their range of ranks. A node ν is a leaf of $\mathcal{S}(\mathcal{P})$ if range$(\nu) = [j, j]$ for some j. Clearly a leaf contains exactly one point, but not every node that contains only one point is a leaf. (We could prune these nodes, which always have a range $[j, k]$ with $j < k$, but we chose to keep them in the skeleton for ease of description.) If ν is not a leaf and axis$(\nu) = x_i$ then $h(\nu)$ is defined by the point whose rank in $\mathcal{A}_i$ is equal to the median of range(ν).

We construct $\mathcal{S}(\mathcal{P})$ incrementally by inserting the points of $\mathcal{P}$ one by one. Let p be the point that we are currently inserting into the tree and let ν be the last node visited by p; initially ν = root. Depending on which side of $h(\nu)$ contains p we select the appropriate child ω of ν to be visited next. If ω does not exist, then we create it and compute range(ω) as described above. We recurse with $\nu = \omega$ un-

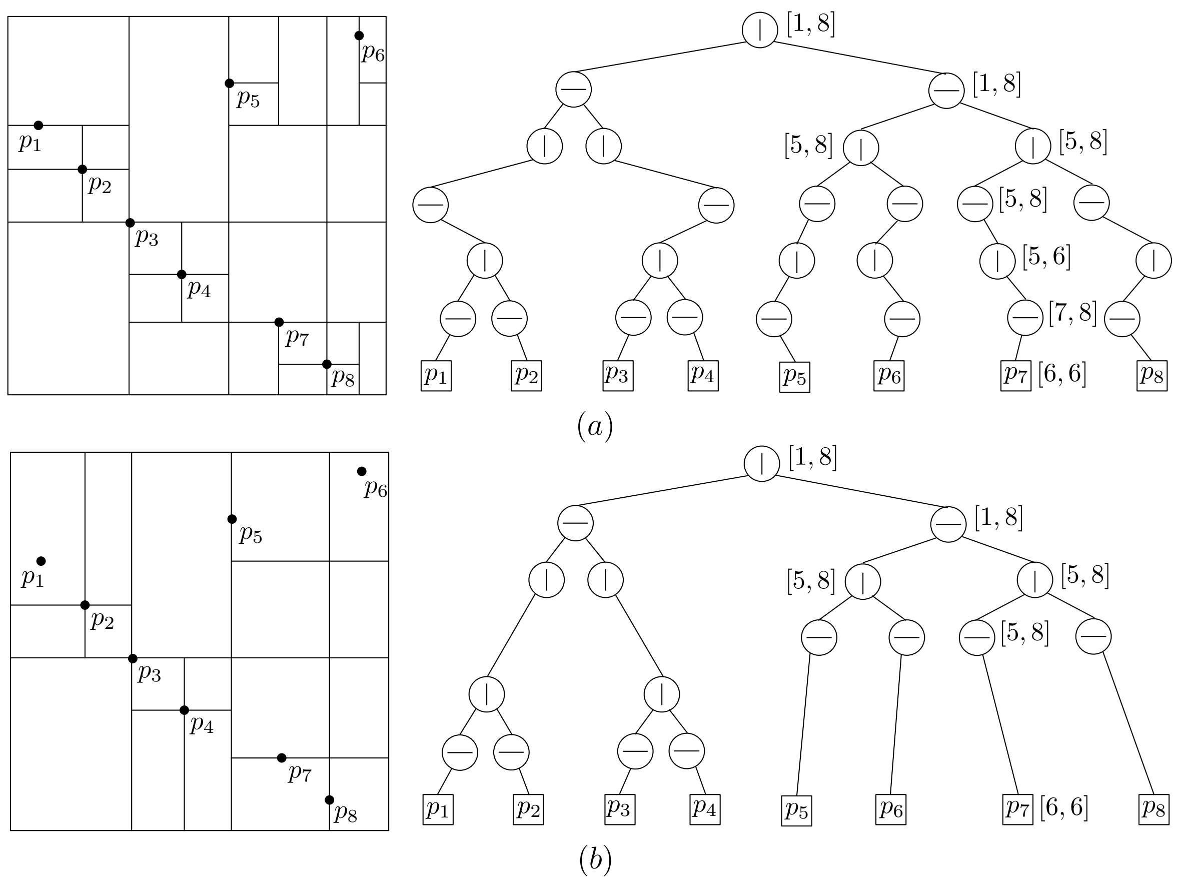


Figure 1: (a) The skeleton of a rank-based kd-tree and (b) the rank-based kd-tree itself.

til range(ν) $=$ $[j, j]$ for some j. We always reach such a node after $d \log n$ steps, because the length of range(ν) is a half of the length of range(x_i-parent(ν)) and depth(ν) $=$ depth(x_i-parent(ν)) $+\, d$ for an x_i-node ν. Figure 1(a) illustrates $\mathcal{S}(\mathcal{P})$ for eight points. Since the depth of each leaf is $d \log n$, the size of $\mathcal{S}(\mathcal{P})$ is $O(n \log n)$.

Lemma 1 *The depth of $\mathcal{S}(\mathcal{P})$ is $O(\log n)$ and the size of $\mathcal{S}(\mathcal{P})$ is $O(n \log n)$ for any fixed dimension d. $\mathcal{S}(\mathcal{P})$ can be constructed in $O(n \log n)$ time.*

A node $\nu \in \mathcal{S}(\mathcal{P})$ is *active* if and only if both its children exist, that is, both its children contain points. A node ν is *useful* if it is either active, or a leaf, or its first $d-1$ ancestors contain an active node. Otherwise a node is *useless*. We derive the rank-based kd-tree for $\mathcal{P}$ from the skeleton by pruning all useless nodes from $\mathcal{S}(\mathcal{P})$. The parent of a node ν in the rank-based kd-tree is the first unpruned ancestor of ν in $\mathcal{S}(\mathcal{P})$. Roughly speaking, in the pruning phase every long path whose nodes have only one child each is shrunk to a path whose length is less than d. The rank-based kd-tree has exactly n leaves and each contains exactly one point of $\mathcal{P}$. Moreover, every node ν in the rank-based kd-tree is either active or it has an active ancestor among its first $d-1$ ancestors. The rank-based kd-tree derived from Figure 1(a) is illustrated in Figure 1(b).

Lemma 2 *A rank-based kd-tree on a set of n points in $\mathbb{R}^d$ has depth $O(\log n)$ and size $O(n)$.*

Proof. A rank-based kd-tree is at most as deep as its skeleton $\mathcal{S}(\mathcal{P})$. Since the depth of $\mathcal{S}(\mathcal{P})$ is $O(\log n)$ by Lemma 1, the depth of a rank-base kd-tree is also $O(\log n)$. To prove the second claim, we charge every node that has only one child to its first active ancestor—recall that each active node has two children. We charge at most $2(d-1)$ nodes to each active node, because after pruning there is no path in the rank-based kd-tree whose length is at least d and in which all nodes have one child. Therefore, to bound the size of the rank-based kd-tree it is sufficient to bound the number of active nodes. Let $\mathcal{T}$ be a tree containing all active nodes and all leaves of the rank-based kd-tree. A node ν is the parent of a node ω in $\mathcal{T}$ if and only if ν is the first active ancestor of ω in the rank-based kd-tree. Obviously, $\mathcal{T}$ is a binary tree with n leaves where each internal node has two children. Hence, the size of $\mathcal{T}$ is $O(n)$ and consequently the size of the rank-based kd-tree is $O(n)$. $\square$

Like a kd-tree, a rank-based kd-tree can be used to report all points inside a given orthogonal range search query—the reporting algorithm is exactly the same. At first sight, the fact that the splits in our rank-based kd-tree can be very unbalanced may seem to have a big, negative impact on the query time. Fortunately this is not the case, since we can bound the number of cells intersected by an axis-parallel plane h. The following theorem summarizes our results.

Theorem 3 *A rank-based kd-tree for a set $\mathcal{P}$ of n points in d dimensions uses $O(n)$ storage and can be built in $O(n \log n)$ time. An orthogonal range search query on a rank-based kd-tree takes $O(n^{1-1/d} + k)$ time where k is the number of reported points.*

The KDS. We now describe how to kinetize a rank-base kd-tree for a set of continuously moving points $\mathcal{P}$. The combinatorial structure of a rank-base kd-tree depends only on the ranks of the points in the arrays $\mathcal{A}_i$, that is, it does not change as long as the order of the points in the arrays $\mathcal{A}_i$ remains the same. Hence it suffices to maintain a certificate for each pair p and q of consecutive points in every array $\mathcal{A}_i$, which fails when p and q change their order. Now assume that a certificate, involving two points p and q and the x_i-axis, fails at time t. To handle the event, we simply delete p and q and re-insert them in their new order. These deletions and insertions do not change anything for the other points, because their ranks are not influenced by the swap and the deletion and re-insertion of p and q. Hence the rank-based kd-tree remains unchanged except for a small part that involves p and q. A detailed description of this “small part” can be found below.

Deletion. Let ν be the first active ancestor of the leaf μ containing p—see Figure 2(a). Leaf μ and all nodes on the path from μ to ν must be deleted, since they do not contain any points anymore (they only contained p and p is now deleted). Furthermore, ν stops being active. Let ω be the first active descendent of ν. There are at most d nodes on the path from ν to ω. Since ν is not active anymore, any of the nodes on this path might become useless and hence have to be deleted.

Insertion. Let ν be the highest node in the rank-based kd-tree such that its region contains p and the region corresponding to its only child ω does not contain p—note that p cannot reach a leaf when we re-insert p, because the range of a leaf is $[j, j]$ for some j and there cannot be two points in this range. Let ν' and ω' be the nodes in $\mathcal{S}(P)$ corresponding to ν and ω. Let u' be the lowest node on the path from ν' to ω' whose region contains both region(ω') and p as illustrated in Figure 2(b)—note that we do not maintain $\mathcal{S}(\mathcal{P})$ explicitly but with the information maintained in ν and ω the path between ν' and ω' can

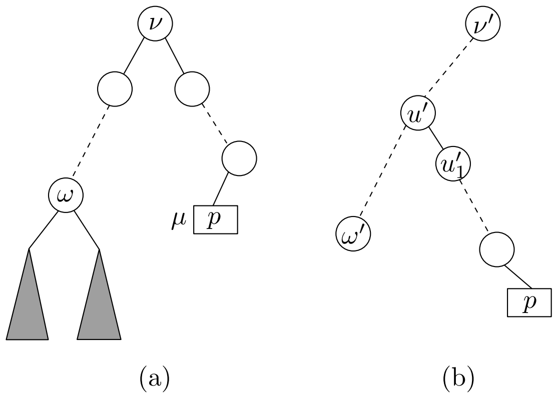


Figure 2: Inserting and deleting a point p.

be constructed temporarily. Because u' will become an active node, it must be added to the rank-based kd-tree and also every node on the path from u' to ω' must be added to the rank-based kd-tree if they are useful. From u', the point p follows a new path $u'_1, \cdots, u'_k$ which is created during the insertion. All first $d-1$ nodes in the list $u'_1, \cdots, u'_k$ and the leaf u'_k must be added to the rank-based kd-tree—note that range(u'_k) = $[j, j]$ for some j.

Theorem 4 *A kinetic rank-based kd-tree for a set $\mathcal{P}$ of n moving points in d dimensions uses $O(n)$ storage and processes $O(n^2)$ events in the worst case, assuming that the points follow constant-degree algebraic trajectories. Each event can be handled in $O(\log n)$ time and each point is involved in $O(1)$ certificates.*

References

[1] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66(1):207-243, 2003.

[2] P. Agarwal, J. Gao, and L. Guibas. Kinetic medians and kd-trees. In *Proc. 10th European Symposium on Algorithms*, pages 5–16, Lecture Notes in Computer Science 2461, Springer Verlag, 2002.

[3] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31:1–28, 1999.

[4] J. Basch, L. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Symposium on Computational Geometry*, pages 344–351, 1997.

[5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] L. Guibas. Kinetic data structures: A state of the art report. In *Proc. 3rd Workshop on Algorithmic Foundations of Robotics*, pages 191–209, 1998.

[7] L. Guibas. Modeling motion. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1117–1134. CRC Press, 2nd edition, 2004.

Approximating Boundary-Triangulated Objects with Balls*

O. Aichholzer† F. Aurenhammer‡ T. Hackl† B. Kornberger† M. Peternell§ H. Pottmann§

Abstract

We compute a set of balls that approximates a given 3D object, and we derive small additive bounds for the overhead in balls with respect to the minimal solution with the same quality. The algorithm has been implemented and tested using the CGAL library [7].

1 Introduction

Representing a complex geometric object with primitives is a fundamental task in computational geometry and computer graphics. A general distinction is between exact representation, like slab decomposition or triangulation, and approximate representation, like spline boundary conversion or approximate covering. The choice of the primitives used depends on the application, for example, whether the object is to be processed for visualization or for subsequent manipulation.

In the present note, we deal with the problem of converting a 3D object into a set of balls. While such a representation may be less advantageous for visualization (than, e.g., a conversion into ellipsoids [4]) it is sometimes particularly convenient for further processing. The main applications we have in mind are computing Minkowski sums and detecting collisions.

Calculating the Minkowski sum of two nonconvex 3D objects is a complicated task with various applications to problems where proximity is involved [14]. For instance, planning a translational motion of a robot A in a workspace B can be based on constructing their Minkowski sum $A \oplus B$. A common approach is to decompose the (polyhedral) objects A and B into convex parts and adding them up pairwise. Using few but complex parts leads to the need of invoking Minkowski sum algorithms for general convex polyhedra [10, 8], and decomposition is a challenging question in itself even when tetrahedra are used [6, 18]. As an alternative, the objects to be summed may be approximately covered by balls. Calculating the Minkowski sum of two balls is trivial; the main objective is finding a small set of covering balls.

In collision detection, object handling is typically based on hierarchical representations. Common data structures in this context are so-called sphere-trees which, as an easy possibility, can be derived from octrees [17, 16]. This approach does not make explicit use of the geometry of the objects, however. Refined methods for constructing sphere-trees have been proposed [11, 5], utilizing the medial axes of the objects. Again, the problem of converting an object into a minimum number of balls arises. The reverse process of extracting the boundary of an approximating set of balls is treated in [3].

We develop an algorithm that takes as input a 3D object with triangulated boundary, and generates an almost-minimal set of balls that covers all the triangle endpoints, ensuring that no such point is covered with more than a user-specified offset ε. The quality of the approximation thus will also depend on the quality of the boundary mesh. If the object boundary is not covered completely, this can be achieved if desired, with a simple postprocessing step. Following known paths [1, 2], we first generate a candidate set of approximating balls centered on the triangles endpoints' Voronoi diagram. Correct labeling of balls as having their centers inside or outside the object becomes an issue, and we propose a simple though efficient labeling algorithm using the boundary triangulation. A method for reducing the candidate set of balls is then applied as an instance of the set covering problem. The heuristic we use allows us to determine how close to the optimum is the produced set of balls. Experimental results for practical data are described.

2 Ball generation

We specify the input object, A, as a bounded and interior-connected 3-manifold whose boundary is connected and triangulated. Thus A may have tunnels, but holes are disallowed. For each boundary triangle of A its orientation with respect to A is given. This is a common representation of an object, sufficiently general for many applications. Let P be the set of vertices of A, called *sample points* in the following. Our first aim is to produce a candidate set of balls which covers P and at the same time approximates A.

Call a ball $b \subseteq A$ *maximal* if there exists no ball $b' \subseteq A$ such that $b' \supsetneq b$. The medial axis of A

*Research supported by the FWF Joint Research Project 'Industrial Geometry' S9205-N12.

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

‡Institute for Theoretical Computer Science, Graz University of Technology, Austria, auren@igi.tugraz.at

§Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Austria, {peternell,pottmann}@geometrie.tuwien.ac.at

is defined as the set of centers of all its maximal balls. As A is just the union of all maximal balls, a set of n sufficiently large balls centered close to the medial axis of A will serve the desired purpose. This observation has been made use of in various papers in computational geometry and computer graphics; see, for example, [11, 5] and [1, 2], respectively.

Following the approach in [2], we consider the Voronoi diagram, $V(P)$, of the given point sample P. For a point $p \in P$, let π_p be an *(inner) pole* of p, that is, a vertex of the region of p in $V(P)$ that lies inside A and maximizes $\|p - \pi_p\|$. In contrast to arbitrary Voronoi vertices, the value of poles is that they are located near the medial axis of A if the sample P is sufficiently dense [1]. (Let us temporarily ignore the fact that π_p does not exist if all region vertices for p happen to lie outside of A.) If we take, for each $p \in P$, the (closed) Delaunay ball with center π_p and radius $\|p - \pi_p\|$ then the resulting set, B, of balls covers P. Also, B approximates A with a quality which depends on that of the boundary mesh. Observe that B is optimal in the sense that any other set of balls which 'touches' P and is of the same cardinality will be inferior to B in approximating A.

Identifying poles among the vertices of a Voronoi diagram is a problem in itself. In [2], for the sake of subsequent power crust construction, two vertices per sample point are identified using an angle criterion that works for dense samplings. (At most) one of these two vertices is located in A and is the pole we are looking for. We need to exactly find the poles, as balls centered outside A will lead to a violation of any approximation property. To this end, we utilize the given triangular mesh that bounds A (which is not part of the input in [2]). After having computed all the vertices of $V(P)$, we use ray shooting to determine their location with respect to A. Among those lying inside A, one vertex per region which is at maximal distance from the respective sample point is selected. This direct method will work correctly regardless of the quality of the sample mesh.

The outcome of the ray shooting procedure is crucial, hence a correct and efficient implementation is needed. To decide $u \in A$ for a vertex u of point p's Voronoi region, we use the ray $\overrightarrow{up}$ and determine the first point of intersection of $\overrightarrow{up}$ with the boundary of A. Clearly, if this point is p, then decision can be made locally from the orientations of the incident triangles. Otherwise, the boundary triangle hit first is not incident to p, and we adopt the following strategy for finding it. Let S_u be the sphere with center u and radius $\|p - u\|$. Define a *critical sphere* S_p centered at p, as below. Let L be the length of the longest edge of the boundary mesh, and put $R = \frac{1}{\sqrt{3}}L$. If $\|p - u\| \geq R$ then choose S_p so as to intersect S_u in a circle of radius R. Otherwise, define the radius of S_p as $\sqrt{\|p - u\|^2 + R^2}$. Consult Figure 1.

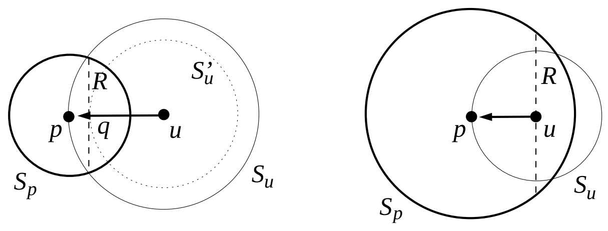


Figure 1: Defining the critical sphere S_p

Lemma 1 *If the ray $\overrightarrow{up}$ intersects a boundary triangle Δ of A then at least one endpoint of Δ is enclosed by S_p.*

Proof. Let Δ be intersected by $\overrightarrow{up}$. Consider the smallest disk, D_Δ, that contains Δ. All edges of Δ are of length at most L, so the radius of D_Δ is at most $R = \frac{1}{\sqrt{3}}L$. We treat the case $\|p - u\| \geq R$ first. As the (Delaunay) sphere S_u does not enclose any endpoint of Δ, we get that Δ intersects $\overrightarrow{up}$ between p and the center, q, of the circle $S_u \cap S_p$. On the other hand, assuming that no endpoint of Δ is enclosed by S_p implies that Δ intersects $\overrightarrow{up}$ between q and u, a contradiction. Now let $\|p - u\| < R$. In this case, the plane normal to $\overrightarrow{up}$ at u intersects S_p in a circle of radius R. Assuming that S_p encloses no endpoint of Δ now implies that Δ avoids the ray $\overrightarrow{up}$ altogether, a contradiction again. □

By Lemma 1, the first triangle hit by $\overrightarrow{up}$ can be detected once the subset Q of P enclosed by the critical sphere S_p has been reported. This spherical range search problem has a practically efficient implementation based on kd-trees. Observe that the radius of S_p is $\Theta(L)$; it lies between R and $\sqrt{2} \cdot R$. Thus only $O(1)$ points will be reported if P obeys a minimum distance of $c \cdot L$ for some constant c.

To facilitate later decisions $v \in A$, we make use of the following property, which holds if the (practically more relevant) case $\|p - u\| \geq \frac{1}{\sqrt{3}}L$ occurs for the present vertex u. See Figure 1, left-hand side.

Lemma 2 *Let S'_u be the sphere centered at u and passing through the center of the circle $S_u \cap S_p$. All Voronoi vertices v enclosed by S'_u have the same relative position to A as the vertex u.*

Proof. By the bound on the smallest containing disk for a mesh triangle, no such triangle intersects the sphere S'_u. Thus S'_u either lies completely inside or completely outside of A. Also, S'_u does not enclose A because S'_u is empty of points from P, as is S_u. □

The output is a set of Delaunay balls for P whose union approximates the object A. We observed a runtime linear in $|P|$ in all our examples. In particular, the number of vertices of $V(P)$ stayed below $9 \cdot |P|$. Each produced ball covers at least four points in P with its boundary. There may be uncovered points (at rare cases), due to the lack of their poles. Such points are added to the set as balls of radius zero. In order to be able to delete a large fraction of balls later on, we increase redundancy in covering by enlarging the radius of each ball by a user-specified constant ε. Each point in P is now covered with an offset of at most ε. It will turn out that the obtained set of balls is highly redundant even for small offsets ε.

3 Reduction algorithm

Let B_ε denote the set of ε-offset balls produced in Section 2. We aim at finding a subset of B_ε of minimal cardinality that still covers the set P of sample points. This is an instance of the classical set covering problem, shown to be NP-complete in [13]. We are going to describe a hybrid heuristic that reduces B_ε almost to the optimum in many cases, and that allows to bound the produced overhead.

In a first step, we arrange P in a kd-tree and determine which points of P are covered by which balls in B_ε. The result is stored in an incidence matrix, where entry (i, j) is put to 1 or 0 depending on whether the i-th ball contains the j-th point. Standard reduction rules are then applied iteratively to the rows and columns of this matrix: (1) If column j has exactly one entry 1, say (i, j), then delete row i and all columns having entry 1 in that row. Sphere i has to be taken for any solution. (2) If row i_1 is dominated (in 1s) by row i_2 then delete row i_1. All points covered by sphere i_1 are also covered by sphere i_2. (3) If column j_1 dominates column j_2 then delete column j_1. Every solution that covers point j_2 has to cover point j_1 as well. We speed up these operations by using hashing techniques, and avoid excessive storage by sparse matrix representation.

If the reduced matrix is nonempty, we try to decompose it into independent submatrices. These are given by the connected components in the corresponding (bipartite) incidence graph. For each submatrix M, if small enough, the respective set covering problem is solved exactly, using branch-and-bound [15] for its integer programming formulation: Minimize

$$\sum x_j \text{ w.r.t. } M^T \cdot x \geq (1, \ldots, 1)^T, \; x_j \in \{0, 1\}.$$

Whereas up to this point optimality is ensured, we have to resort to a heuristic for approximating subproblems too large for exact solving. To this end, we next choose one ball covering the most yet uncovered points. When applied repeatedly, this is a simple greedy algorithm, yielding an $O(\log n)$ approximation [12]. In fact, the set covering problem is unlikely to be approximable beyond a factor of $c \log n$ in polynomial time; see [9]. Here n denotes the number of covering sets (i.e., balls in our case). Our algorithm, after each single greedy pick, runs through the steps before again. Figure 2 displays its flow chart.

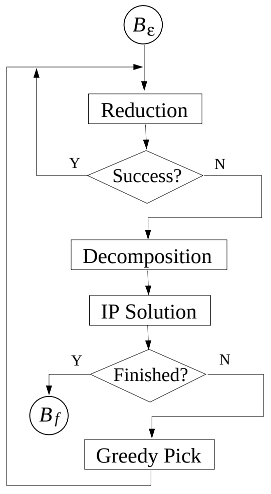


Figure 2: Control flow of the algorithm

There are several ways of checking the overhead in balls for the produced set, B_f. An obvious though quite effective way is to solve the linear program above under the relaxation $x_j \geq 0$. Unfortunately, even when being applied after the first reduction and decomposition, this method becomes very time consuming for larger offsets ε, and it also suffers from numerical problems. We therefore consider two alternative methods where a lower bound is easy to obtain.

Theorem 3 *Let n^* be the size of the optimal solution. If the algorithm takes k greedy picks then*

$$|B_f| \leq n^* + k \; .$$

Proof. After the last greedy pick, b_k, let $\tilde{B}$ be the set of balls still to choose from. Assume that the optimal solution for $\tilde{B}$ uses m balls. Taking no more greedy picks, our algorithm solves $\tilde{B} \cup \{b_k\}$ with $m + 1$ balls, whereas the optimal solution for $\tilde{B} \cup \{b_k\}$ uses at least m balls. The theorem follows by induction. □

To get another bound, consider any set B of balls which covers P. For a ball $b \in B$ put $\alpha(b) = |P \cap b|$. Further, for a point $p \in P$, define its share as

$$share(p) = \frac{1}{\max_{p \in b \in B} \alpha(b)} \; .$$

Let $b(p)$ be a ball that achieves the maximum in the denominator above. Under the (ideal) assumption that B is a disjoint covering of P (and hence covers each point p with $b(p)$ and no other ball), $share(p)$ expresses the amount that p contributes to $|B|$. Thus, for any solution B, we have the lower bound

$$\lceil \sum_{p \in P} share(p) \rceil \leq |B| \; .$$

Let now $\hat{B}$ denote the set of balls being selected before the first greedy pick. Clearly, the size of the optimal solution for $B_\varepsilon \setminus \hat{B}$ differs from that for B_ε by exactly $|\hat{B}|$. If we fix the shares for the points in P with respect to the set $B_\varepsilon \setminus \hat{B}$, then the following holds.

Theorem 4 *The overhead in the set B_f is at most*

$$|B_f| - |\hat{B}| - \left\lceil \sum_{p \in P} share(p) \right\rceil .$$

4 Experimental results

We applied the ball generation method to various data sets, including the two benchmark examples below. In the following tables, the first column displays the allowed offset in percent of the (longest edge of the) bounding box. The second column shows the number of balls produced by a pure greedy algorithm, as opposed to our algorithm, shown in column three. The remaining columns list the three bounds for the overhead, described in Section 3. The LP bound, though mostly dominant, turned out to be too time expensive for the software [15] at entries '•'.

Offset %	Greedy	Hybrid	LP	Thm 3	Thm 4
.00001	4326	4085	3	0	3
.0001	3653	3209	24	62	51
.001	3564	3159	21	55	44
.01	3151	2836	17	42	40
.1	1458	1304	10	46	58
1	168	135	12	31	68
2	73	55	•	8	31
3	45	34	3	8	22

Table 1: Bunny model, $|B_\varepsilon| = 8820$

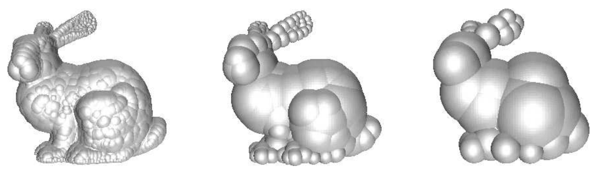


Figure 3: Bunny for offsets .00001%, 1%, and 3%

Offset %	Greedy	Hybrid	LP	Thm 3	Thm 4
.00001	16009	15092	39	105	55
.0001	13235	11699	326	1008	863
.001	12502	11193	317	996	864
.01	8995	8142	226	768	828
.1	2749	2378	221	612	928
1	321	255	•	78	162
2	136	107	•	25	74
3	86	65	•	12	37

Table 2: Dragon model, $|B_\varepsilon| = 34636$

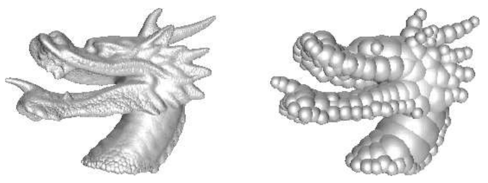


Figure 4: Dragon for offsets .00001% and 1%

References

[1] N. Amenta, M. Bern. *Surface reconstruction by Voronoi filtering.* Discrete & Computational Geometry 22 (1999), 481-504.

[2] N. Amenta, S. Choi, R.K. Kolluri. *The power crust, unions of balls, and the medial axis transform.* Computational Geometry: Theory and Applications 19 (2001), 127-153.

[3] F. Aurenhammer. *Improved algorithms for discs and balls using power diagrams.* J. Algorithms 9 (1988), 151-161.

[4] S. Bischof, L. Kobbelt. *Ellipsoid decomposition of 3D-models.* Proc. 1st IEEE Symp. 3D Data Processing Visualization and Transmission, 2002, 480-488.

[5] G. Bradshaw, C. O'Sullivan. *Adaptive medial-axis approximation for sphere-tree construction.* ACM Transactions on Graphics 23 (2004), 1-26.

[6] B. Chazelle. *Triangulating a nonconvex polytope.* Discrete & Computational Geometry 5 (1990), 505-526.

[7] `http://www.CGAL.org`

[8] E. Fogel, D. Halperin. *Exact and efficient construction of Minkowski sums of convex polyhedra with applications.* 8th Workshop Alg. Eng. Exper., Alenex'06, 2006.

[9] U. Feige. *A threshold of ln n for approximating set cover.* Proc. 28th Ann. ACM Symp. on Theory of Computing, 1996, 314-318.

[10] K. Fukuda. *From the zonotope construction to the Minkowski addition of convex polytopes.* J. Symbolic Computation 38 (2004), 1261-1272.

[11] P.M. Hubbard. *Approximating polyhedra with spheres for time-critical collision detection.* ACM Transactions on Graphics 15 (1996), 179-210.

[12] D.S. Johnson. *Approximation algorithms for combinatorial problems.* J. Computer and System Sciences 9 (1974), 256-278.

[13] R.M. Karp. *Reducibility among combinatorial problems.* R.E. Miller, J.W. Thatcher (eds.), Plenum Press, New York, 1972, 85-103.

[14] M.C. Lin, D. Manocha. *Collision and proximity queries.* J.E. Goodman, J. O'Rourke (eds.), Handbook of Discrete and Computational Goemetry, 2nd Ed., CRC, 2004, 787-807.

[15] `http://lpsolve.sourceforge.net/5.5/`

[16] C. O'Sullivan, J. Dingliana. *Collisions and perception.* ACM Transactions on Graphics 20 (2001), 151-168.

[17] I.J. Palmer, R.L. Grimsdale. *Collision detection for animation using sphere-trees.* Computer Graphics Forum 14 (1995), 105-116.

[18] J. Ruppert, R. Seidel. *On the difficulty of triangulating three-dimensional nonconvex polyhedra.* Discrete & Computational Geometry 7 (1992), 227-253.

Correcting Distortion of Laser-Sintered Parts by Means of a Surface-Based Inverse Deformation Algorithm: An Experimental Study

Christian Pfligersdorffer[*] Martin Held[†]

Abstract

We present an experimental study on correcting shrinkage deformations that occur in laser sintering. The basic idea is to apply an inverse deformation to a polyhedral model prior to its fabrication, and the underlying algorithmic problem is to obtain suitable deformation vectors and to interpolate them over the surface of the model. We use a discretized natural neighbor interpolation adapted to work on triangulated surfaces.

1 Introduction

In *Laser Sintering* (LS), a laser is used to manufacture a part (solid model) by sintering powder-based materials. A thin layer of powder is spread across the build platform (inside a process chamber). Then the laser traces a two-dimensional cross section of the part, thereby sintering the new layer of powder with the previous layer. This process continues layer-by-layer until all parts in a job are completed and is finished by a cooling phase.

Typically, the input data for LS is encoded as a STL file. An STL (Standard Tessellation Language) file describes a raw unstructured triangulated surface of a three dimensional object. Despite of its shortcomings it has become the de-facto industry standard in the entire rapid-prototyping industry; see Burns [1] for more details on the STL format.

Laser Sintering enables the production of durable and functional parts with very much the same properties as their "standard" molded or machined counterparts, for a variety of applications, and even if only a few parts of the same shape are needed. Furthermore, snap fits and living hinges can be produced.

Unfortunately, this heating and cooling down of the material may cause process-inherent problems like inhomogenous temperature fields and the "shrinkage" phenomenon. Roughly speaking, shrinkage is the result of a change in the morphology of the molten powder. During the cooling process the morphology changes from amorphous to part-crystalline. The crystalline regions thereby have a higher density than the amorphous regions, which leads to a loss in volume. Since laser sintering is a layer-wise process, individual layers may undergo a different shrinkage, thus leading to inter-layer tensions between the layers. The tensions may result in a bimetallic effect called "curl" in the rapid prototyping industry. Summarizing, a part may undergo a noticeable distortion, as shown in Fig. 1 for a real-world example. (For ease of visualization the distortion has been exaggerated by a multiplicative factor of ten in this picture.)

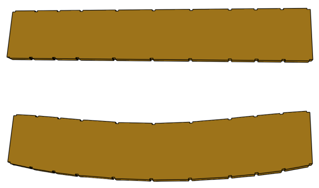


Figure 1: A simple test part produced by laser sintering that ends up with a slight U-shaped deformation.

Of course, process engineers do their best to research the shrinkage of LS parts and to fine-tune the process parameters. However, geometers may be tempted to come up with a different approach to mitigate the effects of shrinkage: Can we apply an *inverse deformation* to a model prior to its LS manufacture such that the part actually manufactured matches the goal shape more closely? (Similar ideas have been applied successfully in textile productions and injection molding.)

2 Problem Studied

Consider a polyhedral part bounded by a triangulated surface which we assume to be a closed 2-manifold. In order to apply an inverse deformation, we (1) need to obtain *deformation vectors* for at least some vertices of the part, and (2) need to interpolate between appropriate deformation vectors in order to obtain deformation vectors for all the other vertices of the part. In the sequel we assume that deformation vectors are already known for some vertices, and focus on the appropriate interpolation of those vectors. (Deformation vectors can be obtained by means of 3D scanners for instance.) We use the term *data sites* for those vertices of the part for which deformation vectors are known.

[*]EOS GmbH, Krailing bei München, Germany, christian.pfligersdorffer@eos.info

[†]Department of Computer Science, University of Salzburg, Salzburg, Austria, held@cosy.sbg.ac.at

Let $P := \{p_1, \ldots, p_n\}$ denote the set of vertices of the part and let $S := \{s_1, \ldots, s_k\}$ denote the sites given, with $S \subset P$. The deformation vector to be applied to s_i is denoted by v_i. Our goal is to construct an *interpolation function* $f : P \to \mathbb{R}^3$ such that $f(s_i) = v_i$ and such that decent deformation vectors for the vertices in $P \setminus S$ are obtained.

To solve this discrete interpolation problem we adapted two well-known multivariate interpolation schemes to make them work on triangulated surfaces: *Inverse Distance Method* and *Natural Neighbor Interpolation*. Both schemes classify as distance-based, statistical schemes [7] and, thus, seem fit for a generalization to arbitrary metric spaces.

3 Inverse Distance Method

Shepard's method of inverse distances [5] dates back to 1968 and as such was one of the first interpolation schemes for scattered, multi-dimensional data.

Definition 1 *For a set S of k sites and a metric d,*

$$f(p) := \sum_{i=1}^{k} v_i w_i(p) \tag{1}$$

gives the Shepard interpolation function for $p \notin S$. The weight functions w_i are parameterized by an exponent $\mu > 0$:

$$w_i(p) := \frac{d(p, s_i)^{-\mu}}{\sum_{j=1}^{k} d(p, s_j)^{-\mu}}. \tag{2}$$

Typical values for μ are 1, 2 and 3. For a thorough discussion of this interpolant in the Euclidean plane we refer to [6]. Although $f \in C^\infty$ for even exponents μ and $f \in C^{\mu-1}$ otherwise, the Shepard interpolation function is known to be of only constant precision. Furthermore the resulting surfaces have gradient zero in the data sites for $\mu > 1$ which causes 'bumps', see Fig. 2, and thus renders this interpolation scheme effectively useless for our purposes.

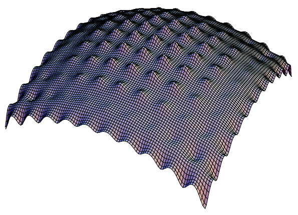


Figure 2: A Shepard surface interpolating 121 data points sampled from a paraboloid (with $\mu = 2$).

A recent modification proposed in [6] achieves higher precision and gets rid of the bumps, unfortunately at the price of incorporating Taylor polynomials. It remains to be seen how their work can be generalized to interpolations on surfaces. In any case, this modification would also require gradient values on the data sites which are not present in our data (and which seem difficult to estimate reliably).

4 Natural Neighbor Interpolation

Natural neighbor interpolation is a multivariate interpolation scheme based on *natural coordinates*, both introduced by Sibson [2, 3]. The basic idea is to express a point p as a linear combination of its neighboring sites, where the weights are derived from the Voronoi diagram of the sites together with p. (Again, a suitable metric d has to be used for the definition of the Voronoi diagram.)

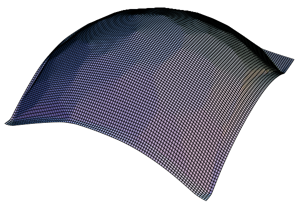


Figure 3: A Sibson surface interpolating 100 randomly spaced data points sampled from a paraboloid.

Definition 2 *The natural coordinates of a point p with respect to a set of sites $S = \{s_1, \ldots, s_k\}$ are given as a vector of weights $(\lambda_1, \ldots, \lambda_k)$, where*

$$\lambda_i(p) := \frac{|V(p, s_i)|}{|V(p)|}. \tag{3}$$

Here, $|V(p)|$ denotes the area of the Voronoi cell of p and $|V(p, s_i)|$ corresponds to the area of the ordered second-order Voronoi cell of p and s_i, i.e. the region that lies closest to p and second-closest to s_i.

The so-called *local coordinates property* establishes the fact that every point p within the convex hull of S can be expressed in natural coordinates $(\lambda_1, \ldots, \lambda_k)$:

$$p = \sum_{i=1}^{k} s_i \lambda_i(p) \tag{4}$$

For a proof we refer to Sibson's original work [2], or to a recent study on Voronoi-based interpolation schemes by Hiyoshi and Sugihara [4].

Definition 3 *For a set S of k sites, with λ_i as defined above,*

$$f(p) := \sum_{i=1}^{k} v_i \lambda_i(p). \tag{5}$$

gives Sibson's C^0-interpolation function for $p \notin S$.

Sibson introduced this basic interpolant that is of linear precision and lies in C^0, and an extended version known as C^1-interpolant that is able to reproduce spherical quadrics. As in the case of Shepard interpolations, this improvement is achieved by incorporating gradients in first-order Taylor polynomials. However, provided that a large enough sample of data sites is available even the linear version is able to produce decent results, see Fig. 3.

Figure 4 illustrates why the method is also called *area stealing method.* The Voronoi cell of p "steals" area from the Voronoi cells of its natural neighbors. The percentage of the Voronoi area around p stolen from a site s_i is given by $\lambda_i(p)$.

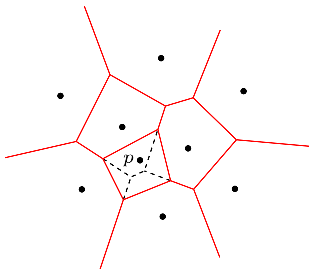


Figure 4: A vertex p is inserted into the Voronoi Diagram of the data sites.

5 Interpolating Deformation Vectors on Surfaces

Since our interpolation is to operate on a triangulated surface the choice of a suitable metric d and the computation of the corresponding Voronoi diagram becomes an imminent problem. No reliable and efficient implementation of the corresponding geodesic Voronoi diagram on a triangulated surface is known even for the Euclidean metric. Thus, we resort to computing discretized approximations of the Voronoi cells. We note, however, that working on closed surfaces also bears the benefit that the restriction to the convex hull of S is no restriction at all.

We apply Dijkstra's shortest path algorithm to compute Euclidean distances between vertices along edges of the triangulated surface. Since the only Voronoi information that we actually need is information on the size of the areas we can get away with not building the Voronoi data structure explicitly. Rather, in a preprocessing phase we store for every vertex a reference to its closest data site by growing the Voronoi regions one by one. For k sites among the n vertices this computation takes $O(kn \log n)$ time in the worst case. Figure 5 shows such an approximate Voronoi diagram.

The main loop iterates over all vertices in order to calculate natural coordinates for them. For every vertex p we grow its Voronoi cell and sum up the area stolen from the data sites. Since we already rely on the mesh triangulation to compute the distances among the vertices we may as well assume the vertex density to be roughly constant over the surface and count stolen vertices instead of evaluating the sizes of stolen areas. This takes $O(n^2 \log n)$ time in the worst-case, and $O(\frac{n^2}{k} \log n)$ if the k sites are uniformly distributed among the n vertices.

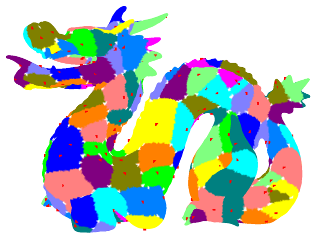


Figure 5: Voronoi diagram on the surface of the Stanford dragon; the data sites are marked as red points.

Observation 1 *One might be tempted to insert vertices permanently into the Voronoi diagram once their natural coordinates are known, in an attempt to let them act as data sites for subsequent iterations of the main loop. Tests showed that this modification sometimes speeds up the algorithm by up to a multiplicative factor of 2–3 (as could be expected) but also downgrades the results obtained! Since Sibson's interpolant is C^0 in the data sites, C^1 on their Delaunay circles, and C^∞ everywhere else, this observation is quite understandable [8].*

Due to the various approximations and the discrete nature of our approach it became necessary to smooth the mesh afterwards. Using a simple *mean value smoothing* this postprocessing step takes only linear time.

6 Discussion of Results Obtained

The CPU-time consumption measured in practical tests of our algorithm reflects the complexity bounds: if interpolations are performed on parts of various complexity, with 100 data sites (and deformation vectors per part), our algorithm clearly runs in time quadratic in the number of vertices, see Fig. 6.

As suggested by the complexity bound $O(\frac{n^2}{k} \log n)$, CPU time can indeed be saved by providing additional input data sites. Fig. 7 shows the decrease in the CPU-time consumption (y-axis) for an STL model with 8404 vertices as the number of data sites distributed randomly among the vertices is increased. Thus, for a practical application of our algorithm it will be important to strike a good balance between the efforts spent on obtaining data sites together with the deformation vectors and the time consumed by the actual inverse deformation.

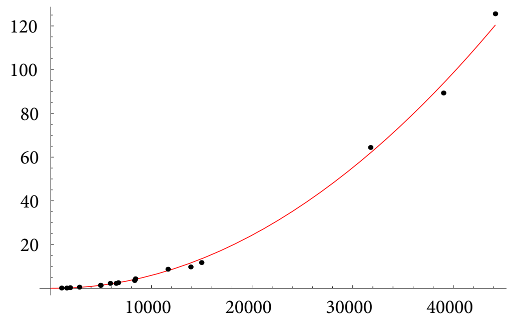


Figure 6: The y-axis shows the CPU time in seconds necessary to interpolate 100 randomly placed sites on triangulated surfaces having x vertices.

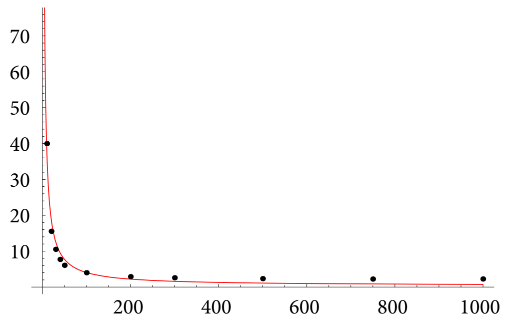


Figure 7: Decrease of CPU-time consumption (y-axis) as the number of data sites (x-axis) is increased (for $n = 8404$).

We note that by applying ideas used for the computation of chessboard distances in 2D we could shave a log-factor from the complexity bounds if we would use the link distance rather than the true Euclidean distance as a metric. However, the speed-up gained in practice is rather small while using the link distance makes the interpolation even more susceptible to irregularly spaced vertices.

Comparisons between parts sintered with and without inverse deformation clearly showed that an inverse deformation pays off. For instance, after applying our algorithm to a model of the bar depicted in Fig. 1, the resulting part matched its template better than the one built without inverse deformation: the deviation from the target geometry was reduced by a factor of 10. (We placed 44 data sites along the bar's main edges, among a total of 6708 vertices.) Similar results were obtained for other long and thin parts.

During the practical evaluation of our algorithm two problems quickly surfaced and turned into major hurdles for a large-scale test:

- How can the deformation vectors be obtained in an automated way?
- How can we re-triangulate surfaces such that we obtain meshes that are suitable for a deformation?

All commercial 3D scanners available to us do a point-to-surface matching but do not support a point-to-point matching. Thus, we need a surface registration mechanism to directly match automatically scanned data with the STL model in order to obtain a suitable set of deformation vectors fully automatically. (The current approach to getting deformation vectors for points on the part's surface that do indeed coincide with vertices of the part involves operations done by hand and is far too time consuming.)

The availability of decent input triangulations also is a critical problem: STL models are generated by engineers using various CAD packages, but few of them are suitable for a (surface-based) deformation. First of all, it is obvious that one cannot hope to deform the bar depicted in Fig. 1 appropriately if its surface consists of a bunch of long rectangles. (Recall that we only shift vertices but do not bend edges!) Similarly, long but skinny triangles or highly irregular sizes of the triangles downgrade the quality of the deformation achieved by our algorithm. Thus, prior to fine-tuning our inverse deformation algorithm (or prior to considering switching to a volume-based interpolation) we will have to include a good re-meshing software into our test set-up.

Acknowledgments

We thank EOS GmbH for sponsoring our work and providing an EOSINT P 385 laser sintering machine for our experiments together with the personnel to handle production and post-production measurement.

References

[1] M. Burns. *Automated Fabrication. Improving Productivity in Manufacturing.* Prentice Hall, 1993.

[2] R. Sibson. *A Vector Identity for the Dirichlet Tesselation.* Mathematical Proceedings of Cambridge Philosophical Society 87, 151–155, 1980.

[3] R. Sibson. *A Brief Description of Natural Neighbor Interpolation.* Interpreting Multivariate Data, John Wiley & Sons, Chichester 1981.

[4] H. Hiyoshi, K. Sugihara. *Voronoi-Based Interpolation with Higher Continuity.* Annual Symposium on Computational Geometry, 242–250, 2000.

[5] D. Shepard. *A Two-dimensional Interpolation Function for Irregularly-Spaced Data.* 23rd ACM National Conference, 1968.

[6] R. Barnhill, R. Dube, F. Little. *Properties of Shepard's Surfaces.* Rocky Mountain Journal of Mathematics 13 (2), 365–382, 1983.

[7] F. Sárközy. *GIS Functions – Interpolation.* Periodica Polytechnica Civil Engineering 43 (1), 63–86, 1999.

[8] G. Farin. *Surfaces over Dirichlet Tesselations.* Computer Aided Geometric Design 7, 281–292, 1990.

Complexity of Approximation by Conic Splines (Extended Abstract)

Sylvain Petitjean* Sunayana Ghosh † Gert Vegter‡

Abstract

In this paper we show that the complexity, i.e., the number of elements, of a parabolic or conic spline approximating a sufficiently smooth curve with non-vanishing curvature to within Hausdorff distance ε is $c_1\varepsilon^{-1/4} + O(1)$, or $c_2\varepsilon^{-1/5} + O(1)$, respectively. The constants c_1 and c_2 are expressed in the Euclidean and affine curvature of the curve. We also prove that the Hausdorff distance between a curve and an optimal conic arc tangent at its endpoints is increasing with its arc-length, provided the affine curvature along the arc is monotone. We use this property in a simple bisection algorithm for computing an optimal parabolic or conic spline.

1 Introduction

Complexity of conic approximants. We show that the complexity—the number of elements—of an optimal *conic spline* approximating a sufficiently smooth curve to within Hausdorff distance (See [6] for a definition) ε, is of the form $c_1\,\varepsilon^{-\frac{1}{5}} + O(1)$, where we express the value of the constant c_1 in terms of the Euclidean and affine curvature (See Corollary 2). An optimal *parabolic spline* approximates a curve to fourth order, so its complexity is of the form $c_2\,\varepsilon^{-\frac{1}{4}} + O(1)$. Also in this case the constant c_2 is expressed in the Euclidean and affine curvature. These bounds are obtained by first deriving an expression for the Hausdorff distance of a conic arc that is tangent to a (sufficiently short) curve at its endpoints, and that minimizes the Hausdorff distance among all such bitangent conics. Applying well-known methods like those of [2] it follows that this Hausdorff distance is of fifth order in the length of the curve, and of fourth order if the conic is a parabola. We derive explicit constants in these asymptotic expansions in terms of the Euclidean and affine curvature of the curve.

Algorithmic issues. For curves with monotone affine curvature, called *affine spirals*, we consider conic arcs tangent to the curve at its endpoints, and show that among such *bitangent conic arcs* there is a unique one minimizing the Hausdorff distance. This optimal bitangent conic arc C_{opt} intersects the curve at its endpoints and at one interior point, but nowhere else. If $\alpha : I \to \mathbb{R}^2$ is an affine spiral, its displacement function $d : I \to \mathbb{R}$ measures the signed distance between the affine spiral and the optimal bitangent conic along the normal lines of the spiral. This displacement function has an *equioscillation property* (See Section 3) and the Hausdorff distance between a section of an affine spiral and its optimal approximating bitangent conic arc is a monotone function of the arc length of the spiral section. This useful property gives rise to a bisection based algorithm for the computation of an optimal interpolating tangent continuous conic spline. The scheme reproduces conics. We implemented such an algorithm, and compare its theoretical complexity with the actual number of elements in an optimal approximating parabolic or conic spline.

Related work. In [4] Fejes Tóth considers the problem of approximating a convex C^2-curve C in the plane by an inscribed n-gon. Fejes Tóth proves that, with regard to the Hausdorff distance, the optimal n-gon P_n satisfies $\delta_H(C, P_n) = \frac{1}{8}\left(\int_0^l \kappa^{1/2}(s)ds\right)^2 \frac{1}{n^2} + O(\frac{1}{n^4})$. Here l is the length of the curve, s its arc length parameter, and $\kappa(s)$ its curvature. Ludwig [8] extends this result by deriving the second term in this asymptotic expansion.

These problems fall in the context of geometric Hermite interpolation, in which approximation problems for curves are treated independent of their specific parametrization. The seminal paper De Boor, Höllig and Sabin [2] also fits in this context. Floater [5] gives a method that, for any conic arc and any odd integer n, yields a geometric Hermite interpolant with $2n$ contacts, counted with multiplicity. This scheme gives a G^{n-1}-spline, and has approximation order $O(h^{2n})$, where h is the length of the conic arc. Degen [3] presents an overview of geometric Hermite interpolation, also emphasizing differential geometry aspects.

Overview. Section 2 reviews some notions from affine differential geometry that we use in this paper. Section 3 introduces affine spirals, a class of curves

*CNRS/Nancy, France; email: petitjea@loria.fr
†University of Groningen, The Netherlands; email: S.Ghosh@cs.rug.nl
‡Corresponding author. University of Groningen, The Netherlands; email: G.Vegter@cs.rug.nl

which have a unique optimal bitangent conic. These optimal bitangent conic arcs have some nice properties giving rise to a bisection algorithm for their computation. The complexity analysis of optimal parabolic and conic splines is presented in Section 4. Section 5 presents the output of the algorithm in a specific example.

2 Mathematical preliminaries

Circular arcs and straight line segments are the only regular smooth curves in the plane with constant Euclidean curvature. Conic arcs are the only smooth curves in the plane with constant *affine curvature*. The latter property is crucial for our approach, so we briefly review some concepts and properties from affine differential geometry of planar curves. See also Blaschke [1].

Affine curvature. Recall that a regular curve $\alpha : J \to \mathbb{R}^2$ defined on a closed real interval J, i.e., a curve with non-vanishing tangent vector $T(u) := \alpha'(u)$, is parametrized according to Euclidean arc length if its tangent vector T has unit length. In this case, the derivative of the tangent vector is in the direction of the unit normal vector $N(u)$, and the Euclidean curvature $\kappa(u)$ measures the rate of change of T, i.e., $T'(u) = \kappa(u)\, N(u)$. Euclidean curvature is a differential invariant of regular curves under the group of rigid motions of the plane, i.e., a regular curve is uniquely determined by its Euclidean curvature, up to a rigid motion.

The larger group of *equi-affine transformations* of the plane, i.e., linear transformations with determinant one (in other words, area preserving linear transformations), also gives rise to a differential invariant, called the *affine curvature* of the curve. To introduce this invariant, let $I \subset \mathbb{R}$ be an interval, and let $\gamma : I \to \mathbb{R}^2$ be a smooth, regular plane curve. The curve γ is parametrized according to *affine arc length* if

$$[\gamma'(r), \gamma''(r)] = 1. \tag{1}$$

Here $[v, w]$ denotes the determinant of the pair of vectors $\{v, w\}$. It follows from (1) that γ has non-zero Euclidean curvature. Conversely, every curve $\alpha : J \subset \mathbb{R} \to \mathbb{R}^2$ with non-zero Euclidean curvature satisfies $[\alpha'(u), \alpha''(u)] \neq 0$, for $u \in J$, so it can be reparametrized according to affine arc length.

Note that the property of being parametrized according to affine arc length is an invariant of the curve under equi-affine transformations. If γ is parametrized according to affine arc length, then differention of (1) yields $[\gamma'(r), \gamma'''(r)] = 0$, so there is a scalar function k such that

$$\gamma'''(r) + k(r)\, \gamma'(r) = 0. \tag{2}$$

The quantity $k(r)$ is called the *affine curvature* of the curve γ at $\gamma(r)$. A regular curve is uniquely determined by its affine curvature, up to an equi-affine transformation of the plane.

The affine curvature can be expressed in terms of the derivatives of γ up to and including order four. We refer to the full version of the paper for details.

At a point of non-vanishing Euclidean curvature there is a unique conic, called the *osculating conic*, having fourth order contact with the curve at that point (or, in other words, having five coinciding points of intersection with the curve). The affine curvature of this conic is equal to the affine curvature of the curve at the point of contact. Moreover, the contact is of order five if the affine curvature has vanishing derivative at the point of contact. (The curve has to be C^5.) In that case the point of contact is a *sextactic point*. See [1] for further details.

Conics have constant affine curvature. Solving the differential equation (2) shows that a curve of constant affine curvature is a conic arc. More precisely, a curve with constant negative affine curvature is a hyperbolic, parabolic, or elliptic arc iff its affine curvature is negative, zero, or positive, respectively.

3 Approximation of affine spirals

Displacement function. A *bitangent conic* of a regular curve $\alpha : I \to \mathbb{R}^2$ is a conic arc which is tangent to α at its endpoints, such that each normal line of α intersects the conic arc in a unique point. Therefore, a bitangent conic has a parametrization $\beta : I \to \mathbb{R}^2$ of the form $\beta(u) = \alpha(u) + d(u)\, N(u)$, where $d : I \to \mathbb{R}$ is the *displacement function* of the conic arc. The Hausdorff distance between α and a bitangent conic C is equal to

$$\delta_H(\alpha, C) = \max_{u \in I} |d(u)|. \tag{3}$$

There is a one-parameter family of bitangent conics, so the goal is to determine an *optimal bitangent conic*, i.e., a conic in this family that minimizes the Hausdorff distance.

Equioscillation property. An *affine spiral* is a regular curve without sextactic points, in other words, a curve with monotone affine curvature. Affine spirals have a *unique optimal bitangent conic*, which is tangent to the curve at its endpoints, and intersects the curve in one additional interior point, but at no other interior point. Moreover, the displacement function of this optimal bitangent conic has an *equioscillation property*: there are exactly two parameter values at which the maximum in (3) is attained. More precisely, there are $u_+, u_- \in I$ such that $d(u_+) = -d(u_-) = \delta_H(\alpha, C_{\text{opt}})$ and $|d(u)| < \delta_H(\alpha, C_{\text{opt}})$ if $u \neq u_\pm$. The points $\alpha(u_-)$ and $\alpha(u_+)$ are separated by the interior point of intersection of α and C_{opt}. The optimal bitangent conic is the unique bitangent conic having this equioscillation property, a property that gives rise to a simple algorithm for computing it. See the full paper for details.

Monotonicity of optimal Hausdorff distance. If one endpoint of the affine spiral moves along the curve α, the Hausdorff distance between the affine spiral and its optimal bitangent conic arc is *monotone* in the arc length of the affine spiral. More precisely, let $\alpha : [u_0, u_1] \to \mathbb{R}^2$ be an affine spiral arc. For $u_0 \le u \le u_1$, let α_u be the sub-arc between $\alpha(u_0)$ and $\alpha(u)$, and let β_u be the (unique) optimal bitangent conic arc of α_u. Then the Hausdorff-distance between α_u and β_u is a monotonically increasing function of u.

This property gives rise to a bisection method for the computation of an optimal conic spline approximating a spiral arc to within a given Hausdorff distance. Section 5 presents the output of this algorithm in a specific example.

4 Complexity of conic splines.

In this section our goal is to determine the Hausdorff distance of a conic arc of best approximation to an arc of α of Euclidean length $\sigma > 0$, that is tangent to α at its endpoints. If the conic is a parabola, these conditions uniquely determine the parabolic arc. If we approximate by a general conic, there is one degree of freedom left, which we use to minimize the Hausdorff distance between the the arc of α and the approximating conic arc β.

The main result of this section gives an asymptotic bound on this Hausdorff distance.

Theorem 1 (Optimal Hausdorff distance) *Let β be a conic arc tangent at its endpoints to an arc of a regular curve α of length σ, with non-vanishing Euclidean curvature.*

1. If α is a C^8-curve, and β is a parabolic arc, then the Hausdorff distance between these arcs has asymptotic expansion

$$\delta_H(\alpha,\beta) = \tfrac{1}{128}\,|k_0|\,\kappa_0^{\frac{5}{3}}\,\sigma^4 + O(\sigma^5), \qquad (4)$$

where κ_0 and k_0 are the Euclidean and affine curvature of α at one of its endpoints, respectively.

2. If α is a C^9-curve, and β is a conic arc, then the Hausdorff distance between these arcs is minimized if the affine curvature of β is equal to the average of the affine curvatures of α at its endpoints, up to quadratic terms in the length of α. In this case the Hausdorff distance has asymptotic expansion

$$\delta_H(\alpha,\beta) = \tfrac{1}{2000\sqrt{5}}\,|k_0'|\,\kappa_0^2\,\sigma^5 + O(\sigma^6), \qquad (5)$$

where κ_0 is the Euclidean curvature of α at one of its endpoints, and k_0' is the derivative of the affine curvature of α at one of its endpoints.

The proof of this result is quite involved, but the main idea is rather simple. Let $\alpha : [0, \varrho] \to \mathbb{R}^2$ be parametrized according to affine arc length. In particular, ϱ is the affine arc length of α. One can show that

$$\varrho = \kappa_0^{\frac{1}{3}}\,\sigma + O(\sigma^2). \qquad (6)$$

The parabolic arc, which is bitangent to α at $\alpha(0)$ and $\alpha(\varrho)$, is an offset curve depending on ϱ. Therefore it has a parametrization $u \mapsto \beta(u, \varrho)$ of the form

$$\beta(u,\varrho) = \alpha(u) + d(u,\varrho)\,N(u), \qquad (7)$$

where the displacement function d is of the form $d(u,\varrho) = u^2(u-\varrho)^2\,D(u,\varrho)$. Then

$$\delta_H(\alpha,\beta) = \max_{0\le u\le\varrho} |d(u,\varrho)| = \tfrac{1}{16}\,\varrho^4|D(0,0)| + O(\varrho^5). \qquad (8)$$

In the full paper we show that the affine curvature of a curve of the form (7) is of the form

$$k_\beta = k_0 + 8\,\kappa_0^{-\frac{1}{3}}\,D(0,0) + O(\varrho). \qquad (9)$$

Since β is a parabolic arc, its affine curvature is zero, i.e., $k_\beta = 0$. Combining (6), (8), and (9) yields the asymptotic expression for the Hausdorff distance between the curve and its bitangent parabolic arc as stated in the first part of the theorem. The proof of the second part is more involved, but follows the same line of reasoning.

The preceding result gives an asymptotic expression for the minimal number of elements of an optimal parabolic or conic spline in terms of the maximal Hausdorff distance.

Corollary 2 (Complexity of conic splines) *Let $\alpha : [0, L] \to \mathbb{R}^2$ be a regular curve of length L, with non-vanishing Euclidean curvature parametrized by Euclidean arc length, and let $\kappa(s)$ and $k(s)$ be its Euclidean and affine curvature at $\alpha(s)$, respectively.*

1. If α is a C^8-curve, then the minimal number of arcs in a tangent continuous **parabolic** *spline approximating α to within Hausdorff distance ε is*

$$N(\varepsilon) = c_1\,\Big(\int_0^L |k(s)|^{\frac{1}{4}}\,\kappa(s)^{\frac{5}{12}} ds\Big)\,\varepsilon^{-\frac{1}{4}}(1 + O(\varepsilon^{\frac{1}{4}})), \qquad (10)$$

where $c_1 = 128^{-\frac{1}{4}} \approx 0.297$.

2. If α is a C^9-curve, then the minimal number of arcs in a tangent continuous **conic** *spline approximating α to within Hausdorff distance ε is*

$$N(\varepsilon) = c_2\,\Big(\int_0^L |k'(s)|^{\frac{1}{5}}\,\kappa(s)^{\frac{2}{5}} ds\Big)\,\varepsilon^{-\frac{1}{5}}(1 + O(\varepsilon^{\frac{1}{5}})), \qquad (11)$$

where $c_2 = (2000\sqrt{5})^{-\frac{1}{5}} \approx 0.186$.

We only sketch the proof, and refer to the papers by McClure and Vitale [9] and Ludwig [8] for details about this proof technique in similar situations. Consider a small arc of α, centered at $\alpha(s)$. Let $\sigma(s)$ be its Euclidean arc length. Then the Hausdorff distance between this curve and a bitangent parabolic arc is $\frac{1}{128}\,|k_0|\,\kappa_0^{\frac{5}{3}}\,\sigma(s)^4 + O(\sigma(s)^5)$, cf Theorem 1. Therefore,

$$\sigma(s) = \sqrt[4]{128}\,|k(s)|^{-\frac{1}{4}}\,\kappa(s)^{-\frac{5}{12}}\,\varepsilon^{\frac{1}{4}}(1+O(\varepsilon^{\frac{1}{4}})).$$

The first part follows from the observation that $N(\varepsilon) = \int_{s=0}^{L} \frac{1}{\sigma(s)}\,ds$. The proof of the second part is similar.

5 Implementation

We implemented an algorithm for the computation of an optimal parabolic or conic spline, based on the monotonicity property. For computing the optimal parabolic spline, the curve is subdivided into affine spirals. Then for a given maximal Hausdorff distance ε, the algorithm iteratively computes optimal parabolic arcs starting at one endpoint. At each step of this iteration the next breakpoint is computed via a standard bisection procedure, starting from the most recently computed breakpoint. The bisection procedure yields a parabolic spline whose Hausdorff distance to the subtended arc is ε. An optimal conic spline is computed similarly. The bisection step is slightly more complicated, since the algorithm has to select the optimal conic arc from a one-parameter family. Here the equioscillation property gives the criterion for deciding whether the computed conic arc is optimal.

A Spiral Curve. We present the results of our algorithm applied to the *spiral curve*, parametrized by $\alpha(t) = (t\cos(t), t\sin(t))$, with $\frac{1}{6}\pi \le t \le 2\pi$.

Table 1 gives the number of arcs computed by the algorithm, and the theoretical bounds on the number of arcs for varying values of ε, both for the parabolic and for the conic spline.

ε	Parabolic		Conic	
	Exp.	Th.	Exp.	Th.
10^{-1}	5	3	3	2
10^{-2}	9	5	4	3
10^{-3}	15	9	6	5
10^{-4}	26	16	9	7
10^{-5}	46	28	13	11
10^{-6}	82	50	21	17
10^{-7}	145	88	32	26
10^{-8}	254	157	50	41

Table 1: The complexity of the parabolic spline and the conic spline approximating the Spiral Curve, comparing the theoretical complexity with the complexity measured in experiments, for various values of the maximal Hausdorff distance ε.

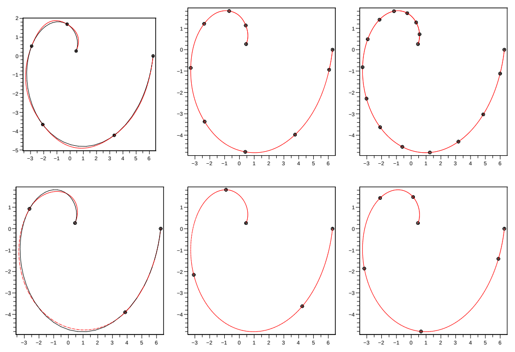


Figure 1: Row 1 shows the parabolic spline and row 2 shows conic spline approximation for the Spiral for $\varepsilon = 10^{-1}, 10^{-2}$ and 10^{-3}

References

[1] W. Blaschke. *Vorlesungen über Differentialgeometrie II. Affine Differential Geometrie*, volume VII of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen.* Springer-Verlag, 1923.

[2] C. de Boor, K. Höllig, and M. Sabin. High accuracy geometric Hermite interpolation. *Computer Aided Geometric Design*, 4:269–278, 1987.

[3] W. Degen. Geometric Hermite interpolation – in memoriam Josef Hoschek. *Computer Aided Geometric Design*, 22:573–592, 2005.

[4] L. Fejes Tóth. Approximations by polygons and polyhedra. *Bull. Amer. Math. Soc.*, 54:431–438, 1948.

[5] M. Floater. An $O(h^{2n})$ Hermite approximation for conic sections. *Computer Aided Geometric Design*, 14:135–151, 1997.

[6] J.E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.

[7] M. Ludwig. Asymptotic approximation of convex curves. *Arch. Math.*, 63:377—384, 1994.

[8] M. Ludwig. Asymptotic approximation of convex curves; the Hausdorff metric case. *Arch. Math.*, 70:331–336, 1998.

[9] D. McClure and R. Vitale. Polygonal approximation of plane convex bodies. *J. Math. Anal. Appl.*, 51:326–358, 1975.

[10] R. Schaback. Planar curve interpolation by piecewise conics of arbitrary type. *Constructive Approximation*, 9:373–389, 1993.

Automatic Local Remeshing of Unstructured Quadrilateral Meshes in Problems with Large Deformations

Alexander V. Skovpen* Vladimir A. Bychenkov Irina I. Kuznetsova

Abstract

Automatic local remeshing is used to maintain mesh quality in 2D problems which are modeled on the moving meshes. The mesh is first smoothed in local regions selected for remeshing. If smoothing does not help improve the mesh, we construct a new one using the advancing front method of unstructured quadrilateral meshing.

1 Introduction

When problems with large deformations are modeled with Lagrangian meshes, quality of cells degrades and may become unacceptable for further calculation. This problem can be resolved by using remeshing/rezoning techniques. Triangles can be remeshed using topological operations such as splitting and elimination of cells and edges and merging of cells. However, for many modeling problems, it is preferable to use quadrilaterals. Direct topological operations with quadrilateral elements are not usually suitable for remeshing (but can be used for adaptive refinement). "Topological" remeshing may be based on splitting quads into triangles; optimization of triangular mesh and further transformation of triangles into quads. But this approach usually generates mesh with nonuniformly oriented cells and poor topological quality (if it is not the advancing front method based on triangle transformations). Another technique widely used for remeshing (especially in 3D) is a grid-based approach proposed by Schneiders [1]. This method is robust but it generates poor quality elements near the boundary and mesh orientation depends on the interior grid only. The grid-based approach is suitable for remeshing of the entire domain but its use for local remeshing tends mesh to be not uniform in topology and element sizes. Local remeshing is more effective because only small region of domain often needs to be remeshed. Remeshing of regions with good elements causes diffusion of state variables in the process of rezoning which leads to unjustified losses in accuracy. It is especially undesirable in numerical analysis of shock waves propagation. In this paper we suggest automatic local remeshing based on the advancing front method of unstructured quadrilateral meshing. Mesh quality is continuously controlled during numerical simulation. If the mesh becomes "bad" in a local sub-domain, it is smoothed. If smoothing does not help, overall remeshing is done in the bad sub-domain. The orientation of new cells depends on the boundaries of the sub-domain, therefore new mesh is partly adapted to the entire mesh of the domain. This approach has successfully been implemented and is used in a number of 2D codes of numerical analysis. An example of problems solved with the use of automatic local remeshing is provided.

2 The meshing algorithm

We use the QMV method [2] for remeshing. The QMV borrows the basic scheme of the Q-Morph method [3], but differs from it mainly in the following:

a) In topologically difficult cases, it uses an additional procedure which forms the new cell from the active front edge. The procedure is based on exhaustive search for all possible variants of cell formation using edges adjacent to active edge nodes as lateral edges of the new cell.

b) Front closing technology is based on splitting of front closing cells according to several patterns.

c) A new topological cleanup procedure was developed. It is based on elementary topological operations and topological operations with cell sets enclosed in six-edge contours.

3 Local remeshing

The local remeshing procedure is called after each computational step. It first checks mesh quality. If it is satisfactory, nothing more is done and the calculation continues.
The local remeshing algorithm consists of the following basic steps:

1. Check the measure of quality M of cells C_i in the computational domain Ω (the measure of quality is described in Section 3.1). If "bad" cells, with quality measure below the limiting value $M_{\lim}$ are found, the algorithm determines the remeshing sub-domain Ω_R. The "bad" cells (subset Ω_r) and

*Russian Federal Nuclear Center – Zababakhin Institute of Technical Physics (RFNC-VNIITF) Snezhinsk, Russia, alex.skovpen@gmail.com, a.v.skovpen@vniitf.ru

a number of neighbor cells which are, as a rule, "almost bad" (subset Ω_r^+) are included in Ω_R. In addition, the sub-domain is extended according to criteria formulated in Section 3.2. Let Ω_r be a subset of cells such that $M(C_i) < M_{\rm lim}$, Ω_r^+ be a subset of cells such that $M(C_i) < M_{\rm lim}^+$. Then $\Omega_r' = \Omega_r \cup \Omega_r^+$ and $\Omega_R = \Psi(\Omega_r')$, where Ψ is a function of the extension of the cell subset.

2. For cells in Ω_R, the mesh is smoothed using a barrier variational method with regularization [4].
3. The repeated check of cell quality in Ω_R and re-determination of Ω_R. If there are no "bad" cells and, accordingly, Ω_R has become empty, we do rezoning for cells, which nodes coordinates changed, and the calculation continues.
4. The C_R – boundary contour of Ω_R is formed and cells of Ω_R are removed from Ω (C_R is a connected set of boundary edges).
5. Adjustment of C_R. Only edges which are boundary in Ω can be modified. Too long edges are splitted and too short ones are eliminated.
6. A new mesh Ω_R' is generated for C_R, using the QMV method.
7. The sub-domain Ω_R' is inserted into Ω, topological connections between new and old cells being recovered.
8. The modification of the mesh in Ω (Section 3.3).
9. The mesh is smoothed for cells of Ω_R' plus 2-3 layers of old cells around them.
10. Rezoning for new cells and for the cells with changed nodal coordinates.

A mesh generated with the QMV method will be fully quadrilateral if the initial boundary consists of an even number of edges. If the number is odd, at least one triangle will exist in the generated mesh. Since the number of edges may become odd after the boundary adjustment, there may be triangles in the domain. If domain boundaries are locally pinched, i.e., one layer of cells can be generated between them, quadrilateral cells with angles close to π may appear there. These cells are unsuitable for modeling and are transformed into triangular cells.

3.1 Cell quality measure

The measure of cell quality M is used to decide whether remeshing is needed. In general, the measure is arbitrary and may differ for different classes of problems. Here M is a set of quality criteria μ_j. To satisfy the measure M a cell must satisfy each of the criteria included in M, i.e., $M(C_i) \geq M_{\rm lim}$ is equivalent to $\mu_j(C_i) \geq \mu_{j,\rm lim}$ for all μ_j $j = 1\ldots m$. Therefore, $M(C_i) < M_{\rm lim}$ if $\mu_j(C_i) < \mu_{j,\rm lim}$ even for one μ_j.
In this paper, 4 quality criteria were used:
- A measure of shape quality μ_1. This measure was proposed in [5] and [6] for, respectively, triangular and quadrilateral cells. Measure for triangle ABC:

$$\mu_1(ABC) = 2\sqrt{3}\left(\frac{\mathbf{AB}\times\mathbf{AC}\cdot\mathbf{n}}{|\mathbf{AB}|^2+|\mathbf{BC}|^2+|\mathbf{CA}|^2}\right),$$

where $\mathbf{n}$ is the unit normal vector of the triangle ABC. The measure for quadrilateral is equal to the minimal measure of triangles which can be built inside the quad splitting it by diagonals.
- A measure μ_2 estimates the angles of the cell. Let the i-th cell angle equals α_i; the ideal angle for the cell is α_I. For a quadrilateral, $\alpha_I = \pi/2$; for a triangle, $\alpha_I = \pi/3$.

$$\mu_2 = \min_{i=1\ldots n} \mu_{2,i}\ ,$$

$$\text{where}\quad \mu_{2,i} = \begin{cases} (\pi-\alpha_i)/(\pi-\alpha_I), & \text{if } \alpha_i \geq \alpha_I \\ \alpha_i/\alpha_I, & \text{if } \alpha_i < \alpha_I \end{cases},$$

n is the number of angles in a cell.
- A cell "stretching" measure μ_3. For a triangular cell, $\mu_3 = l_3/l_1$, where l_i are edge lengths sorted so as $l_1 \geq l_2 \geq l_3$. For a quadrilateral cell, $\mu_3 = (l_3+l_4)/(l_1+l_2)$, where $l_1 \geq l_2 \geq l_3 \geq l_4$.
- Consistency of edge lengths with the specified range:

$$\mu_4 = \min_{i=1\ldots n} [\min(l_i/L_s,\ L_b/l_i)]\ ,$$

where l_i are edge lengths. L_s and L_b are lower and upper limits of edge length.

3.2 The extension of the remeshing sub-domain

The choice of the remeshing sub-domain is an important part of the remeshing algorithm. On one hand, it must not be too large to avoid redundant rezoning. On the other hand, if the sub-domain is too small and narrow, the quality of a new mesh generated within its boundaries won't be good.
The extension of the remeshing sub-domain is done to make it "more convex" and to expand "narrow" regions. The main extension procedure is based on the expansion of "own" narrow regions and on absorbing "alien" narrow regions. Let $O^d(C_i)$ be a set of cells enclosing cell C_i, where d is the number of cell layers around C_i. If $d = 1$, these are the cells which are connected to the cell C_i via a node or edge. The next layers are defined by iterations: $O^{d+1}(C_i) = O^1(O^d(C_i))$. Initially Ω_R is taken to be equal to Ω_r'. Further extension of Ω_R is performed so as to satisfy the following conditions: for each cell C_i of Ω_R, there must exist a cell $C_j \in \Omega_R$ such that $C_j \in O^d(C_i)$, $O^d(C_j) \subset \Omega_R$, $d = D_1$; the number of cells added to Ω_R during extension must be minimal.

Here D_1 characterizes the "attachment" of the cell C_i to Ω_R. This parameter helps to control the extension of Ω_R. Usually D_1 is taken to be $2-4$. For example, if $D_1 = 2$, then the minimal "thickness" of Ω_R will be equal to $2D_1 + 1 = 5$ cells. A similar algorithm is used to absorb the "alien" narrow regions. The cells of Ω which do not belong to Ω_R and have their "attachment" parameter $D_2 = 0$ (for the subset $\Omega_s = \Omega \setminus \Omega_R$), are included in Ω_R.

The remeshing sub-domain can also be extended in order to limit the ratio between the maximum and minimum lengths of its boundary edges and to extend regions along the boundary of domain Ω.

3.3 Mesh modification after local remeshing

If position of boundaries in the domain Ω significantly changes, it may become necessary to topologically disconnect some of its regions or to remove singularities on the boundary. Here "singularities" means thin, wedge-shaped boundary cells. Mesh modification is done using the following operations:

a) Disconnection of inner edges between boundary nodes if $l < L_s$, where l is edge length, L_s is the lower bound of edge length (Figure 1,a).

b) Transformation and elimination of cells whose characteristic size $l_h < L_s$ (Figure 1,b-c). For a triangular cell l_h is the length of minimal height located inside the cell, for quadrilateral $l_h = \min_{i=1 \dots n} [\max(h_1(E_i), h_2(E_i))]$, where $h_1(E_i)$ and $h_2(E_i)$ are heights drawn to edge E_i from nodes opposite this edge. Heights located outside the cell are not taken into account.

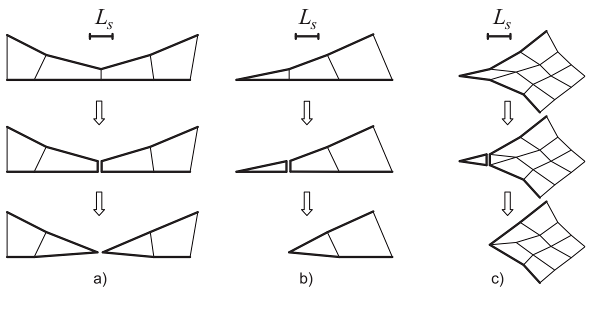


Figure 1: Mesh modification operations.

4 Implementation and example of local remeshing

The proposed method of automatic local remeshing is implemented as a dynamic link library. In case of remeshing, the calling program receives information on all removed, changed and new nodes and cells and also on intersections of new and old cells which are needed for rezoning.

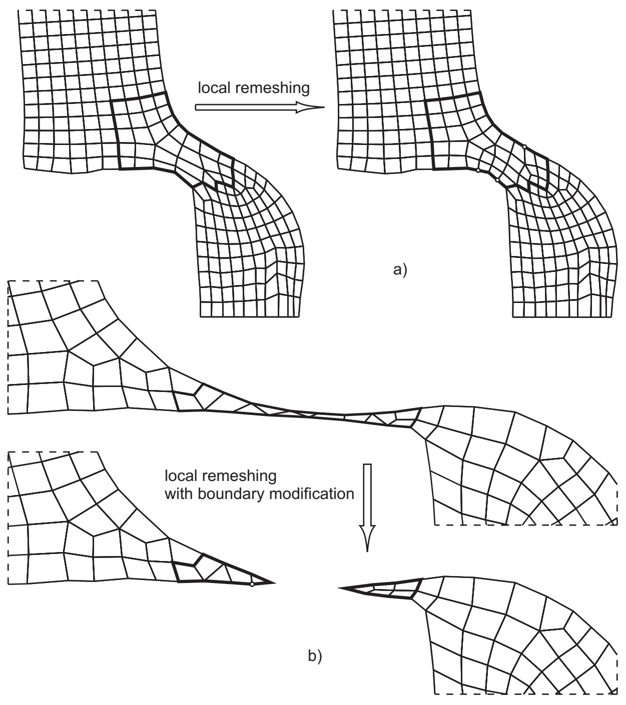


Figure 2: Examples of local remeshing.

Figure 2 shows two moments of remeshing and Figure 3 shows the results of a 2D numerical simulation of perforation of an aluminum plate by a uranium cylinder. The plate is 10mm thick and 200mm in diameter; the cylinder is 20mm in diameter and its velocity is 500 m/s. The numerical simulation was carried out using code SPRUT (RFNC-VNIITF) based on an explicit Lagrangian finite difference scheme with models of elasto-plasticity, material damage, porosity and cracking. Local remeshing was done 18 times, including 2 in which only smoothing was performed. The total number of computational steps was ~ 2000. The calculation was done in an axially symmetric setup. The mesh was mirror-reflected below the axis of symmetry to make Figure 3 more demonstrative. The nodes marked with circles at Figure 2 were added to the boundary of the domain (step 5 of the algorithm). It is seen from Figure 2,b that the modification of the mesh (step 8 of the algorithm) resulted in the topological break of the region. Before the break of Ω_2, there is no mass disbalance; after the break the mass of the plate reduced due to the elimination of several cells. Mass disbalance of Ω_2 was $\vartriangle M/M = 1.21\text{e-}4$. The coefficient of remeshing for Ω_2 was $K_r = 0.07\%$ which is equivalent to remeshing of 7% of cells of a domain at every hundredth step. If $n_c^r(s)$ is the number of cells changed or generated in remeshing at step s, $n_c(s)$ is the number of cells of a domain at step s and S is the total number of computational steps then

$$K_r = \left(\sum_{s=1}^{S} n_c^r(s) \right) \Big/ \left(\sum_{s=1}^{S} n_c(s) \right).$$

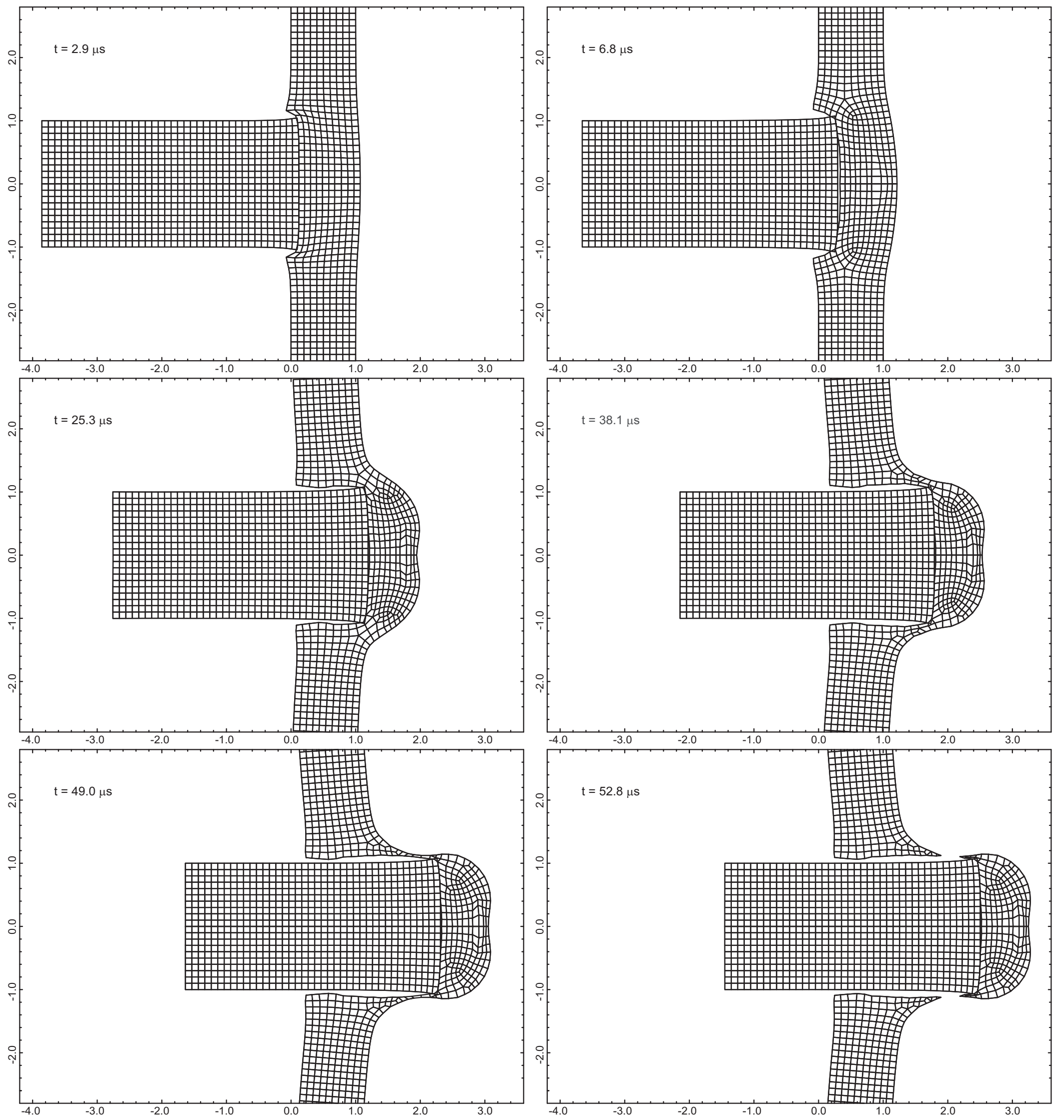


Figure 3: Local remeshing in the perforation problem.

5 Conclusion

Local remeshing helps automatically to maintain geometric quality of computational meshes. This significantly extends the class of problems which can be solved and reduces the time required for the simulation of complex, strongly deformable systems.

Acknowledgments

The authors are thankful to Garanzha V.A., Kozmanov M.Yu., Barabanov S.N., Zyryanova I.L., Pavlov I.V. and Melnikova S.N. for help and support.

References

[1] Schneiders R. *A Grid-based algorithm for the generation of hexahedral element meshes.* Engineering with Computers, 1996, V. 12, p. 168-177.

[2] Skovpen A.V. *Modified algorithm for unstructured quadrilateral meshing.* Snezhinsk, RFNC-VNIITF, ISBN 5-902278-07-4, 2004, 62p. (http://www.vniitf.ru/gridgen/2004_skovpen.pdf).

[3] Owen S.J., Staten M.L., Canann S.A., Sunil S. *Advancing Front Quadrilateral Meshing Using Triangle Transformations.* Proceedings 7th International Meshing Roundtable, 1998, p.409-428.

[4] Garanzha V.A., Kaporin I.E. *Regularization of the barrier variational method of grid generation.* J. Comp. Math. M. Phys. 1999, V. 39, p.1426-1440.

[5] Lo S.H. *A new mesh generation scheme for arbitrary planar domains.* Int. J. Numer. Meth. Engng. 1985, V. 21, p.1403-1426.

[6] Canann S.A., Tristano J.R., Staten M.L. *An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral and tetrahedral meshes.* Proceedings 7th International Meshing Roundtable, 1998, p.479-494.

Edges and Switches, Tunnels and Bridges

David Eppstein[*] Marc van Kreveld[†] Elena Mumford[‡] Bettina Speckmann[‡]

Abstract

Edge casing is a well-known method to improve the readability of drawings of non-planar graphs. A cased drawing orders the edges of each edge crossing and interrupts the lower edge in an appropriate neighborhood of the crossing. Certain orders will lead to a more readable drawing than others. We formulate several optimization criteria that try to capture the concept of a "good" cased drawing. Further, we address the algorithmic question of how to turn a given drawing into an optimal cased drawing. For many of the resulting optimization problems, we either find polynomial time algorithms or NP-hardness results.

1 Introduction

Drawings of non-planar graphs necessarily contain edge crossings. The vertices of a drawing are commonly marked with a disk, but it can still be difficult to detect a vertex within a dense cluster of edge crossings. *Edge casing* is a well-known method—used, for example, in electrical drawings and, more generally, in information visualization—to alleviate this problem and to improve the readability of a drawing. A *cased drawing* orders the edges of each crossing and interrupts the lower edge in an appropriate neighborhood of the crossing. One can also envision that every edge is encased in a strip of the background color and that the casing of the upper edge covers the lower edge at the crossing. See Fig. 1 for an example.

If there are no application specific restrictions that dictate the order of the edges at each crossing, then we can in principle choose freely how to arrange them. Certain orders will lead to a more readable drawing than others. In this paper we formulate several optimization criteria that try to capture the concept of a "good" cased drawing. Further, we address the algorithmic question of how to turn a given drawing into an optimal cased drawing.

Definitions. Let G be a graph with n vertices and m edges and let D be a drawing of G with k crossings.

[*]Department of Computer Science, University of California, Irvine, `eppstein@ics.uci.edu`

[†]Department of Information and Computing Sciences, Utrecht University, `marc@cs.uu.nl`

[‡]Department of Mathematics and Computer Science, TU Eindhoven, `e.mumford@tue.nl` and `speckman@win.tue.nl`

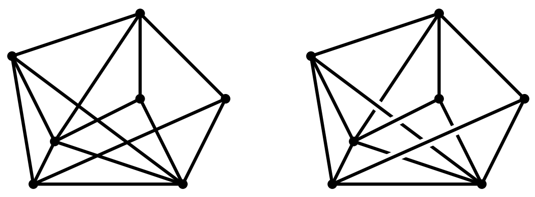


Figure 1: Normal and cased drawing of a graph.

We assume that no vertex v of D lies on (or very close to) an edge e of D unless v is an end-point of e. Further, no more than two edges of D cross in one point and any two crossings are far enough apart so that the casings of the edges involved do not interfere. With these assumptions we can consider crossings independently. We define the *edge crossing graph* G_{DC} for D as follows. G_{DC} contains a vertex for every edge of D and an edge for any two edges of D that cross.

Let C be a crossing between two edges e_1 and e_2. In a cased drawing either e_1 is drawn on top of e_2 or vice versa. If e_1 is drawn on top of e_2 then we say that C is a *bridge* for e_1 and a *tunnel* for e_2. In Fig. 2, C_1 is a bridge for e_1 and a tunnel for e_2. A pair of consecutive crossings C_1 and C_2 along an edge e is called a *switch* if C_1 is a bridge for e and C_2 is a tunnel for e, or vice versa. In Fig. 2, (C_1, C_2) is a switch.

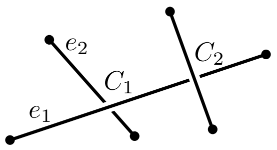


Figure 2: Tunnels and bridges.

Stacking and weaving. When we turn a given drawing into a cased drawing then we need to define a drawing order for every edge crossing. We can choose to either establish a global top-to-bottom order on the edges or to treat each edge crossing individually. We call the first option the *stacking model* and the second one the *weaving model*, since cyclic overlap of three or more edges can occur (see Fig. 3).

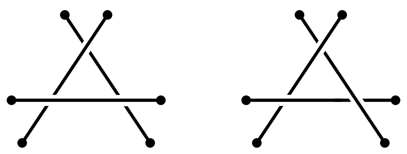


Figure 3: Stacking and weaving.

Quality of a drawing. Globally speaking two factors may influence the readability of a cased drawing in a negative way. Firstly, if there are many switches along an edge then it might become difficult to follow that edge. Drawings that have many switches can appear somewhat chaotic. Secondly, if an edge is frequently below other edges, then it might become hardly visible. These two considerations lead to the following optimization problems for a drawing D.

MinTotalSwitches Minimize the total number of switches.

MinMaxSwitches Minimize the maximum number of switches for any edge.

MinMaxTunnels Minimize the maximum number of tunnels for any edge.

MinMaxTunnelLength Minimize the maximum total length of tunnels for any edge; the length of a tunnel is $\mathit{casingwidth}/\sin\alpha$, where $\alpha \leq \pi/2$ is the angle of the edges at the crossing.

MaxMinTunnelDistance Maximize the minimum distance between any two consecutive tunnels.

Fig. 4 illustrates that the weaving model is stronger than the stacking model for MinTotalSwitches—no cased drawing of this graph in the stacking model can reach the optimum of four switches. For, the thickly drawn bundles of $c > 4$ parallel edges must be cased as shown (or its mirror image) else there would be at least c switches in a bundle, the four vertical and horizontal segments must cross the bundles consistently with the casing of the bundles, and this already leads to the four switches that occur as drawn near the midpoint of each vertical or horizontal segment. Thus, any deviation from the drawing in the casing of the four crossings between vertical and horizontal segments would create additional switches. However, the drawing shown is not a stacked drawing.

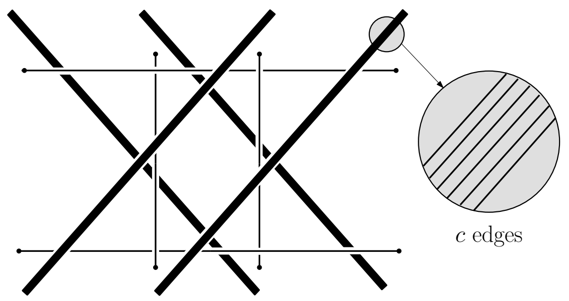


Figure 4: Optimal drawing in the weaving model for MinTotalSwitches.

Results. For many of the problems described above, we either find polynomial time algorithms or NP-hardness results in both the stacking and weaving models. We summarize our results in Table 1.

Model	Stacking	Weaving
MinTotalSwitches	*open*	polyn.
MinMaxSwitches	*open*	*open*
MinMaxTunnels	polyn.	polyn.
MinMaxTunnelLength	polyn.	NP-hard
MaxMinTunnelDistance	polyn.	polyn.

Table 1: Table of results.

2 Minimizing switches

In this section we discuss results related to the MinTotalSwitches and MinMaxSwitches problems. We first discuss some non-algorithmic results giving simple bounds on the number of switches needed, and recognition algorithms for graphs needing no switches. As we know little about these problems for the stacking model, all results stated in this section will be for the weaving model.

Lemma 1 *Given a drawing D of a graph we can turn D into a cased drawing without any switches if and only if the edge crossing graph G_{DC} is bipartite.*

Corollary 2 *Given a drawing D of a graph we can decide in $O((n+m)\log(n+m))$ time if D can be turned into a cased drawing without any switches.*

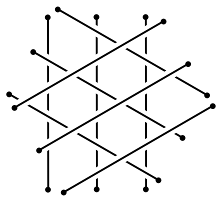
Proof. We apply the bipartiteness algorithm of [2]. Note that this does not construct the arrangement, so there is no factor of k in the runtime. □

Lemma 3 *Given a drawing D of a graph the minimum number of switches of any cased drawing obtained from D is at least half of the number of odd length face cycles in the arrangement of edges (we only count odd-length cycles that do not include a graph vertex).*

Proof. Every odd-length cycle must have a switch. If two odd-length cycles are adjacent, this switch may occur on their shared edge. □

Lemma 4 *For any n large enough, a drawing of a graph G with n vertices and $O(n)$ edges exists, for which any crossing choice gives rise to $\Omega(n^2)$ switches.*

Proof. A construction with three sets of parallel lines, each of linear size, gives $\Omega(n^2)$ triangles, and each triangle gives at least one switch. □



Lemma 5 *For any n large enough, a drawing of a graph G with n vertices and $O(n^2)$ edges exists for which any crossing choice gives rise to $\Omega(n^4)$ switches.*

Proof. We build our graph in the following way. Make a very elongated rectangle, place $n/6$ vertices equally spaced on each short edge, and make the complete bipartite graph. This graph has $(n/6)^2$ edges. One can prove that there is a strip parallel to the short side of the rectangle, such that the parts of the edges inside the strip behave in the same way as parallel ones do with respect to creating triangles when overlapped the way it is described in the previous lemma. This gives us the desired graph with $\Omega(n^4)$ triangles, and hence with $\Omega(n^4)$ switches. □

Theorem 6 MinTotalSwitches *can be solved in polynomial time in the weaving model.*

Proof. Let D be the drawing which we wish to case for the minimum number of switches. We may assume without loss of generality that each vertex of D has degree one, for we may replace any degree-k vertex v by a set of k degree-one vertices placed on a small circle surrounding v, minimize the number of switches in the resulting modified drawing, and then reconnect each edge to v, without changing the number of switches. Define a *segment* of the drawing to be a maximal component of an edge of D that does not include any crossing point with another edge, and define the *parity* of a cycle in D to be the number of segments along the cycle, plus the number of vertices of D contained within the cycle, modulo two.

We apply a solution technique related to the Chinese Postman problem, and also to the problem of via minimization in VLSI design [1]: form an auxiliary graph G, and include in G a single vertex for each odd-parity face cycle in D. Also include in G an edge connecting each pair of vertices, and label this edge by the number of segments of the drawing that are crossed in a path connecting the corresponding two faces in D that crosses as few segments as possible. We claim that the minimum weight of a perfect matching in G equals the minimum total number of switches in any casing of D.

In one direction, we can case D with a number of switches equal to or better than the weight of the matching, as follows: for each edge of the matching, insert a small break into each of the segments in the path corresponding to the edge. The resulting broken arrangement can be shown to have no odd face cycles, for the breaks connect pairs of odd face cycles in D to form larger even cycles. More strongly, it has no odd cycles at all, for in any graph drawing in which all vertices have degree one, the parity of any cycle is the sum (modulo two) of the face cycles within it. Therefore, it has a bipartite edge crossing graph, and can be drawn without switches. Forming a drawing of D by reconnecting all the break points adds at most one switch per break point, so the total number of switches equals at most the weight of the perfect matching.

In the other direction, from any casing of D we can derive a set of paths connecting odd cycles via switch points, showing that the number of switch points is at least as large as the weight of the optimal perfect matching; we omit the details. □

3 Minimizing tunnels

In this section we present three algorithms that solve MinMaxTunnels, MinMaxTunnelLength, and MaxMinTunnelDistance in the stacking model. We also present algorithms for MinMaxTunnels and MaxMinTunnelDistance in the weaving model. MinMaxTunnelLength is NP-hard in the weaving model.

3.1 Stacking model

In the stacking model, some edge e has to be bottommost. This immediately gives the number of tunnels of e, the total length of tunnels of e, and the shortest distance between two tunnels of e. The idea of the algorithm is to determine for each edge what its value would be if it were bottommost, and then choose the edge that is best for the optimization to be bottommost (smallest value for MinMaxTunnels and MinMaxTunnelLength, and largest value for MaxMinTunnelDistance). The other $m-1$ edges are stacked iteratively above this edge. It is easy to see that such an approach indeed maximizes the minimum, or minimizes the maximum. We next give an efficient implementation of the approach. The idea is to maintain the values of all not yet selected edges under consecutive selections of bottommost edges instead of recomputing it.

We start by computing the arrangement of edges in $O(m \log m + k)$ expected time, for instance using Mulmuley's algorithm [4]. This allows us to determine the value for all edges in $O(k)$ additional time.

For MinMaxTunnels and MinMaxTunnelLength, we keep all edges in a Fibonnacci heap on this value. One selection involves an extract-min, giving an edge e, and traversing e in the arrangement to find all edges it crosses. For these edges we update the value and perform a decrease-key operation on the Fibonnacci heap. For MinMaxTunnels we decrease the value by one and for MinMaxTunnelLength we decrease by the length of the crossing, which is *casingwidth*$/\sin\alpha$, where α is the angle the crossing edges make. For MinMaxTunnels and MinMaxTunnelLength this is all that we need. We perform m extract-min and k decrease-key operations. The total traversal time along the edges throughout the whole algorithm is $O(k)$. Thus, the algorithm runs in $O(m \log m + k)$ expected time.

For MaxMinTunnelDistance we use a Fibonnacci heap that allows extract-max and increase-

KEY. For the selected edge we again traverse the arrangement to update the values of the crossing edges. However, we cannot update the value of an edge in constant time for this optimization. We maintain a data structure for each edge that maintains the minimum tunnel distance in $O(\log m)$ time under updates. The structure is an augmented balanced binary search tree that stores the edge parts in between consecutive crossings in its leaves. Each leaf stores the distance between these crossings. Each internal node is augmented such that it stores the minimum distance for the subtree in a variable. The root stores the minimum distance of the edge if it were the bottommost one of the remaining edges. An update involves merging two adjacent leaves of the tree and computing the distance between two crossings. Augmentation allows us to have the new minimum in the root of the tree in $O(\log m)$ time per update. The whole algorithm therefore takes $O(m \log m + k \log m)$ expected time.

Theorem 7 *Given a straight-line drawing of a graph with n vertices, $m = \Omega(n)$ edges, and k edge crossings, we can solve* MinMaxTunnels *and* MinMaxTunnelLength *in $O(m \log m + k)$ expected time and* MaxMinTunnelDistance *in $O(m \log m + k \log m)$ expected time in the stacking model.*

3.2 Weaving model

In the weaving model, the polynomial time algorithm for MinMaxTunnels comes from the fact that the problem of directing an undirected graph, and minimizing the maximum indegree, can be solved in time quadratic in the number of edges [5]. We apply this on the edge crossing graph of the drawing, and hence we get $O(m^4)$ time. For minimizing tunnel length per edge, we can show:

Theorem 8 MinMaxTunnelLength *is NP-hard in the weaving model.*

In the remainder of this section we show how to solve MaxMinTunnelDistance. We observe that there are polynomially many possible values for the smallest tunnel distance, and perform a binary search on these, using 2-SAT instances as the decision tool.

Our algorithm first computes the arrangement of the m edges to determine all crossings. Only distances between two—not necessarily consecutive—crossings along any edge can give the minimum tunnel distance. One edge crosses at most $m-1$ other edges, and hence the number of candidate distances, K, is $O(m^3)$. Obviously, K is also $O(k^2)$. From the arrangement of edges we can determine all of these distances in $O(m \log m + K)$ time. We sort them in $O(K \log K)$ time to set up a binary search. We will show that the decision step takes $O(m + K)$ time, and hence

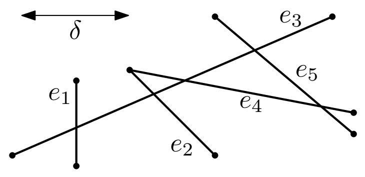


Figure 5: Example where the 2-SAT formula is $(\overline{x}_{13} \vee \overline{x}_{23}) \wedge (\overline{x}_{23} \vee x_{34}) \wedge (\overline{x}_{23} \vee x_{35}) \wedge (x_{34} \vee x_{35})$.

the whole algorithm takes $O(m \log m + K \log K) = O((m+K) \log m)$ time.

Let δ be a value and we wish to decide if we can set the crossings of edges such that all distances between two tunnels along any edge is at least δ. For every two edges e_i and e_j that cross and $i < j$, we have a Boolean variable x_{ij}. We associate x_{ij} with TRUE if e_i has a bridge at its crossing with e_j, and with FALSE otherwise. Now we traverse the arrangement of edges and construct a 2-SAT formula. Let e_i, e_j, and e_h be three edges such that the latter two cross e_i. If the distance between the crossings is less than δ, then e_i should not have the crossings with e_j and e_h as tunnels. Hence, we make a clause for the 2-SAT formula as follows (Fig. 5): if $i < j$ and $i < h$, then the clause is $(x_{ij} \vee x_{ih})$; the other three cases ($i > j$ and/or $i > h$) are similar. The conjunction of all clauses gives a 2-SAT formula that is satisfiable if and only if we can set the crossings such that the minimum tunnel distance is at least δ. We can construct the whole 2-SAT instance in $O(m+K)$ time since we have the arrangement, and satisfiability of 2-SAT can be determined in linear time [3].

Theorem 9 *Given a straight-line drawing of a graph with n vertices and $m = \Omega(n)$ edges, we can solve* MaxMinTunnelDistance *in $O((m+K) \log m)$ expected time in the weaving model, where $K = O(m^3)$ is the total number of pairs of crossings on the same edge.*

References

[1] R.-W. Chen, Y. Kajitani, and S.-P. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Trans. Circuits and Systems*, 30(5):284–299, 1983.

[2] D. Eppstein. Testing bipartiteness of geometric intersection graphs. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 853–861, 2004.

[3] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[4] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms.* Prentice Hall, 1994.

[5] V. Venkateswaran. Minimizing maximum indegree. *Discrete Applied Mathematics*, 143:374–378, 2004.

Rendering the Flow of Comparabilities in Ordered Sets

Guy-Vincent Jourdan[*] Livaniaina Rakotomalala Nejib Zaguia[†]

Abstract

In this paper, we introduce a new concept to draw an ordered set: the *LR-Upward* drawing is an upward drawing based on a chain decomposition of the order such that elements drawn on the same vertical line are always comparable and all other comparabilities flow from left to right. We describe a particular technique for automatically generating enhanced *LR-Upward* drawing for *N-Free* orders that are *X-Cycle-Free.* This technique first enhances locally the drawing, around a particular chain, and then expands the enhancement on the remaining part of the order. This technique can be used to enhance the usability of some complex pictures.

1 Introduction

When a complex and potentially confusing information should be presented to a user, one option is to use a graph showing all the relevant elements and their relationships in a visual way. In some cases, the relationships between elements is a hierarchy (i.e. an antisymmetric and transitive relation), in which case the graph is an ordered set. Several methods for the automatic, computer generated drawing of ordered sets are available in the literature (see [1] for a survey on the question), but none of them is fully satisfying. The most common drawing technique used to represent ordered sets is the upward drawing (or Hasse diagram). An upward drawing suppresses all nonessential edges (those implied by transitivity and the loops at each vertex due to reflexivity) and draws only the directed covering graph of the order in such a way that covering relations are all directed upward. In other words, an upward drawing contains no horizontal edges, and no nonessential edges, and if an element x is smaller than y then there exists a path from x to y that is directed upwards.

Short of having a general definition of what a "good" upward drawing of a given order is, several standard criteria have been identified and analyzed in isolation or combined. These criteria include planarity [2], the slope of the edges, the number of directions, the number of edge crossing [3] etc. Among the qualities that are expected from a good drawing, one obviously important one is how easy it is to see if two elements are comparable in the order.

In this paper we propose a modified version of the upward drawing: the *LR-Upward* drawing concept. It is a new way to visualize ordered sets. Our approach is to use a chain decomposition of the ordered set as the underlying structure for positioning the vertices of the order. We focus particularly on *N-Free* and *X-Cycle-Free* orders. For that class, we detail a complete technique to generate and improve automatically an *LR-Upward* drawing. A Java implementation of the algorithms described in this paper is available. Due to the space constraint and a part from some simple lemmas, none of the technical results that are behind the algorithms will be presented here.

2 Definitions

An ordered set $P = (X, \leq)$ is a pair consisting of a non-empty set X and a binary relation $\leq$ on X, satisfying *reflexivity*, *antisymmetry* and *transitivity.* Two elements x and y are *comparable* in P if either $x \leq y$ (x is a *predecessor* of y) or $y \leq x$ (x is a *successor* of y); otherwise, x and y are *incomparable*, which we note $x \| y$. The covering relation of an ordered set P is the transitive reflexive reduction of P. An element y of P *covers* another element x, and we note $x \prec y$ (x is a *immediate predecessor* of y), if $\forall z, x \leq z \leq y$ and $x \neq z$ implies $z = y$.

A *chain* in P is a set of pairwise comparable elements. An *antichain* in P is a set of pairwise incomparable elements. The *width* of P, $width(P)$, is the size of its longest antichain. A *chain decomposition* of P is a partition of P into chains. Every order P can be partitioned into $width(P)$ chains, and this is the smallest possible partition . A *linear extension* of P is a total ordering $x_1, x_2, \cdots, x_n$ of P such that if $x_i \leq x_j$ in P then $i \leq j$. A linear extension $L = \{x_1 < x_2 < \cdots < x_n\}$ of P is *greedy* if it is constructed inductively such that the $i + 1^{th}$ element is chosen minimal in $P \setminus \{x_1, \cdots, x_i\}$ so that it is comparable to x_i whenever possible. Intuitively, a greedy linear extension is built by always "climbing as high as you can" along the chains.

[*]School of Information Technology and Engineering (SITE), University of Ottawa,Canada, gvj@site.uottawa.ca

[†]School of Information Technology and Engineering (SITE), University of Ottawa,Canada, zaguia@site.uottawa.ca

3 *LR-Upward* drawing

The fundamental idea of *LR-Upward* drawing of an ordered set P is to start from a chain decomposition of P and draw each chain along a different vertical line. The goal is to have comparability information more implicit: if two elements are on the same vertical line, then they are comparable. This very simple idea provides a good visual rendering for some classes of ordered sets, *e.g.* the ones than can be decomposed into a small number of chains. However, this idea alone does not help when it comes to figuring out the comparability of two elements that are not on the same chain in the initial chain decomposition.

In order to improve the readability of the comparability of elements that are not on the same chain, we propose to use a chain decomposition derived from a linear extension of the order. Thanks to that choice, we are able to ensure that in addition to the usual "bottom to top" direction, all covering relations go from left to right (hence the name of *LR-Upward*, for "left to right, upward"). This technique introduces a sense of "flow" in the figure, which helps with the reading of the comparabilities. As one can see from Figure 1, and for this example, the flow of information is much better rendered with the *LR-Upward* drawing.

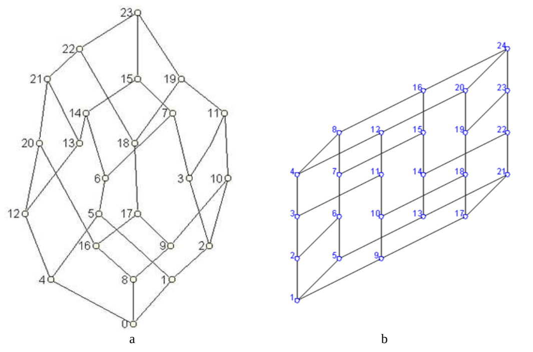


Figure 1: A 24 vertex order drawing generated with LatDraw (a) and with our *LR-Upward* drawing system (b).

The basic algorithm used to obtain an *LR-Upward* drawing of an ordered set P can be informally sketched as follows ($C_i[k]$ stands for the k^{th} element of the chain C_i, $|C_i|$ stands for the number of elements in the chain C_i, and x_x and x_y are the x and y coordinate of the element x in the figure):

Because L is a linear extension of P, it is clear that when x is selected at line 4, all of its predecessors have been already selected, thus the line 12 can be achieved and the resulting edges go either vertically on chain C_i or from left to right for the predecessors that are not in C_i.

4 *N-Free* and *X-Cycle-Free* ordered sets

We now investigate the *LR-Upward* drawing using the approach described above on *N-Free* and *X-Cycle-Free* ordered sets. *N-Free* orders define an important class in the theory of ordered sets that has been very well investigated due to its relations with many applications [5].

The N ordered set is an ordered set of four distinct elements a, b, c and d such that $a \prec c$, $b \prec d$, $b \prec c$ and $a \| d$. An ordered set is *N-Free* if its diagram contains no sub-diagram isomorphic to N.

The X-cycle ordered set is an ordered set of four distinct elements a, b, c and d such that $a \prec c$, $b \prec d$, $b \prec c$ and $a \prec d$. An ordered set is *X-Cycle-Free* if its diagram contains no sub-diagram isomorphic to the X-cycle.

Every finite ordered set can be embedded into an *N-Free* and/or *X-Cycle-Free* ordered set in a linear time for an extra $O(n)$ space in the worst-case (e.g. by adding an element on each covering relation). Thus, an upward drawing solution for this family of orders is also an upward drawing solution for general orders (although it is not necessarily a straight edge upward drawing).

The following are preliminary and important facts: we prove that *N-Free* ordered sets can be drawn using the *LR-Upward* Drawing in such a way that the only relationships between elements that are not on the same chain are between the bottom of a chain and a single smaller element, and between the top of the chain and a single larger element. In addition we show, that if the order is also *X-Cycle-Free*, a given chain can only have one such a relation from its bottom element to another (smaller) chain, and one such relation from its top element to another (larger) chain.

4.1 The "chains interchange" technique

The chain interchanging is a systematic technique that allows us to manipulate the chains of a greedy linear extension without affecting its basic property of being greedy. It only works for *N-Free* ordered sets and does not affect the number of chains in the decomposition. This operation has been introduced by Rival [4] to successfully prove that any greedy linear extension is optimal for the jump number problem. Moreover, every optimal linear extension for the jump number problem is actually a greedy linear extension [6].

This technique will allow us to "navigate" among the greedy linear extensions in order to find the "appropriate" one that could be used as the underlying structure for our upward drawing.

Let P be an *N-Free* ordered set and let L be a greedy linear extension of P where $L = C_1 \oplus C_2 \oplus C_3 \cdots \oplus C_n$. For every index i such that $0 \leq i < n$, there are at most two covering relations between the chains C_i and C_{i+1}, that is, $Top(C_i) \prec u$ for some $u \in C_{i+1} \setminus Bottom(C_i{+}1)$ and $v \prec Bottom(C_{i+1})$ for some $v \in C_i \setminus Top(C_i)$.

Let $C'_i = \{x \in C_i : x \leq v\} \cup \{x \in C_{i+1} : x < u\}$ and $C'_{i+1} = \{x \in C_i : x > v\} \cup \{x \in C_{i+1} : x \geq u\}$.

We transform the greedy linear extension $L = C_1 \oplus C_2 \oplus \cdots \oplus C_i \oplus C_{i+1} \cdots \oplus C_n$ into another greedy linear extension $L = C_1 \oplus C_2 \oplus \cdots \oplus C'_i \oplus C'_{i+1} \cdots \oplus C_n$. We denote this operation by $IC(L, i) = L'$ (see Figure 2).

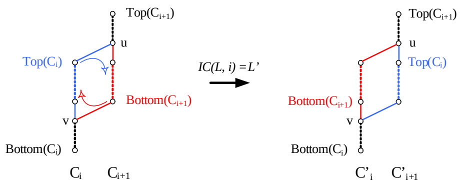


Figure 2: The chains interchange between two consecutive chains.

5 *LR-Upward* drawing of *N-Free* and *X-Cycle-Free* ordered sets

5.1 Local drawing improvement

Contrary to many conventional approaches, our aim is not to directly improve globally the upward drawing. Instead, we first approach the problem with a local improvement of certain parts of the drawing and then we recursively try to expand this local enhancement to the remaining parts of the diagram.

Our goal is to enhance the local display around a chain C_i: reduce the number of edges crossing and give a sense of a flow from the bottom left to the top right. To achieve this, we rearrange the chains on both sides of the chain C_i so that these chains are ordered according to the position in which they are connected to C_i. In other words, we want to reach a chain decomposition $C_1 \oplus C_2 \oplus \cdots \oplus C_i \oplus \cdots \oplus C_n$ such that $\forall j, k \leq n$, if $\exists x, y \in C_i$ such that $x < y$, $Top(C_j) \prec x$ and $Top(C_k) \prec y$, then $j > k$. Conversely, if $\exists x, y \in C_i$ such that $x < y$, $x \prec Bottom(C_j)$ and $y \prec Bottom(C_k)$, then $j > k$.

The left-neighbors chains have to be ordered as displayed in the Figure 3. To accomplish this, we "pull" successively each left neighbor until it reaches its "targeted" place. We start the process by pulling the neighbor chain which is linked by the smaller element in C_i. The general formula for the position is as follows: if C_k is linked by the d^{th} element of C_i (among these elements that are linked to a chain to the left), then the targeted place of C_k is the position $i - d$. However, the place exchange between C_k and the chain currently located at position $i - d$ cannot always be done immediately, since our chain exchange algorithm requires the chains to be consecutive in the linear extension. Consequently, we will need $i - d - k$ chain exchanges to achieve our goal.

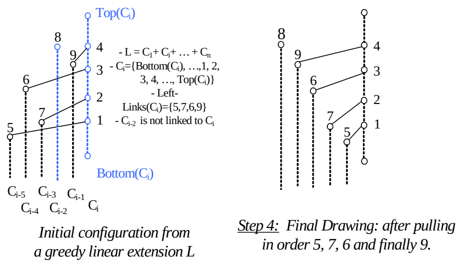

Top(C_i)
8
9
4
3
2
1
6
7
5
- L = C_1+ C_i+ … + C_n
- C_i={Bottom(C_i), …,1, 2, 3, 4, …, Top(C_i)}
- Left-Links(C_i)={5,7,6,9}
- C_{i-2} is not linked to C_i
Bottom(C_i)
C_{i-5}
C_{i-3}
C_{i-1}
C_{i-4}
C_{i-2}
C_i
Initial configuration from a greedy linear extension L
8
9
4
3
6
2
7
1
5
Step 4: Final Drawing: after pulling in order 5, 7, 6 and finally 9.

Figure 3: The Left neighbors chains restructuring.

Formally, let $L = C_1 \oplus C_2 \oplus \cdots \oplus C_i \oplus \cdots \oplus C_n$ be a greedy linear extension of P, and let $i < n$. We define the set *Left-Links*(L, C_i) as the set of elements $x \in P$, such that $x = Top(C_k)$ for some $k < i$ and x is covered by some element in C_i. The reorganization of the left-neighborhood of a chain C_i in a greedy linear extension can be established as follows:

Lemma 1 *Let P be a finite ordered set which is N-Free and X-Cycle-Free. Let $L = C_1 \oplus C_2 \oplus \cdots \oplus C_{i-1} \oplus C_i \oplus \cdots \oplus C_n$ be a greedy linear extension of P, and let $i \leq n$. There exists another greedy linear extension $L' = C'_1 \oplus C'_2 \oplus \cdots \oplus C'_{i-1} \oplus C_i \oplus \cdots \oplus C_n$ of P such that:*

1. *Left-Links(L, C_i) = Left-Links(L', C_i)*

2. *If $Top(C'_k) < u$ and $Top(C'_j) < v$ for some elements $u, v \in C_i$ and $u < v$, then $j < k$*

3. *All chains having their top element in Left-Links(L', C_i) are consecutive: let j such that $Top(C'_j) \in$ Left-Links(L', Ci), $\forall k$ such that $j \leq k < i$, $Top(C'_k) \in$ Left-Links(L', C_i).*

Clearly, the same idea can be applied to the "right" side of the chain C_i.

5.2 The global picture enhancement algorithm

One of the main issues with generalizing the local improvement presented above is that "improving" one chain may deteriorate the improvement already done on another chain. However, restructuring the "left" neighbors of a chain C_j will never conflict with the left-neighborhood of a chain C_i already arranged as long as C_j is on the left of C_i (i.e. $j < i$). Conversely, restructuring the "right" neighbors of a chain C_k will never conflict with the right-neighborhood of a chain C_i already arranged as long as C_k is on the right of C_i (i.e. $i < k$).

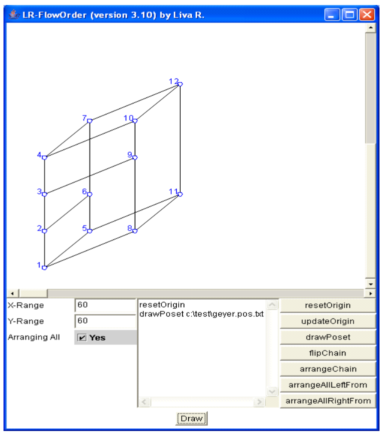
5.2.1 The "heaviest" chain approach

The technique we suggest to improve globally the drawing is to automatically select a chain that "splits" our drawing in two parts. Such a chain must have the most "connections" in terms of left-neighbors and right-neighbors. We choose that particular chain because it is the one that will most benefit from a "complete" local enhancement, and that will have the most chances of producing edge crossing.

Once such a starting chain C_i is selected, we then restructure left-neighborhood "decreasingly" from the index i to the first chain and the right-neighborhood "increasingly" from the index i to the last chain. The index k of the heaviest chain can be computed beforehand during the initial drawing of P.

6 Conclusion

In this paper, we introduce a new technique for drawing ordered sets: the *LR-Upward* drawing is an upward drawing that has its edges flowing from the bottom left to the top right. The technique can be used on any ordered set, but we give a specific algorithm that is effective at producing an improved *LR-Upward* drawing for the ordered sets that are *N-Free* and *X-Cycle-Free*. A full Java implementation of the techniques introduced in this paper and several examples are available from `http://www.site.uottawa.ca/~zaguia`. The example of Figure 4 was produced with that software, while the same order is shown in Figure 5 rendered with two other tools, LatDraw which is available from `http://www.math.hawaii.edu/~ralph/LatDraw` and GraphWin which is available from `http://www.algorithmic-solutions.info/leda\_guide/graphwin.html`.



```
X-Range        60
Y-Range        60
Arranging All  ☑ Yes

resetOrigin
drawPoset c:\test\geyer.pos.txt
```

Figure 4: A 12 vertex order as rendered by our tool.

In future work, we intend to investigate in further details the properties of such a diagram. In particular, one intriguing direction is the possibility to omit in

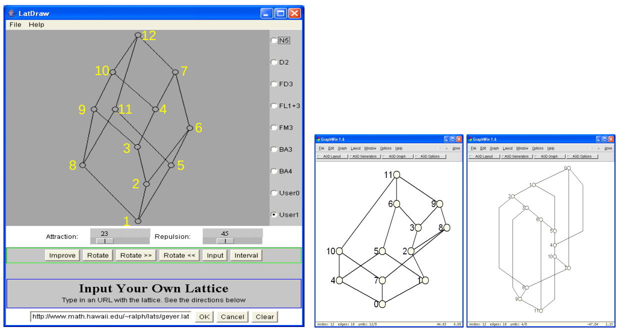


Figure 5: The order of Figure 4 rendered with LatDraw and with two versions of GraphWin.

the figure all the vertical edges, since comparabilities on vertical lines are implied. Under that new context, what are the orders that are planar? Can it be decided in polynomial time?

References

[1] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[2] A. Garg and R. Tamassia. Upward planarity testing. *Order*, 12:109–133, 1995.

[3] C. Lin. The crossing number of posets. *Order*, 11:1–25, 1994.

[4] I. Rival. Optimal linear extensions by interchanging chains. *Proc. American Math. Society*, (89):387–394, 1983.

[5] I. Rival. Stories about order and the letter n (en). *Contemporary Mathematics*, (57), 1986.

[6] I. Rival and N. Zaguia. Constructing greedy linear extensions by interchanging chains. *Order*, (3):107–121, 1986.

An Algorithm for 3D-biplanar Graph Drawing

Meysam Tavassoli* Mohammad Ghodsi† Farnoosh Khodakarami ‡ S. Mehdi Hashemi §

Abstract

We introduce the concept of 3D-biplanar drawing in which we partition a graph into two planar induced subgraphs. Our goal is to find such a partition with the minimum number of edges between the two partitions. We prove that this problem is NP-complete and present a randomized parameterized algorithm with $O(n^k)$ time, where k is the ratio of the optimal solution to the min-cut size of the graph.

1 Introduction

Layered graph drawing [1, 14] is a popular paradigm for drawing graphs which has applications in visualization [15], in DNA mapping, and in VLSI layout [9]. In a layered drawing of a graph, vertices are arranged in horizontal layers, and edges are routed as polygonal lines between distinct layers. For acyclic digraphs, it may be required that edges point downward. Figure 1 shows a sample graph with its 3-Layer drawing.

The quality of layered drawings is assessed in terms of criteria to be minimized, such as the number of edge crossings; the number of edges when removed eliminates all crossings; the number of layers; the maximum span of an edge, i.e., the number of layers it crosses; the total span of the edges; and the maximum number of vertices in one layer.

Research on layered graph drawing has been mainly focused on drawing a graph which admits a 2-layer drawing with no edge crossings. There are some well known problems in this area:

BIPLANAR DRAWING: Given a bipartite graph $G = (A, B; E)$, G is said to be biplanar if the vertices can be drawn on two layers so that none of the edges of G cross. Eades and Whitesides proved that determining whether a given G has a biplanar subgraph with at least K edges is NP-complete. This remains true when the positions of the vertices on one layer are specified [5].

PLANARIZATION: 2-LAYER PLANARIZATION problem, in which given a graph G (not necessarily bipartite), and an integer k called its parameter, the question is whether G can be made biplanar by deleting at most k edges. If a permutation π of A is given, this problem is called 1-LAYER PLANARIZATION.

CROSSING MINIMIZATION: Instead of deleting edges, one can seek to minimize the number of crossings in a 2-layer drawing (here the input graph must be bipartite). The corresponding problems are called 1- and 2-LAYER CROSSING MINIMIZATION.

Unfortunately, the question of whether a graph G can be drawn in two layers with at most k crossings (Crossing Minimization), where k is a part of the input, is NP-complete [6], as is the question of whether r or fewer edges can be removed from G so that the remaining graph has a crossing-free drawing on two layers (Planarization) [5]. Both problems remain NP-complete when the permutation of vertices in one of the layers is given [5, 6].

Two-layer drawings are of fundamental importance to Sugiyama approach to multilayer drawing [14]. There are numerous different algorithms for planarization and crossing minimization problems, such as integer linear programming algorithms [8, 16], heuristic methods [6, 8, 14], approximation algorithms [12], and fixed parameter algorithms [3, 2].

We extend biplanar drawing method on 3D space, and instead of line layers we use the plane layers. Note that a kind of drawing similar to 3D-biplanar drawing, has been purposed before for clustered graphs [4]. We let vertices be placed in two parallel planes, and the edges can connect two vertices in the same layer or in different layers, but in each layer the induced subgraph must be planar as illustrated in Figure 2. We call such drawing *3D-biplanar*, and define 3D-biplanar cut as the number of edges between the two different layers. Our goal is to find such partition with minimum number of edges between these two partitions. In other words, we want to find 3D-biplanar cut with minimum size.

We prove that this problem is NP-complete and present a randomized parameterized algorithm for it with $O(n^k)$ time, where k is the ratio of the optimal solution to the min-cut size of graph.

This paper is organized as follows. After proving the NP-hardness of this problem in Section 2, we present our randomized parameterized algorithm in

*Department of Computer Engineering, Sharif University of Technology, `tavassoli@ce.sharif.edu`

†Department of Computer Engineering, Sharif University of Technology, and IPM School of Computer Science (No. CS1382-2-02), `ghodsi@sharif.edu`

‡Department of Mathematics and computer science, AmirKabir University of Technology, `khodakarami@aut.ac.ir`

§Department of Mathematics and computer science, AmirKabir University of Technology, `hashemi@aut.ac.ir`

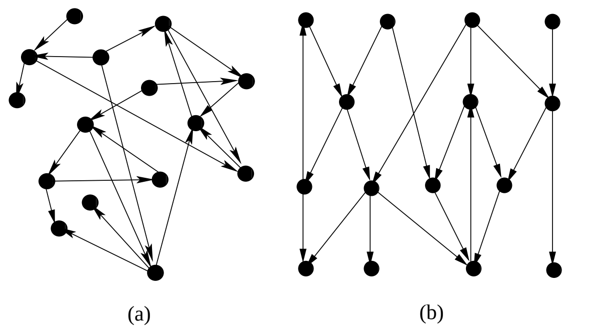


Figure 1: (a)A sample graph and (b)its 3-Layer drawing

Section 3. In Section 4 we analyze our algorithm. Finally, in Section 5 we draw some conclusions.

2 Hardness of 3D-biplanar Drawing

In this section we prove that finding a 3D-biplanar drawing with minimum 3D-biplanar cut, is NP-complete as many other layered graph drawing problems.

To prove that finding a 3D-biplanar drawing with minimum 3D-biplanar cut is NP-complete, we use a theorem from Lewis and Yannakakis [10] which is based on independent work by the two authors that actually proves a more general result. They use this result to prove that Maximum Induced Planar Subgraph [1] is NP-complete.

Theorem 1 *[10] Supposeπ is a graph property satisfying the following conditions:*

1. *There are infinitely many graphs for which π holds.*
2. *There are infinitely many graphs for which π does not hold.*
3. *If π holds for a graph G and if G' is an induced subgraph of G, then π holds for G'. This is called hereditary property.*

Then, the following problem is NP-complete: Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, is there a subset $V' \subseteq V$ with $|V'| \geq k$ such that π holds for the subgraph of G induced by V'?

Theorem 2 *Given a graph $G = (V, E)$, the problem of finding a cut C that splits vertex set V into two subsets V_1 and V_2 such that each subset being planar is NP-complete.*

Proof. Consider planarity property for a graph G. Planarity satisfies the following three conditions:

[1]Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, is there a subset $V' \subseteq V$ with $|V'| \geq k$ such that the subgraph of G induced by V' is planar?

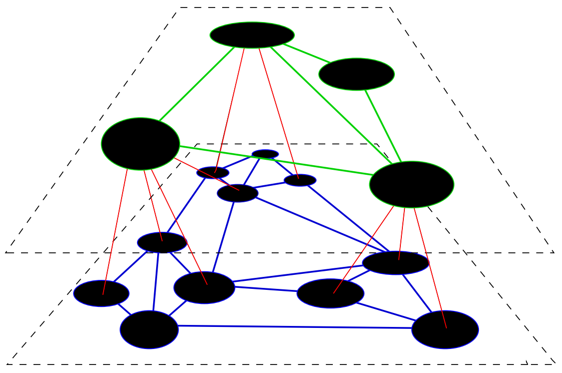


Figure 2: A 3D-biplanar drawing

1. It is straightforward that there are infinitely many planar graphs. For example all trees are planar.
2. There are infinitely many graphs that has $K_{3,3}$ or K_5 as a subgraph. Each graph that has $K_{3,3}$ or K_5 as a subgraph is not planar.
3. Each induced subgraph G' of a planar graph G is planar too.

Hence, planarity property satisfies the three conditions discussed in theorem 1. So, finding a planar induced subgraph with vertex set V' such that $|V'| \geq k$ for some $k \leq |V|$ is NP-complete. We know that $|V_1| + |V_2| = |V|$ thus one of $|V_1|$ or $|V_2|$ is greater than or equal to $|V|/2$. If we choose $k = |V|/2$, we are done. $\square$

We can't find a 3D-biplanar drawing for all graphs. So, in the following lemma we will show a necessary condition for graphs that have a 3D-biplanar drawing.

Lemma 3 *If a graph G has a 3D-biplanar drawing, then G can not contain K_9 as a subgraph.*

Proof. Suppose graph G contains K_9 as a subgraph. Hence, when we split vertex set V into two subsets V_1 and V_2, the induced subgraph with one of vertex subsets V_1 or V_2 contains K_r with $r \geq 5$ as a subgraph. We know that planar graphs can not contain K_5 as a subgraph. So graph G can not split into two induced planar subgraphs. $\square$

3 Randomized Parameterized Algorithm

In this section we introduce our main algorithm for finding a minimum 3D-biplanar cut for a graph G if it exists. With lemma 3, we can first skip graphs that doesn't have necessary condition for 3D-biplanar drawing.

We repeat the following step: pick an edge uniformly at random and merge the two vertices at its end-points as illustrated in Figure 3. If as a result there are several edges between some pairs of (newly formed) vertices, retain them all. Edges between vertices that are merged are removed, so that there are never any self-loops. We refer to this process of merging the two end-points of an edge into a single vertex as *contraction* of that edge. With each contraction, the number of vertices of G decreases by one. The crucial observation is that an edge contraction does not reduce the 3D-biplanar cut size of graph G. This is because every cut in the graph at any intermediate stage is a cut in the original graph. The algorithm continues the contraction step until only two vertices remain. At this point, the set of edges between these two vertices is a cut in G and is output as a candidate 3D-biplanar cut.

Now, there are two subsets V_1 and V_2 with a cut-edge C. Consider two induced subgraphs G_1, G_2 with respectively vertex sets V_1 and V_2. We can easily test the planarity of these two induced subgraphs by one of the planarity test algorithms like [7, 11, 13]. These algorithms take linear time in the worst case.

If G_1 or G_2 are not planar, we should repeat this algorithm to find two induced planar subgraphs. This algorithm does not always find a minimum 3D-biplanar cut even we find two induced planar subgraphs. So, we need to repeat this algorithm to achieve the minimum 3D-biplanar cut. In the next section we will compute the number of times needed to repeat this algorithm until to find a minimum 3D-biplanar cut, if it exists.

4 Analysis of the Algorithm

Let k denote the minimum biplanar cut size. We fix our attention to a particular minimum biplanar cut C with k edges. We will bound from below the probability that no edge of C is ever contracted during an execution of the algorithm, so that the edges surviving till the end are exactly the edges in C.

Suppose the min-cut size in graph $G = (V, E)$ is k'. So $k \geq k'$ and $k = hk'$ (h is a positive constant factor that is not dependent on input size, it depends on only the minimum biplanar cut size and the min-cut size). Clearly, G has at least $k'n/2$ edges; otherwise there would be a vertex of degree less than k', and its incident edges would be a min-cut of size less than k'.

Let $\mathcal{E}_i$ denote the event of not picking an edge of C at the ith step, for $1 \leq i \leq n-2$. The probability that the edge randomly chosen in the first step is in C is at most $k/(nk'/2) = 2h/n$, so that $Pr[\mathcal{E}_1] \geq 1 - 2h/n$. Assuming that $\mathcal{E}_1$ occurs, during the second step there are at least $k'(n-1)/2 = k(n-1)/2h$ edges, so the probability of picking an edge in C is at most $2h/(n-1)$, so that $Pr[\mathcal{E}_2|\mathcal{E}_1] \geq 1-2h/(n-1)$. At the ith step, the number of remaining vertices is $n-i+1$. The size of the min-cut is still at least k', so the graph has at

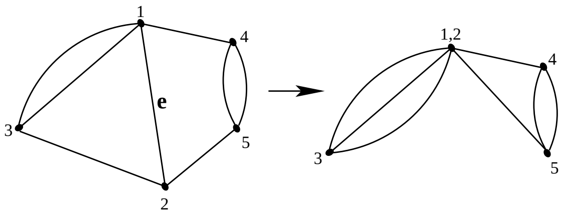


Figure 3: contraction of edge e

least $k'(n-i+1)/2 = k(n-i+1)/2h$ edges remaining at this step. Thus, $Pr[\mathcal{E}_i|\cap_{j=1}^{i-1}\mathcal{E}_j] \geq 1-2h/(n-i+1)$.

We use the basics to compute probability that no edges of C is ever picked in the process:

$$Pr[\cap_{j=1}^{n-2}\mathcal{E}_i] \geq \prod_{i=1}^{n-2}(1 - \frac{2h}{n-i+1}) \geq \frac{2}{n^{2h}}$$

The probability of discovering a particular 3D-biplanar drawing (which may in fact be the unique 3D-biplanar drawing in G) is larger than $2/n^{2h}$. Thus, our algorithm may err in declaring the drawing it outputs to be a optimum 3D-biplanar drawing. Suppose we were to repeat the above algorithm $n^{2h}/2$ times, where h is the ratio of optimal solution to min-cut size of graph, making independent random choices each time. Obviously, the probability that a min-cut is not found in any of the $n^{2h}/2$ attempts is at most

$$\left(1 - \frac{2}{n^{2h}}\right)^{\frac{n^{2h}}{2}} \leq \frac{1}{e}.$$

By this process of repetition, we have managed to reduce the probability of failure from $1 - 2/n^{2h}$ to a more respectable $1/e$. Further, executions of the algorithm will make the failure probability arbitrarily small–the only consideration being that repetitions increase the running time.

5 Conclusion

This paper introduces the concept of 3D-biplanar drawing in which a graph partitioned into two planar induced subgraphs. By straightforward reduction from a more general result, the paper shows that deciding whether a given graph can be cut into two planar graphs is NP-complete. A randomized algorithm for finding an optimal cut is given, whose running time depends on the ratio of the optimal solution and the min-cut size, as a parameter.

A further step will be to design an efficient algorithm or an approximation algorithm with a good approximation factor for finding a minimum 3D-biplanar cut.

References

[1] M. J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Trans. Syst. Man Cybern.*, SMC-10(11):705–715, 1980.

[2] V. Dujmovic, M. R. Fellows, M. T. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. A. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. In *ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms*, pages 488–499, London, UK, 2001. Springer-Verlag.

[3] V. Dujmovic, M. R. Fellows, M. T. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. A. Rosamond, M. Suderman, S. Whitesides, and D. R. Wood. A fixed-parameter approach to two-layer planarization. In *GD '01: Revised Papers from the 9th International Symposium on Graph Drawing*, pages 1–15, London, UK, 2002. Springer-Verlag.

[4] P. Eades and Q.-W. Feng. Multilevel visualization of clustered graphs. In S. North, editor, *Graph Drawing, Berkeley, California, USA, September 18-20, 1996*, pages pp. 101–112. Springer, 1997.

[5] P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science 131*, pages 361–374, 1994.

[6] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379–403, 1994.

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

[8] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. *Graph Algorithms Appl.*, 1(1):1–25, 1997.

[9] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Teubner, Stuttgart, 1990. Also published by Wiley in Chichester UK.

[10] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[11] K. Mehlhorn and P. Mutzel. On the embedding phase of the hopcroft and tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

[12] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrto. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.

[13] W.-K. Shih and W.-L. Hsu. A new planarity test. *Theor. Comput. Sci.*, 223(1-2):179–191, 1999.

[14] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man Cybernetics*, 11(2):109–125, 1981.

[15] I. G. Tollis, G. D. Battista, P. E. (Editor), and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs.* PH, 1998. ISBN: 0–133–01615–3.

[16] V. Valls, R. Marti, and P. Lino. A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs. *Journal of Operational Research*, 90:303–319, 1996.

A linear bound on the expected number of rectilinear full Steiner tree components spanning a fixed number of terminals

Christian Wulff-Nilsen[*]

Abstract

The best exact algorithm to date for the rectilinear Steiner tree problem in the plane uses a two-phase approach: in the first phase, so called full components are generated and in the second phase, a subset of these components are concatenated to form a rectilinear Steiner minimal tree. Given n points randomly distributed in a square with uniform distribution, we prove that the expected number of generated full components with Hwang form satisfying the empty lune property and spanning exactly K of the given points is $O(n)$ for fixed $K > 2$.

1 Introduction

The *rectilinear Steiner tree problem* (RSTP) in the plane asks for a tree of minimal L_1-length spanning a given finite set of points, where the L_1-metric is defined as

$$L_1(p, q) = |p_x - q_x| + |p_y - q_y|$$

for points $p = (p_x, p_y)$ and $q = (q_x, q_y)$. New points may be incorporated to shorten the tree. A feasible solution is called a *rectilinear Steiner tree* (RST) and an optimal solution is called a *rectilinear Steiner minimal tree* (RSMT). Given points are referred to as *terminals* and new points are called *Steiner points.* Figure 1 shows an example of an RSMT of a set of ten terminals.

RSMTs have applications in VLSI design where an important objective is to minimize the total length of a network of wires interconnecting a set of pins on a chip. Due to manufacturing limitations, wires are typically restricted to having horizontal and vertical orientations only, making the L_1-metric suitable for measuring wire length.

The RSTP is NP-complete [2]. Currently, the best known exact algorithm for the problem is the GeoSteiner algorithm by Warme, Winter, and Zachariasen [7]. It can solve randomly generated instances of the RSTP consisting of thousands of terminals in a reasonable amount of time.

A *full Steiner tree component* (FST) of an RST is a subtree in which all leaves are terminals and all interior nodes are Steiner points. By splitting at terminals of degree greater than one, it is easy to see that any RSMT has a (unique) decomposition into FSTs, see Figure 1. If an FST spans exactly $K \geq 2$ terminals, we refer to it as a K-FST.

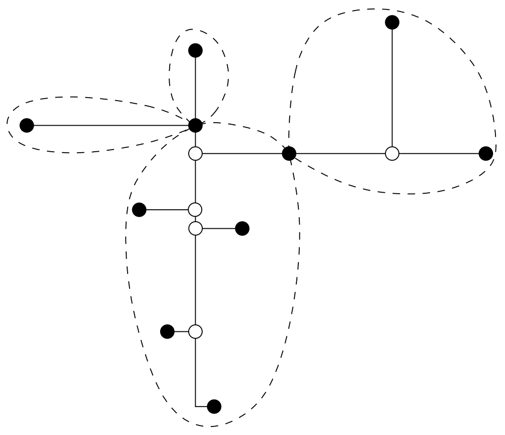


Figure 1: An RSMT of a set of ten terminals. Black nodes are terminals and white nodes are Steiner points. The RSMT consists of four full components.

The GeoSteiner algorithm solves the RSTP in two phases. In the first phase, a set $\mathcal{F}$ of FSTs is generated such that some subset of $\mathcal{F}$ is guaranteed to form an RSMT of the given set of terminals. Strong pruning techniques are applied to reduce the cardinality of $\mathcal{F}$. In the second phase, FSTs of $\mathcal{F}$ are concatenated to form an RSMT.

Although the worst-case bound on the number of generated FSTs is exponential in the number n of terminals [1], experimental results suggest that when terminals are randomly distributed in a square with uniform distribution, the expected number of FSTs generated is only $O(n)$.

A theoretical polynomial bound on the expected number of generated FSTs for randomly generated terminal sets has not yet been found. However, it has been shown that, for any $K > 2$, the expected number of K-FSTs is $O(n(\log \log n)^{K-2})$ [8], and an $O(1)$ bound on the expected number of FSTs spanning $\Omega(n)$ terminals is given in [5]. Note that the expected number of 2-FSTs is $n-1$ for randomly generated terminal sets since any 2-FST is an MST edge and since MST edges are unique with probability 1.

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

In this paper, we show that for $K > 2$, the expected number of K-FSTs generated is $O(n\pi^{K-1})$, thereby improving the bound given in [8].

2 Hwang Form of Full Components

Hwang [3] showed that FSTs of an RSMT can be assumed to have a very restricted form known as the *Hwang form*. An FST with this form consists of a *backbone* defined by two terminals, a *root* z_1 and a *tip* z_k, see Figure 2. The backbone consists of a long and a short leg. Terminal z_1 is incident to the long leg and z_k is incident to the short leg. Alternating line segments attach terminals to the long leg. There are two main types of FSTs with Hwang form: a type one FST has no terminals attached to the short leg (except the tip), and a type two FST has exactly one terminal attached to the short leg (in addition to the tip).

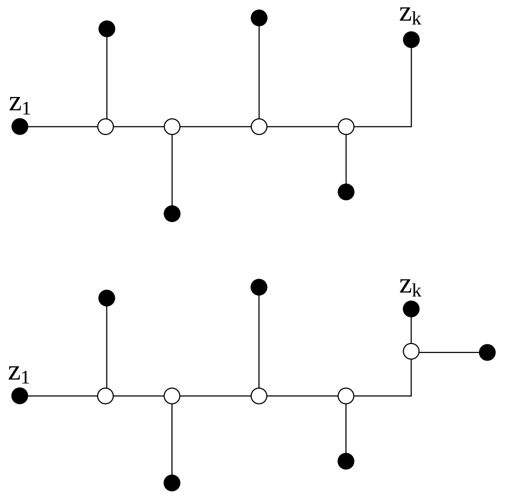


Figure 2: The two types of FSTs with Hwang form (up to rotation by a multiple of 90° and reflection through the axes).

In [8], an algorithm computing FSTs is presented. It grows FSTs along their backbones as follows. First, a root and a direction of the long leg is selected. Then a line is swept in the direction of the backbone and terminals intersected by this line are recursively attached to the backbone. At each step, various pruning techniques are applied to the partially generated FST. If pruned, the partial FST is not grown any further. If a partial FST defines an FST, it is stored in a set of candidate FSTs.

This algorithm, used in GeoSteiner, is very powerful. For random terminal sets, its observed running time is $O(n^2)$.

3 Empty Lune Property

The algorithm of [8] makes use of several properties of RSMTs, including the so called *empty lune property*. This property states that, for any edge (u, v) of an RSMT T, the *lune* $L(u, v)$ of (u, v), defined by

$$L(u, v) = \{p \in \mathbb{R}^2 | \max\{L_1(u, p), L_1(v, p)\} \leq L_1(u, v)\},$$

contains no terminals in its interior.

To see that this property holds, suppose for the sake of contradiction that a terminal z belongs to the interior of $L(u, v)$. The removal of (u, v) splits T into two components, one containing u and one containing v. Assume w.l.o.g. that z and u belong to the same component. Then adding edge (v, z) reconnects T and shortens its L_1-length, a contradiction.

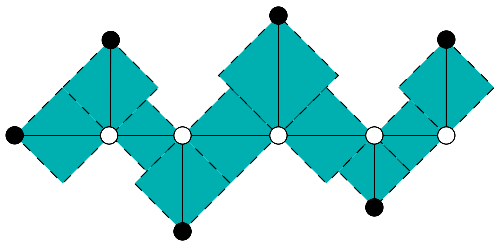


Figure 3: The lunes of the edges of a partially grown FST.

Hence, if an edge of a partially generated FST has a lune containing a terminal in its interior, the partial FST is pruned.

We will need the following Lemma.

Lemma 1 *The lunes of two distinct edges belonging to the same FST of an RSMT have disjoint interiors.*

Proof. If one of the edges is a backbone edge or if the two edges are not on the same side of the backbone then the Lemma is clearly satisfied.

Now, consider two edges (s, z) and (s', z'), where s and s' are Steiner points on the backbone and z and z' are terminals on the same side of the backbone, see Figure 4. Assume w.l.o.g. that the backbone is horizontal, that $L_1(s, z) \geq L_1(s', z')$, and that (s, z) is to the left of (s', z'). Let zp be the horizontal line segment to the left of z having length $|sz|$.

Suppose for the sake of contradiction that $L(s, z)$ and $L(s', z')$ share interior points. Then z' must belong to triangle Δszp and not to line segment sp. This implies that $L_1(z, z') < L_1(z, s)$. Removing edge (s, z) and adding edge (z, z') therefore shortens the RSMT, a contradiction. □

4 Bounding the Expected Number of FSTs

Let $K > 2$ be given. In this section, we show that the expected number of K-FSTs with Hwang form satisfying the empty lune property is linear in the number

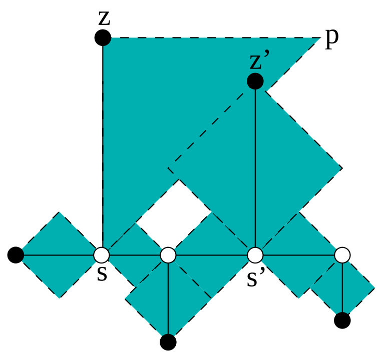


Figure 4: The lunes of an FST of an RSMT have disjoint interiors.

of terminals when terminals are randomly distributed in a square with uniform distribution.

Suppose we have grown a partial FST F with terminals $z_1, \dots, z_{k-1}$, where z_{k-1} is the last terminal added. Let z_k be the next terminal to be added.

Lemma 2 *The expected number of candidates for z_k when terminals are randomly distributed in a square with uniform distribution is less than π.*

Proof. Let $\mathcal{B}$ be the square in which the n terminals are distributed and let $\mathcal{L}$ be the union of lunes of edges in the partially grown FST F. Then the remaining $n-k+1$ terminals are randomly distributed in $\mathcal{B} \setminus \mathcal{L}$ with uniform distribution. Let A be the area of $\mathcal{B} \setminus \mathcal{L}$.

Let $z_k = (x_k, y_k)$ be a candidate terminal. For convenience, assume that the Steiner point s_{k-1} attached to z_{k-1} is located at the origin and that z_k belongs to the first quadrant. Letting s_k be the new Steiner point attached to s_{k-1} and z_k, we have $s_k = (x_k, 0)$.

Since z_k is a candidate terminal, lunes $L(s_{k-1}, s_k)$ and $L(s_k, z_k)$ contain no terminals in their interiors.

The area of $L(s_{k-1}, s_k) \cup L(s_k, z_k)$ is $\frac{1}{2}(x_k^2 + y_k^2) = \frac{1}{2}r^2$, where r is the Euclidean distance from z_k to the origin. Since $s_{k-1}, s_k, z_k \in \mathcal{B}$, at least half of $L(s_{k-1}, s_k) \cup L(s_k, z_k)$ is contained in $\mathcal{B}$. Thus, by Lemma 1, at least half of $L(s_{k-1}, s_k) \cup L(s_k, z_k)$ is contained in $\mathcal{B} \setminus \mathcal{L}$.

By the above, if a terminal in the first quadrant has Euclidean distance r to the origin, the probability that it is a z_k-candidate is no more than

$$\left(1 - \frac{r^2}{4A}\right)^{n-k+1},$$

and we see that $1 - r^2/(4A) > 0$, giving the upper bound $2\sqrt{A}$ on r.

Given $r, h > 0$ and letting $|p|$ denote the Euclidean distance from point p to the origin, the region

$$C(r,h) = \{p \in \mathbb{R}_+^2 | r \le |p| \le r+h\}.$$

has area

$$\frac{\pi}{4}((r+h)^2 - r^2) = \frac{\pi}{2}rh + \frac{\pi}{4}h^2.$$

A terminal in $C(r,h)$ has Euclidean distance at least r to the origin, see Figure 5.. By the above, the

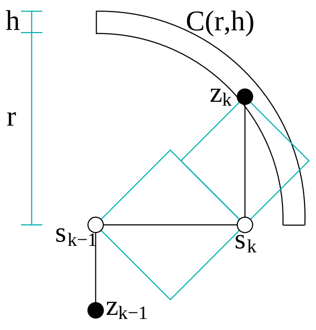


Figure 5: The situation in the proof of Lemma 2.

probability that it is a z_k-candidate is no more than $(1 - r^2/(4A))^{n-k+1}$.

Since the expected number of terminals in $C(r,h)$ is at most

$$(n-k+1)\left(\frac{\pi}{2}rh + \frac{\pi}{4}h^2\right)/A,$$

the expected number of candidate terminals in $C(r,h)$ is bounded by

$$\left(1 - \frac{r^2}{4A}\right)^{n-k+1} (n-k+1)\left(\frac{\pi}{2}rh + \frac{\pi}{4}h^2\right)/A.$$

Integrating, we obtain a bound E on the expected number of z_k-candidates,

$$\begin{aligned} E &\le \int_{r=0}^{2\sqrt{A}} \left(1 - \frac{r^2}{4A}\right)^{n-k+1} (n-k+1)\frac{\pi}{2A} r dr \\ &= \frac{\pi(n-k+1)}{2A} \int_{r=0}^{2\sqrt{A}} \left(1 - \frac{r^2}{4A}\right)^{n-k+1} r dr \\ &= \frac{\pi(n-k+1)}{2A} \left[-2A \frac{\left(1 - \frac{r^2}{4A}\right)^{n-k+2}}{n-k+2} \right]_{r=0}^{2\sqrt{A}} \\ &= \frac{\pi(n-k+1)}{n-k+2} \\ &< \pi. \end{aligned}$$

□

Our main result follows easily from Lemma 2.

Theorem 3 *Given n terminals randomly distributed in a square with uniform distribution, the expected number of K-FSTs satisfying the empty lune property is $O(n\pi^{K-1})$ for any $K > 2$.*

Proof. Let z be a terminal. By Lemma 2, the expected number of K-FSTs of type one with root z is less than

$$8\pi^{K-1},$$

since there are four possible directions of the backbone, and for each direction, there are two quadrants in which the first backbone terminal can be chosen. The expected number of K-FSTs of type two with root z is $O(\pi^{K-2})$ since the terminal attached to the short leg is unique once the other terminals are chosen [8]. Since z can be chosen in n ways, the theorem follows. $\square$

5 Conclusion

Given n terminals randomly distributed in a square with uniform distribution, we proved that the expected number of K-FSTs of these terminals with Hwang form satisfying the empty lune property is $O(n\pi^{K-1})$.

6 Future research

Lemma 2 gives the upper bound π on the expected number of candidates for the next terminal to be added to the backbone. If this could be improved to a constant below 1, it would give a proof that the total expected number of full components generated is linear in the number of terminals when terminals are randomly distributed in a square with uniform distribution. To obtain such an improved constant bound, other properties of RSMTs, such as the empty corner rectangle property and the bottleneck property [8], should be considered.

Acknowledgments

I thank Pawel Winter and Martin Zachariasen for their comments and remarks.

References

[1] U. Fößmeier and M. Kaufmann. *On Exact Solutions for the Rectilinear Steiner Tree Problem.* Technical Report WSI-96-09, Universität Tübingen, 1996.

[2] M. R. Garey and D. S. Johnson. *The Rectilinear Steiner Tree Problem is NP-Complete.* SIAM J. Appl. Math, 32(4): 826-834, 1977.

[3] F. K. Hwang. *On Steiner minimal trees with rectilinear distance.* SIAM J. Appl. Math, 30: 104-114, 1976.

[4] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* Annals of Discrete Mathematics 53, Elsevier Science Publishers, Netherlands, 1992.

[5] J. S. Salowe and D. M. Warme. *Thirty-Five Point Rectilinear Steiner Minimal Trees in a Day.* Networks, 25, 1995.

[6] D. M. Warme *Spanning Trees in Hypergraphs with Applications to Steiner Trees.* Ph.D. Thesis, Computer Science Dept., The University of Virginia, 1998.

[7] D. M. Warme, P. Winter, and M. Zachariasen. *GeoSteiner 3.1.* Department of Computer Science, University of Copenhagen, http://www.diku.dk/geosteiner/, 2001.

[8] M. Zachariasen. *Rectilinear Full Steiner Tree Generation.* Networks, 33: 125-143, 1999.

Planar Bichromatic Minimum Spanning Trees*

Magdalene G. Borgelt† Marc van Kreveld† Maarten Löffler† Jun Luo† Damian Merrick‡
Rodrigo I. Silveira† Mostafa Vahedi†

Abstract

Given a set S of n red and blue points in the plane, a *planar bichromatic minimum spanning tree* is the shortest possible spanning tree of S, such that every edge connects a red and a blue point, and no two edges intersect. Computing this tree is NP-hard in general. We present an $O(n^3)$ time algorithm for the special case when all points are in convex position. For the general case, we present a factor $O(\sqrt{n})$ approximation algorithm.

1 Introduction

Let S be a set of n points in the plane, where every point has one of two possible colors (red or blue). In computational geometry, several papers have discussed problems that concern such a bichromatic input, see for instance Kaneko and Kano [5] for an overview. This paper discusses bichromatic spanning trees. We obtain a spanning tree T of S by finding a set of $n-1$ edges, which connect points of different colors ("color conforming") and form an acyclic connected component. If T does not contain intersections it is a *planar* spanning tree. In this paper we assume that no three points are collinear, otherwise a planar bichromatic spanning tree does not always exist.

A *minimum weight spanning tree* (MST) of S is a spanning tree of minimum total length. Note that a MST needs not be unique. It is well known that the (monochromatic) MST of a set of points in the plane can be found using a greedy algorithm like Kruskal's [6]. Kruskal's algorithm adds edges in the order of increasing length, and discards edges that would create a cycle in the graph built so far. The (monochromatic) MST of a set of points or line segments in the plane cannot contain intersections [2, 4]. It is shown that the problem of finding the Euclidean (monochromatic) MST of a set of n points in d-dimensional space is related to the problem of finding the bichromatic closest pair among n red and m blue points in d-dimensional space [1].

Recently it was shown that a color conforming spanning tree of S can always be found [4]. It was also shown by illustration that the MST of a given point set S may contain intersections if one uses a greedy algorithm like Kruskal's [3]. Modifying Kruskal's algorithm to check for intersections and discarding an edge if it causes an intersection, leads to a greedy algorithm which we will refer to as the *greedy planar algorithm* or the *augmented Kruskal algorithm*. It was shown that the greedy planar algorithm does not always yield the optimal planar solution [3], and can even be a linear factor off. The problem of finding a superlinear bound for the ratio of the weight of the greedy planar solution to the weight of the optimal planar solution was left open. Another open problem was to find an approximation algorithm for the planar bichromatic MST of S.

In this paper we show that a planar bichromatic spanning tree of a set of red and blue points may not always be obtainable using a greedy algorithm like the augmented Kruskal algorithm (hence, this algorithm has no approximation factor at all). We can show that the planar bichromatic minimum spanning tree problem is NP-hard in the general case (the proof is in the full paper), but we show that the optimal tree can be constructed in cubic time when the points are in convex position. Finally, we present an approximation algorithm that computes an $O(\sqrt{n})$-approximation in $O(n^2)$ time.

2 Greedy Planar Bichromatic Spanning Trees

One approach to create planar bichromatic spanning trees is by using a greedy algorithm. The algorithm proposed by Kruskal [6] and augmented for bichromatic trees [3] is an example of this. In this section we show that a greedy algorithm may in some cases not find any planar bichromatic spanning tree at all,

*This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503, and also through the project GOGO.

†Department of Information and Computing Science, Utrecht University, the Netherlands, { magdalene, marc, loffler, ljroger, rodrigo, vahedi}@cs.uu.nl

‡National ICT Australia; School of Information Technologies, University of Sydney, Australia, dmerrick@it.usyd.edu.au

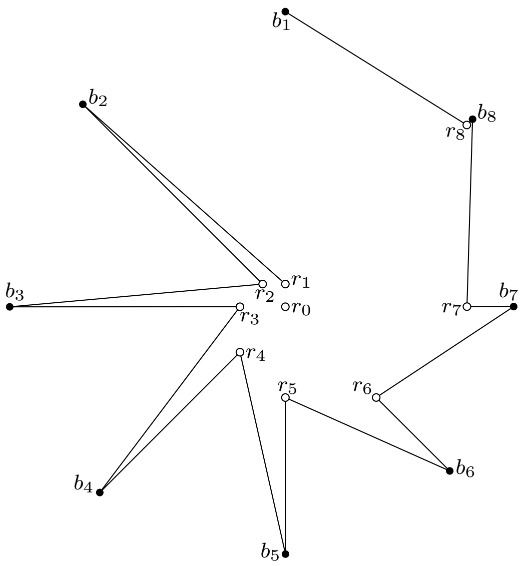


Figure 1: An example of a set of bichromatic vertices and bichromatic edges that are selected by an augmented Kruskal algorithm.

because at some stage there are points that cannot be connected to any point of the other color anymore. Specifically, we show that Kruskal's algorithm may get stuck in this way.

Consider the set of nine red points (r_i, $i = 0, \dots, 8$) and eight blue points (b_i, $i = 1, \dots, 8$), as in Figure 1. Here the point b_i lies slightly to the left of the directed line from r_0 to r_i, for any i, so once all the black edges in the figure have been included by some greedy algorithm, the central point r_0 cannot be connected to any blue point anymore, and therefore the existing tree cannot be extended to a valid planar bichromatic spanning tree.

The augmented Kruskal algorithm adds edges in the order of increasing length, discarding edges that would cause intersections as well as edges that would create a cycle in the graph built so far. In this situation, the algorithm will create the edges shown in the figure. This is because for any edge from r_0 to one of the blue points, there is another edge crossing it that is shorter and therefore will be chosen first.

To see why this happens, consider for any i the group of points r_{i-1}, r_i, b_{i-1}, and b_i. Of the two edges (r_{i-1}, b_i) and (b_{i-1}, r_i) the former must be shorter in order to obtain the desired structure (that is, a structure in which the edge (r_{i-1}, b_i) shields r_0 from seeing b_{i-1}, so that (r_0, b_{i-1}) would lead to an intersection). To simplify the situation, let us assume that r_0, r_{i-1}, and b_{i-1} are collinear, and that r_0, r_i, and b_i are collinear. In addition, let us assume that the angle

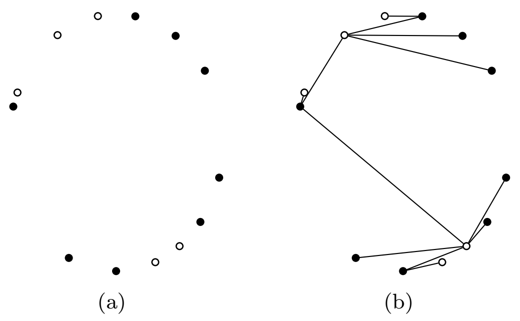


Figure 2: Example of the input (left) and output (right) for a set of bichromatic points in convex position.

between the two directions is exactly $\frac{\pi}{4}$. If we place the red points on a spiral where the distance between r_0 and r_i is $\sqrt{2}$ times larger than the distance between r_0 and r_{i-1}, and we place the b_i at carefully chosen locations at the outside of the construction, we get the required property.

Theorem 1 *A planar bichromatic minimum spanning tree of a set of red and blue points may not always be obtainable using a greedy algorithm like the augmented Kruskal algorithm.*

3 NP-hardness

We show in the full paper that the problem is NP-hard, by reduction from planar 3-SAT.

Theorem 2 *The problem of computing a planar bichromatic minimum spanning tree of a set of red and blue points, is NP-hard.*

4 Dynamic programming for points in convex position

If we have a set of red and blue points in the plane that are in convex position, then we can compute the planar bichromatic minimum spanning tree in $O(n^3)$ time, see Figure 2(b). This is because in this case, any edge of the tree partitions the plane into two independent subproblems.

We are given a set of points in convex position, see Figure 2(a). Let them be ordered cyclicly counterclockwise. For any pair of points p and q (possibly of the same color, but not necessarily) we define T_{pq} as the planar bichromatic minimum spanning tree of the points $\{p, \dots, q\}$ (the set of points encountered when walking from p to q, in counterclockwise order). Then the answer for the whole point set is T_{pq}, for any pair of consecutive points q and p.

We can compute all lengths of T_{pq} by dynamic programming, starting from the simplest ones. The length of T_{pp} is zero. For a given pair of points $p \neq q$ we observe the following cases:

- p and q are neighbors. If they have different colors, then $T_{pq} = \overline{pq}$. Otherwise, T_{pq} does not exist.
- p and q are of the same color. There exists a point r between p and q such that $T_{pq} = T_{pr} \cup T_{rq}$, or T_{pq} does not exist.
- p and q are of different colors, but they are not connected by a direct edge. There exists a point r between p and q such that $T_{pq} = T_{pr} \cup T_{rq}$.
- p and q are of different colors, and they are connected by a direct edge. There exist two neighboring points r and s (one of which may be p or q), such that $T_{pq} = \overline{pq} \cup T_{pr} \cup T_{sq}$.

In any case, we can compute T_{pq} in linear time by guessing the position of r and taking the best over the possible results. Since there are $O(n^2)$ possible choices for p and q, we solve the problem in $O(n^3)$ time.

Theorem 3 *A planar bichromatic minimum spanning tree of a set of n red and blue points in convex position can be computed in $O(n^3)$ time.*

5 An $O(\sqrt{n})$-approximation

Since the problem cannot be solved exactly in polynomial time unless P = NP, we will now describe an approximation algorithm that computes a planar bichromatic spanning tree that is at most $O(\sqrt{n})$ times larger than the optimal one, in polynomial time.

We start by taking a bounding square around the point set, such that the set either touches the square on both the top and bottom edges, or on both the left and right edges. After this we scale the input, such that the bounding square becomes the unit square.

Lemma 4 *The length of the optimal planar bichromatic spanning tree of the scaled point set is at least 1.*

Proof. There are two points on opposite borders of the bounding squares, so they are at least 1 away from each other. □

Next, we create a grid by dividing the unit square into $\sqrt{n} \times \sqrt{n}$ square cells, of side $\frac{1}{\sqrt{n}}$. The point set is also divided by the grid, see Figure 3(b).

Lemma 5 *If m of the n grid cells contain at least a point, then the length of the optimal planar bichromatic spanning tree is at least $O(\frac{m}{\sqrt{n}})$.*

Proof. If m grid cells contain a point, then at least $\frac{1}{4}m$ cells that are not adjacent contain a point. To connect these points with any spanning tree, a connection of length at least $\frac{1}{\sqrt{n}}$ is needed per grid cell. □

We classify the cells according to what points are inside them. There are three possibilities. Each grid cell is either empty, contains only points of one color, or contains points of both colors.

We will now define a set of *core regions* as follows. A core region is either a single grid cell, two adjacent grid cells or four grid cells adjacent to a grid vertex. Each core region contains both red and blue points. Apart from that, the following crucial property should hold.

Crucial Property: Every point is either a distance of at most $O(1/\sqrt{n})$ (a constant number of grid cells) away from a core region, or a distance of at least $1/\sqrt{n}$ (one grid cell) away from the closest point of the other color.

This crucial property guarantees the approximation factor: points in the first category will be connected by an edge of $O(1/\sqrt{n})$ length, so all of these together will not be more than $O(\sqrt{n})$. Points in the second category will be connected by an edge of length at most $O(1)$, but in the optimal solution they are connected by some edge of at least length $\Omega(1/\sqrt{n})$, so they are at most a factor $O(\sqrt{n})$ too long.

We compute the core regions iteratively as follows.

- Every grid cell that contains points of two colors is a core region.
- For every grid cell that contains only points of one color and that is not adjacent to a core region, do the following: if it has a neighboring grid cell that contains points of the other color, make a new core region out of those two grid cells (and possibly two more if they were vertex-adjacent), otherwise do nothing.

This procedure results in a set of core regions with the properties just described. Figure 3(c) gives an example.

We want to get rid of the core regions that touch each other, so we compute a maximal independent set of the core regions with respect to the adjacency (edge or vertex) relation. After this we have a smaller set of core regions, with the property that the regions are separated by a band at least one grid cell wide. Furthermore, the discarded core regions are all adjacent to a core region that belongs to the independent set, so the crucial property still holds.

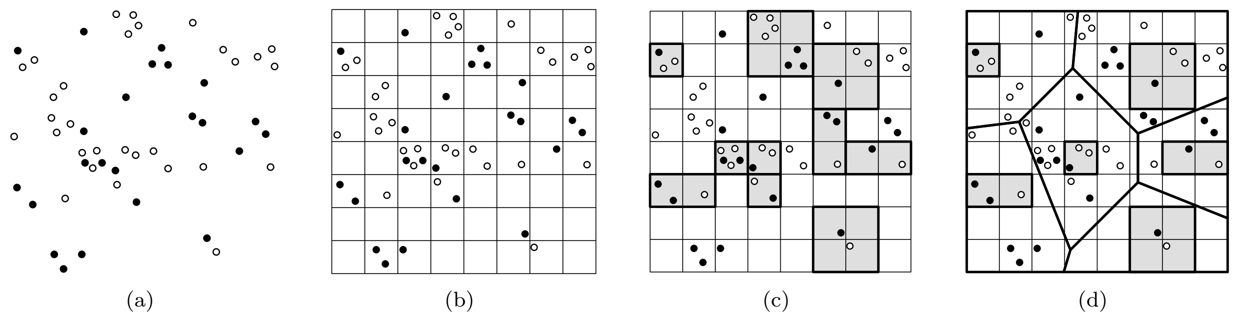


Figure 3: (a) A set of red and blue points. (a) The points divided by a grid. (b) Core regions are shaded and bounded by fat edges. (c) The Voronoi regions of an independent subset of the core regions.

Next, we compute the Voronoi diagram of the centers of the core regions. This is a subdivision of the plane into convex cells that all have one core region inside them, because any point inside a core region is at most $\sqrt{2}/\sqrt{n}$ away from the center of its own region, and at least $1.5/\sqrt{n}$ away from the center of any other region, see Figure 3(d).

Inside each Voronoi cell we compute a factor $O(\sqrt{n})$ approximation spanning tree as follows. First we compute any tree for the points inside the core region. Then we sort the other points on their distance to the core region, and add them in that order. This is always possible, as the following lemma shows.

Lemma 6 *If T is a planar bichromatic spanning tree, and q is a point in one of the colors that is outside the convex hull of T, then T can be extended by one more edge to a planar bichromatic spanning tree that includes q.*

The insertion order ensures that the length of each new connection is at most the distance to the core region plus the size of the core region times some constant. This means that the points that are near the borders of the core regions will have a connection of length $O(\frac{1}{\sqrt{n}})$, which is what we need for the approximation.

After building the trees inside all Voronoi cells, we combine them and extend them to one tree using the $O(n \log n)$ algorithm described in [4]. The length of the edges used by the algorithm to connect each component does not matter, because the number of Voronoi cells is at most the number of grid cells with points inside them, m, and even if all these connections are as bad as possible, we still only have a length of at most m. By Lemma 5, this is a factor $O(\sqrt{n})$ approximation of the optimum.

All steps of the algorithm take quadratic time in the number of occupied grid cells, so the algorithm runs in $O(n^2)$ time.

Theorem 7 *An $O(\sqrt{n})$-approximation of a planar bichromatic minimum spanning tree of a set of n red and blue points can be computed in $O(n^2)$ time.*

6 Conclusions

We studied the problem of computing a planar bichromatic minimum spanning tree of a set of red and blue points in the plane. We showed that this problem is NP-hard, and gave an $O(\sqrt{n})$-approximation algorithm. An interesting open problem is whether a constant factor approximation algorithm for this problem that runs in polynomial time exists.

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Discrete and Computational Geometry*, 6(5):407-422, 1991.

[2] P. Bose and G. Toussaint. Growing a Tree from its Branches. *Journal of Algorithms* 19(1):86-103, 1995.

[3] M. Grantson, H. Meijer, and D. Rappaport. Bi-Chromatic Minimum Spanning Trees. In *Proc. 21st European Workshop on Computational Geometry (EuroCG05)*, pages 199–202, 2005.

[4] F. Hurtado, M. Kano, D. Rappaport and C.D. Tòth. Encompassing Colored Crossing-Free Geometric Graphs. In *16th Canadian Conference on Computational Geometry*, pages 48–52, 2004.

[5] A. Kaneko and M. Kano. Discrete Geometry on Red and Blue Points in the Plane – A Survey. In *Discrete and Computational Geometry* (B. Aronov et al., eds.), Springer-Verlag, Berlin, pages 551–570, 2004.

[6] J.B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proc. of the American Mathematical Society,* 7:48–50, 1956.

Transforming Spanning Trees: A Lower Bound

Kevin Buchin[*] Andreas Razen[†] Takeaki Uno[‡] Uli Wagner[†]

Abstract

For a planar point set we consider the graph of crossing-free straight-line spanning trees where two spanning trees are adjacent in the graph if their union is crossing-free. An upper bound on the diameter of this graph implies an upper bound on the diameter of the flip graph of pseudo-triangulations of the underlying point set.

We prove a lower bound of $\Omega\big(\log(n)/\log(\log(n))\big)$ for the diameter of the graph of spanning trees on a planar set of n points. This nearly matches the known upper bound of $O(\log(n))$. If we measure the diameter in terms of the number of convex layers k of the point set, our lower bound construction is tight, i.e., the diameter is in $\Omega(\log(k))$ which matches the known upper bound of $O(\log(k))$. So far only constant lower bounds were known.

1 Introduction

Given a set S of n points in the plane let T_S denote the set of all crossing-free straight-line spanning trees of S. A straight-line embedded graph is *crossing-free* if every pair of its edges does not share any point other than common endpoints. We call two crossing-free spanning trees T_1 and T_2 of S *compatible* if their union, i.e. the graph on S with edge set $E(T_1) \cup E(T_2)$, is crossing-free.

Aichholzer, Aurenhammer, and Hurtado [2] investigate how fast two spanning trees can be transformed into each other by a sequence of spanning trees with any two consecutive trees being compatible. They prove that the maximum length of a sequence needed is in $O(\log(n))$.

Let $\mathcal{T}_S$ denote the graph with T_S as vertex set and edges between compatible spanning trees. The maximum length of a sequence needed to transform two spanning trees corresponds to the diameter of this graph. Aichholzer, Aurenhammer, Huemer, and Krasser [1] refine the above bound on the diameter of $\mathcal{T}_S$ to a bound of $O(\log(k))$, where k denotes the number of convex layers of S. The *convex layers* of a point set S are defined inductively: the first convex layer U_1 consists of the boundary points of the convex hull of S and, for $i > 1$, the i-th convex layer U_i is defined as the set of boundary points of the convex hull of $S \setminus \bigcup_{j<i} U_j$. The number k of convex layers of a point set S is the minimum i such that $U_{i+1} = \emptyset$.

Aichholzer et al. [1] also prove that an upper bound of d on the diameter of $\mathcal{T}_S$ yields an upper bound of $O(nd)$ on the diameter of the flip graph of pseudo-triangulations of S. They conjecture that the diameter of $\mathcal{T}_S$ is sublogarithmic. So far no example was known where the diameter is not constant.

We give a sublogarithmic but considerably tighter lower bound: we complement the $O(\log(n))$ upper bound with a lower bound of $\Omega\big(\log(n)/\log(\log(n))\big)$. We do this constructively by providing point sets of increasing size, and on each point set we specify two spanning trees achieving this bound. For these examples the bound in the number of convex layers is tight, i.e., the distance between the two trees is in $\Omega(\log(k))$, where k is the number of convex layers.

2 The Lower Bound

In this section we construct point sets in the plane and consider pairs of spanning trees which need a large number of transformation steps to transform one tree into the other.

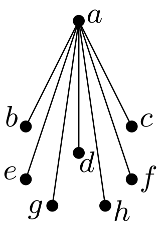
We will first develop a general scheme to construct such trees. Based on this we present two recursive constructions using this scheme in different ways. The first yields a lower bound of $\Omega(\sqrt{\log(n)})$ on the number of transformations needed, where n is the size of the point set. The second gives a lower bound of $\Omega\big(\log(n)/\log(\log(n))\big)$. Both constructions use point sets with more than two points on a line, i.e., the points are not in general position. However, they can easily be changed to do so by applying a small perturbation, without losing any of the relevant properties of the construction.

The basic concept of the constructions is that by placing the topmost vertex of the point set very far away from the others, we consider a first tree with only near vertical edges and a second tree with many near horizontal edges crossing the vertical edges of the first tree. Furthermore, there are dependencies between the horizontal edges such that, when transforming one tree into the other, a vertex that connects to the rest of the tree by a horizontal edge may con-

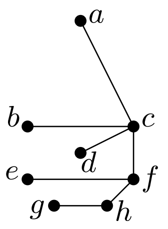
[*]Institute of Computer Science, Freie Universität Berlin, buchin@inf.fu-berlin.de

[†]Institute of Theoretical Computer Science, ETH Zurich, {arazen, wagneru}@inf.ethz.ch

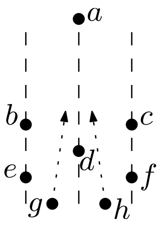
[‡]National Institute of Informatics, Tokyo, Japan, uno@nii.jp



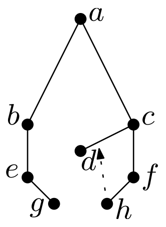
(a) T_1



(b) T_2



(c) Two vertical strips defined by $b, c, \ldots, f$



(d) One vertical strip is blocked

Figure 1: The trees T_1 and T_2 have distance 3 in $\mathcal{T}_S$.

nect to the topmost vertex by a vertical edge only if certain other horizontal edges are no longer in the tree.

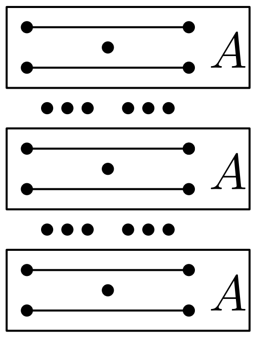
We illustrate this by the example in Figure 1 with $S = \{a, b, \ldots, h\}$ being the underlying point set. The first tree T_1 (Figure 1(a)) has near vertical edges, the second tree T_2 (Figure 1(b)) has mostly near horizontal edges. The points b, c, d, e, and f subdivide the space in which the points of S lie into two vertical strips. In each such strip there is one point at the bottom (g and h) which needs to connect to the topmost point a through the corresponding strip (Figure 1(c)). At the beginning the edges $\{b, c\}$ and $\{e, f\}$ block both strips completely, i.e., the bottommost points g and h cannot connect to a in any neighbor of T_2 in $\mathcal{T}_S$. Furthermore, whatever the first transformation is, thereafter the point d will have an edge to at least one of b, c, e, or f (as in the example of the tree in Figure 1(d)). Thus, after one transformation the edge $\{a, g\}$ or $\{a, h\}$ still crosses an edge of the current tree and cannot be present in the next transformation. In total, three transformations are necessary and also suffice to transform T_2 to T_1, and the diameter of $\mathcal{T}_S$ is at least 3.

2.1 Blocking Vertical Strips

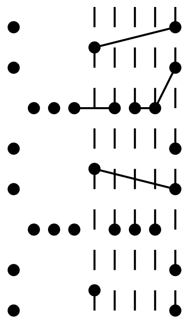
Before turning to the construction of a point set, we further develop the concept of blocking vertical strips. A *vertical strip* R is a subset of $\mathbb{R}^2$ such that there exist $a, b \in \mathbb{R}$ with

$$R = \{(x, y) \in \mathbb{R}^2 \,|\, a \le x \le b\} =: [a, b] \times \mathbb{R};$$

the *width* of the vertical strip R is $b - a$. An edge *blocks a vertical strip* if the end points of the edge lie on different sides (possibly on the border) of the strip. For instance, in Figure 1(b) (assuming a proper coordinate system) the edges $\{b, c\}$ and $\{e, f\}$ of the tree T_2 both block the vertical strip $[0, 1] \times \mathbb{R}$ and the edge $\{d, c\}$ blocks the vertical strip $[1/2, 1] \times \mathbb{R}$.



(a) Stacking three 1-of-2-blockers



(b) After 1 transformation

Figure 2: A 3-of-8-blocker after 2 steps.

A point set A together with a set E of straight-line edges on A *blocks a vertical strip of width* $w > 0$ *after* k *steps*, if for any point set S containing A (and no further point in the convex hull of A) the following holds: if a spanning tree $T \in \mathcal{T}_S$ contains the edges E then in any spanning tree in the k–neighborhood of T in $\mathcal{T}_S$ some vertical strip of width at least w is blocked (not necessarily by an edge in E). For instance, in T_2 in Figure 1(b) the points b, c, d, e, and f together with the edges $\{b, c\}$ and $\{e, f\}$ block a vertical strip of width 1/2 after 1 step: either the strip $[0, 1/2] \times \mathbb{R}$ is blocked by the edge $\{b, d\}$ or $\{e, d\}$, or the strip $[1/2, 1] \times \mathbb{R}$ is blocked by $\{d, c\}$ or $\{d, f\}$.

Note that this concept now implies the following: assume that we have a point set S with the topmost point $p_0 \in S$ placed very far away from the rest, and $A \subset S$ with edges E on A blocks some vertical strip R after k steps. Let $T_1 \in \mathcal{T}_S$ be the tree where p_0 connects to every other point by a near vertical edge and let $T_2 \in \mathcal{T}_S$ contain the edges E. If there is a point in $S \cap R$ lying strictly below the edge responsible for blocking R after k steps then T_1 cannot be in the $(k+1)$–neighborhood of T_2 in $\mathcal{T}_S$. Thus, the diameter of $\mathcal{T}_S$ is at least $k + 2$.

The point sets we are about to construct reside in the strip $[0, 1] \times \mathbb{R}$, and therein we consider specific vertical strips that might be blocked. We call a point set A together with a set of edges E an *l-of-m-blocker after k steps* if A blocks at least l of the vertical strips $[(i-1)/m, i/m] \times \mathbb{R}$ (for $i = 1, \ldots, m$) after k steps, not necessarily the same strips for different trees containing E in their respective k–neighborhood in $\mathcal{T}_S$. In the example of T_2 in Figure 1(b) the points b, c, d, e, f together with the edges $\{b, c\}, \{e, f\}$ are a 1-of-2-blocker after 1 step. We call l/m the *density* of the blocker.

Given a blocker we can construct further blockers with a larger number of steps by stacking the blocker and spreading in further points. Consider for instance the construction in Figure 2(a). It contains three copies of the 1-of-2-blocker after 1 step, A, together with the corresponding horizontal edges, and between two adjacent copies of A additional points subdivide each (of two) strips into four smaller strips resulting in a total of eight vertical strips. After 1 step each copy of A blocks one vertical strip of width 1/2. Since there are three copies of A by the pigeon-hole principle one strip is blocked twice (in the example, Figure 2(b), the right vertical strip). No matter how the points in-between these blocking edges are connected to the rest of the tree at least three of the four corresponding vertical strips of width 1/8 are blocked, and this can only change after the edges blocking the strip of width 1/2 are removed. This is the case at the earliest after 2 steps, thus the construction is a 3-of-8-blocker after 2 steps, and for a point set S containing this blocker the diameter of $\mathcal{T}_S$ is at least 4.

This construction is generalized in the following.

Lemma 1 *Let A be an l-of-m-blocker after k steps of density $l/m > 1/u$ for some $u \in \mathbb{N}$. Stacking u copies of A on top of each other with additional points (equidistantly) subdividing each of the m vertical strips into m' vertical strips between each pair of adjacent copies yields an $(m'-1)$-of-$(m \cdot m')$-blocker after $k+1$ steps.*

Proof. After k steps the u copies of A block within the m vertical strips $l \cdot u > m$ times, thus at least one of the m strips is blocked twice. The points in this vertical strip blocked from above and below subdivide this strip into m' smaller strips, hence in order to connect these points to the rest at least $m'-1$ of the small strips are blocked. This changes at the earliest after $k+1$ steps, thus the construction is an $(m'-1)$-of-$(m \cdot m')$-blocker after $k+1$ steps. □

2.2 Construction 1

We construct a point set S depending on an integer variable d together with two trees $T_1, T_2 \in T_S$ such that at least d steps are needed to transform one of the trees into the other, and the size of S is in $O(2^{d^2})$, i.e., $d \in \Omega(\sqrt{\log(n)})$, where $n = |S|$.

All points of S lie in the infinite strip $[0,1] \times \mathbb{R}$. A special point p_0 has a larger y-coordinate than all other points, and will be chosen such that the slope of any line through p_0 and any other point in S is larger than the slopes of all non-vertical lines through two points in $S \setminus p_0$.

Let L_0 be defined as $L_0 := \{(0,0),(1,0)\}$ and L_k for $k \in \mathbb{N}, k \geq 1$ as

$$L_k := \left\{ \left(\frac{2i-1}{2^k}, 0\right) \middle| i = 1, \dots, 2^{k-1} \right\}.$$

Thus, $\bigcup_{0 \leq k' \leq k} L_{k'}$ subdivides the line segment from $(0,0)$ to $(1,0)$ into 2^k equal parts by 2^k+1 points. The set L_{k+1} places one point in the center of each of these parts.

We define point sets A_k, $k \in \mathbb{N}$, inductively. Let

$$A_1 := L_0 \,\cup\, L_1 \oplus 1 \,\cup\, L_0 \oplus 2,$$

where $P \oplus t := \{(x, y+t) | (x,y) \in P\}$ is a vertical shift of the point set $P \subset \mathbb{R}^2$ by $t \in \mathbb{N}$. Note that A_1 corresponds to the point set A from Figure 2(a).

For $k \in \mathbb{N}$, let A_{k+1} be defined by stacking 2^k+1 copies of A_k with a copy of L_{k+1} between each pair of adjacent copies of A_k. Formally,

$$\begin{aligned} A_{k+1} \;:=\; & \bigcup_{i=0}^{2^k} A_k \oplus i \cdot (h_k + 1) \\ & \cup \bigcup_{i=0}^{2^k-1} L_{k+1} \oplus \big(i \cdot (h_k+1) + h_k\big), \end{aligned}$$

where $h_k := 2 \cdot \prod_{i=0}^{k-1} (2^i + 1) - 1$.

It follows directly from Lemma 1 that the point set A_k together with edges between every pair of points with coordinates $(0,y),(1,y)$, for some $y \in \mathbb{N}$, is a 1-of-2^k-blocker after k steps.

Given $d \in \mathbb{N}$ define $S := L_{d+1} \cup A_d \oplus 1 \cup \{p_0\}$ with p_0 chosen as described above. Let T_1 be the star connecting p_0 to every other point by an edge. Let T_2 be a tree on S obtained by taking all (exactly) horizontal edges blocking the complete vertical strip $[0,1] \times \mathbb{R}$ and adding further edges such that T_2 is a crossing-free straight-line spanning tree. We already know that A_d together with the corresponding horizontal edges is a 1-of-2^d-blocker after d steps. Thus, when transforming T_2 into T_1 there will be one of the points in L_{d+1} blocked away from p_0 after d steps. Therefore, at least $d+2$ transformations are needed.

The cardinality s_d of A_d is given by $s_1 = 5$ and the recursion $s_{k+1} = (2^k+1)s_k + 2^k \cdot 2^k$. Thus, we have $s_{k+1} \leq 2^{2k+1} s_k$ and by induction $s_d \leq 5 \cdot 2^{d^2}$. The size of S is $2^d + s_d + 1$, hence $d \in \Omega(\sqrt{\log(|S|)})$.

Next we consider the number of convex layers. The first layer of S consists of the topmost point, the points of the bottom row, the points in the left most and the right most column of points. With each additional convex layer two more rows and two more columns are considered until only one row or one column is left. If m_1 is the number of different x-coordinates used and m_2 the number of different y-coordinates used in the construction then we can bound the number of convex layers from above by

$$1 + \frac{1}{2} \min(m_1, m_2).$$

The number of different x-coordinates in S is bounded by 2^d, thus d is logarithmic in the number of convex layers.

Theorem 2 *There is a point set S in the plane for which the diameter of $\mathcal{T}_S$ is in $\Omega(\log(k))$, where k is the number of convex layers of S.*

2.3 Construction 2

The point set A_k from Construction 1 suffered from an exponential growth in both, the number of copies of A_{k-1} and the number of points in L_k placed in-between. Note that the recursive construction we present in the following will only require a constant number of copies of previously constructed point sets.

We construct a point set $S \subset [0,1] \times \mathbb{R}$ depending on an integer variable $d > 1$ together with two trees $T_1, T_2 \in T_S$ such that $d \in \Omega\big(\log(n)/\log(\log(n))\big)$, where n is the size of S, and the distance of the trees in $\mathcal{T}_S$ is at least $\lfloor d/2 \rfloor$.

Again, a special point p_0 is included in S with a far larger y-coordinate than any other point in S.

However, contrary to the first construction where the density of the blockers dropped by a factor of 1/2 in every step, we will now keep the density above 1/2 as long as possible by spreading in more points. For this purpose let $L_0 := \{(0,0),(1,0)\}$, and for $k \geq 1$ define

$$L_k := \left\{ (i/d^{k-1} + j/d^k, 0) \,\middle|\, \begin{array}{c} i = 0, \dots, d^{k-1}-1 \\ j = 1, \dots, d-1 \end{array} \right\},$$

i.e., $\bigcup_{0 \leq k' \leq k} L_{k'}$ subdivides the line segment from $(0,0)$ to $(1,0)$ into d^k equal parts by $d^k + 1$ points.

We define the point sets A_k inductively. Let

$$A_1 := L_0 \;\cup\; L_1 \oplus 1 \;\cup\; L_0 \oplus 2,$$

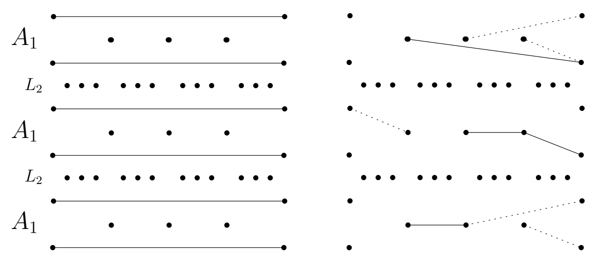
and for $k \in \{1, \dots, \lfloor d/2 \rfloor - 1\}$ and $h_k := 4 \cdot 3^{k-1} - 1$,

$$\begin{aligned} A_{k+1} \;\; := \;\; & A_k \;\cup\; L_{k+1} \oplus h_k \;\cup\; A_k \oplus (h_k + 1) \\ & \cup\; L_{k+1} \oplus (2h_k + 1) \;\cup\; A_k \oplus (2h_k + 2). \end{aligned}$$

Note that here A_{k+1} only uses three copies of the previously constructed A_k.

The point set A_1 and the horizontal edges between points with coordinates $(0,y)$ and $(1,y)$, for some $y \in \mathbb{N}$, form a $(d-1)$-of-d-blocker after 1 step. Applying Lemma 1 at this time gives that A_k together with the corresponding edges is a $(d-1)$-of-d^k-blocker after k steps. However, taking a closer look we can prove something stronger: recall that for the blocker A_1 at most one vertical strip of width $1/d$ is not blocked after 1 step. Placing three copies of A_1 on top of each other implies that after 1 step there cannot be more than one vertical strip of width $1/d$ that is not blocked at least twice. Hence, each of the $d-1$ vertical strips of width $1/d$ that are blocked twice, together with the points from L_2 in-between, behave like a (horizontally) scaled blocker A_1.

See for instance Figure 3 with the corresponding construction for $d = 4$. In Figure 3(b) only blocking edges (of the scaled blockers) are drawn as solid lines.



(a) The blocker A_2 and the horizontal edges, for $d = 4$

(b) Blocking edges (drawn as solid lines) after 1 step

Figure 3: A_2 is a 9-of-16-blocker after 2 steps.

Therefore, A_2 together with the horizontal edges is a $(d-1)^2$-of-d^2-blocker after 2 steps (since $d \geq 2$).

Inductively we find that A_k with the corresponding edges is a $(d-1)^k$-of-d^k-blocker after k steps as long as the density $(d-1)^{k-1}/d^{k-1}$ of the blocker A_{k-1} is at least 1/2 such that the three copies suffice to guarantee the existence of some blocked vertical strip.

As $d \geq 2$, this holds for $k = \lfloor d/2 \rfloor$. Thus, $A_{\lfloor d/2 \rfloor}$ is a blocker with density at least 1/2 after $\lfloor d/2 \rfloor$ rounds. With $S := L_{\lfloor d/2 \rfloor + 1} \cup A_{\lfloor d/2 \rfloor} \oplus 1 \cup \{p_0\}$ and T_1 and T_2 defined as in the first construction, the distance of the two trees in $\mathcal{T}_S$ is $\lfloor d/2 \rfloor + 2$. The size s_1 of A_1 is $d + 3$ and the size s_{k+1} of A_{k+1} can be bounded by the recursion $s_{k+1} < 3 \cdot s_k + 2 \cdot d^{k+1}$. For $d \geq 3$ we get by induction $s_k < 2kd^k$. This yields $|S| < d^{\lfloor d/2 \rfloor + 1} + 2\lfloor d/2 \rfloor d^{\lfloor d/2 \rfloor} + 1$ and hence $d \in \Omega\big(\log(|S|)/\log(\log(|S|))\big)$.

To express the diameter of the set S in terms of the number of convex layers we use the same argument as in Construction 1 but now count the rows instead of the columns. The number of rows is of order $3^{\lfloor d/2 \rfloor}$, thus the diameter is again logarithmic in the number of convex layers.

Theorem 3 *There exists a set S of n points in the plane for which the diameter of the graph $\mathcal{T}_S$ is in $\Omega\big(\log(n)/\log(\log(n))\big)$.*

We have the feeling that the $1/\log(\log(n))$ factor in the lower bound from Theorem 3 is more likely to be an artifact of our construction than the truth about the diameter of $\mathcal{T}_S$ which we think should be in $\Theta(\log(n))$, for a suited S with $n = |S|$.

References

[1] O. Aichholzer, F. Aurenhammer, C. Huemer, and H. Krasser. Transforming spanning trees and pseudo-triangulations. *Inf. Process. Lett. 97*, 1 (2006), 19–22.

[2] O. Aichholzer, F. Aurenhammer, and F. Hurtado. Sequences of spanning trees and a fixed tree theorem. *Comput. Geom. Theory Appl. 21*, 1 (2002), 3–20.

How Difficult is it to Walk the Dog?

Kevin Buchin[*] Maike Buchin[*] Christian Knauer[*] Günter Rote[*] Carola Wenk [†]

Abstract

We study the complexity of computing the Fréchet distance (also called dog-leash distance) between two polygonal curves with a total number of n vertices. For two polygonal curves in the plane we prove an $\Omega(n \log n)$ lower bound for the decision problem in the algebraic computation tree model allowing arithmetic operations and tests. Up to now only a $O(n^2)$ upper bound for the decision problem was known.

The $\Omega(n \log n)$ lower bound extends to variants of the Fréchet distance such as the weak as well as the discrete Fréchet distance. For the one-dimensional case we give a linear-time algorithm to solve the decision problem for the weak Fréchet distance between one-dimensional polygonal curves.

1 Introduction

The Fréchet distance is a metric for comparing parameterized shapes. In this paper we consider the Fréchet distance between polygonal curves. We also study variants of the Fréchet distance, namely the weak Fréchet distance, the discrete Fréchet distance, and the weak discrete Fréchet distance. There is a quadratic upper bound for solving the decision problem for the Fréchet distance between polygonal curves [2] and its variants, but so far no non-trivial lower bound was known.

In this paper we prove the following lower bound:

Theorem 1 *Determining whether or not the Fréchet distance between two polygonal curves in the plane of total complexity (i.e., number of vertices) n is less than a value ε takes $\Omega(n \log n)$ time in the algebraic computation tree model allowing arithmetic operations $(+, -, \times, /)$ and tests $(>, \geq, =)$.*

The same holds for the weak Fréchet distance with and without the restriction that endpoints are mapped to endpoints, the discrete Fréchet distance, and the weak, discrete Fréchet distance with and without endpoint restriction.

We prove Theorem 1 by reducing a problem with an $\Omega(n \log n)$ lower bound in linear time to the decision problem for the Fréchet distance. The problem we reduce from is *set inclusion* for which the lower bound in the above model has been proved by Ben-Or [3].

The lower bound in the theorem holds for polygonal curves in the plane. For the one-dimensional case we show that the lower bound for the weak Fréchet distance between one-dimensional polygonal curves does not hold. We give a linear-time algorithm for this case.

Note that the definition of the *weak Fréchet distance* does not require endpoints to be mapped to endpoints as does the definition of the *non-monotone Fréchet distance* in [2] which coincides with the *weak Fréchet distance with endpoint restriction.*

Theorem 2 *The weak Fréchet distance between one-dimensional polygonal curves can be computed in linear time.*

For the weak Fréchet distance with endpoint restriction Theorem 2 holds if the polygonal curves lie between their endpoints. It remains open whether the lower bound holds if the endpoints lie inside and whether the lower bound holds for the Fréchet distance between one-dimensional curves.

2 Fréchet Distance

In this section we recall the definitions of the Fréchet distance and its variants. For two parameterized curves $f_1, f_2 : [0,1] \to \mathbb{R}^d$ their Fréchet distance is defined as

$$\inf_{\substack{\alpha:[0,1]\to[0,1] \\ \beta:[0,1]\to[0,1]}} \max_{t\in[0,1]} |f_1(\alpha(t)) - f_2(\beta(t))|$$

where $|\cdot|$ denotes the Euclidean metric in $\mathbb{R}^d$ and the reparametrizations α, β range over all orientation-preserving homeomorphisms.

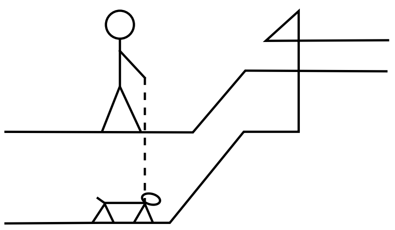


Figure 1: Fréchet Distance: length of shortest leash.

[*]Institute of Computer Science, Freie Universität Berlin, Berlin {buchin, mbuchin, knauer, rote}@inf.fu-berlin.de

[†]Computer Science Department, University of Texas at San Antonio, carola@cs.utsa.edu

The Fréchet distance can be illustrated by a man and a dog walking on the two curves as in Figure 1. The man has the dog on a leash. Both may choose their speed and may stop but not walk backwards. Then the Fréchet distance corresponds to the length of the shortest leash that allows them to walk on their respective curves from beginning to end. The Fréchet distance is therefore also called the *dog-leash distance*.

We focus on polygonal curves. Since the Fréchet distance is invariant under reparametrization we can assume a polygonal curve P to be given by the ordered list of its vertices, i.e., $P = (p_1, \dots, p_l)$.

The *weak Fréchet distance* between polygonal curves is defined in the same way except that the reparametrizations α, β range over all surjective continuous functions. For the *weak Fréchet distance with endpoint restriction* the reparametrizations are further required to map 0 to 0 and 1 to 1, respectively. In the man-dog illustration the weak Fréchet distance with endpoint restriction allows the man and dog to walk also backwards. In the weak Fréchet distance they may also choose their starting and ending point, but must cover both curves.

Since the weak versions of the Fréchet distance are defined as the Fréchet distance but with less constraints, the weak Fréchet distance with endpoint restriction is less or equal to the Fréchet distance, and the weak Fréchet distance is less or equal to the weak Fréchet distance with endpoint restriction.

The *discrete Fréchet distance* is defined using discrete maps on the vertices instead of homeomorphisms on the parameter spaces. Let $P = (p_1, \dots, p_l)$ and $Q = (q_1, \dots, q_m)$ be two polygonal curves given by their ordered lists of vertices. A *coupling* of the vertices is an ordered sequence of pairs of vertices in P, Q, i.e., $C = (c_1, \dots, c_k)$ with

$$c_r = (a, b),\ a \in P,\ b \in Q \quad \text{for } 1 \le r \le k,$$

fulfilling $(0,0)$, $(l,m) \in C$ and for $1 \le r < k$

$$\begin{aligned} & c_r = (a_i, b_j) \ \Rightarrow \\ & c_{r+1} \in \big\{(a_i+1, b_j),\ (a_i, b_j+1),\ (a_i+1, b_j+1)\big\}. \end{aligned}$$

The discrete Fréchet distance between polygonal curves is defined by taking the minimum over all couplings and the maximum over all distances between coupled vertices, i.e.,

$$\min_{C \text{ coupling}} \ \max_{(a_i, b_j) \in C} |a_i - b_j|.$$

A coupling of the vertices can be extended to a limit of homeomorphisms on the parameter spaces of the curves. This implies that the Fréchet distance is less than or equal to the discrete Fréchet distance. Furthermore, for any homeomorphism there exists a coupling which yields a distance that is not more than the distance of the homeomorphism plus half the length of the longest edge of either curve. Thus, if we add vertices to the curves P, Q so that their edge lengths tend to zero, their discrete Fréchet distance will tend to the Fréchet distance. The weak versions of the discrete Fréchet distance are defined analogously to the continuous case, i.e., a coupling can also make backward steps in the sense that from a_i it can step to $a_i - 1, a_i$, or $a_i + 1$.

3 Lower Bound

We reduce the problem of set inclusion to the decision problem for the Fréchet Distance.

Fréchet Distance Given two polygonal curves in $\mathbb{R}^d$ with vertices $P = (p_1, \dots, p_l)$, $Q = (q_1, \dots, q_m)$, $l + m \le n$ and $\varepsilon > 0$, determine whether or not the Fréchet distance between the curves is less than ε.

Set Inclusion Given two sets $A = a_1, \dots, a_n \subset \mathbb{R}$, $B = b_1, \dots, b_n \subset \mathbb{R}$, determine whether or not $A \subseteq B$.

In terms of distance measures the problem of set inclusion corresponds to deciding whether the directed Hausdorff distance between the point sets is 0, i.e., deciding whether

$$\max_{a \in A} \min_{b \in B} |a - b| = 0.$$

Given sets A and B for which we want to determine whether or not $A \subseteq B$, we first scale A and B such that $A \cup B \subset [0,1]$ holds. This can be done in linear time. In the following we assume $A \cup B \subset [0,1]$. For $a_i \in A$ we define

$$p_i := \big(2a_i/(1+a_i^2),\ (1-a_i^2)/(1+a_i^2)\big) \in \mathbb{R}^2$$

and for $b_i \in B$ we define

$$q_i := \big(-2b_i/(1+b_i^2),\ -(1-b_i^2)/(1+b_i^2)\big) \in \mathbb{R}^2.$$

The coordinates of all p_i and q_i can be determined in linear time in total. We define $p_0 := (1,1)$ and $q_0 := (0,0)$. Let C_A be the polygonal curve with vertices $(p_0, p_1, p_0, p_2, \dots, p_0, p_n, p_0)$ and C_B be the polygonal curve with vertices $(q_0, q_1, q_2, \dots, q_n, q_0)$. The construction is illustrated in Figure 2.

Theorem 1 directly follows from the following lemma:

Lemma 3 *Let the curves C_A, C_B be constructed as above from two finite sets $A, B \subset \mathbb{R}$. Then $A \not\subseteq B$ holds if and only if the Fréchet distance between C_A and C_B is less than 2.*

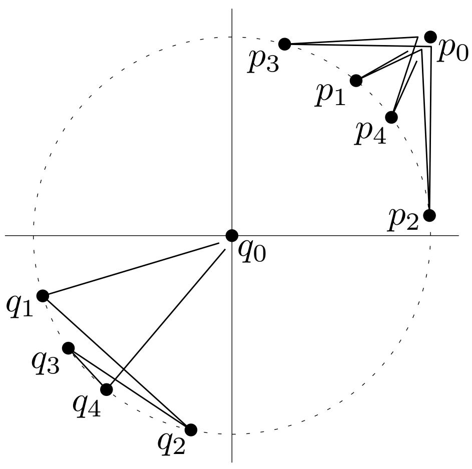


Figure 2: Polygonal curves C_A and C_B. The curves go through p_0 and q_0 but are drawn slightly perturbed for illustration purposes.

The same holds for the discrete Fréchet distance, the weak Fréchet distance with and without endpoint restriction, and the weak variants of the discrete Fréchet distance.

Proof. We prove the lemma first for the Fréchet distance, and then generalize it to the weak and discrete variants of the Fréchet distance. An important property of our construction is that the Euclidean distance between p_i and q_j equals 2 if and only if $a_i = b_j$, otherwise it is strictly less than 2.

Now assume $A \not\subseteq B$, i.e., there is an $a_k \notin \{b_1, \ldots, b_n\}$. Consider the following parametrizations of C_A and C_B: First traverse C_A until p_k is reached. So far the distance between pairs of points on the two curves is clearly less than 2 (actually at most $\sqrt{2}$ since on C_B we stay in q_0). Then C_B is traversed completely. Since no b_i equals a_k, all pairwise distances are less than 2. Now the rest of C_A is traversed but since on C_B we are again in q_0 the distance stays less than 2. In total these parametrizations yield a distance less than 2, therefore the Fréchet distance is less than 2.

For the other direction assume the Fréchet distance between the two curves is less than 2. Then there are parametrizations yielding a distance less than 2. Consider such parametrizations. At the point when the parametrization of C_B reaches q_1, the parametrization of C_A must be in the *neighborhood* of some p_k. The *neighborhood* of p_k is the subcurve of C_A with the vertices p_0, p_k, p_0 excluding the two endpoints p_0. Now, until the parametrization of C_B reaches q_n, the parametrization of C_A cannot leave the neighborhood of p_k because the closest possible point on C_B to p_0 is the point $(-1/2, -1/2)$, which still has distance $3/2 \cdot \sqrt{2} > 2$ to p_0.

It follows that all points in the neighborhood of p_k have distance less than 2 to $q_1, \ldots, q_n$. Since p_k is the closest point in its neighborhood to all of the $q_i, 1 \leq i \leq n$, the distance from p_k to all of them is less than 2. From this we get that $a_k \neq b_i$ for all $1 \leq i \leq n$, thus $A \not\subseteq B$.

This proves the lemma for the Fréchet distance. Since we did not use the monotonicity of the parametrizations the proof directly transfers to the weak Fréchet distance with and without endpoint restriction.

For the discrete Fréchet distance, consider again the two directions of the proof. For $A \not\subseteq B$ we constructed parametrizations realizing a Fréchet distance less than two. But these parametrizations also give a discrete Fréchet distance less than two since they always map vertices to vertices. For the other direction, we need to show that a discrete Fréchet distance less than two implies that $A \not\subseteq B$. This is equivalent to showing that $A \subseteq B$ implies a discrete Fréchet distance greater than or equal to two. This follows from the fact that the discrete Fréchet distance is always greater than or equal to the Fréchet distance. Combining the arguments for the weak Fréchet distance and the discrete Fréchet distance yields the result for the weak discrete Fréchet distance with and without endpoint restriction. □

4 Curves on a Line

In the previous section we showed an $\Omega(n \log n)$ lower bound for the decision problem for various variants of the Fréchet distance between polygonal curves in 2D. A natural question is whether these bounds still hold in 1D, i.e., in the case that the curves are restricted to lie on a line.

For the weak Fréchet distance we show that the lower bound does not hold. We show instead (Proposition 5) that the weak Fréchet distance can be computed in linear time by simply considering the differences of the extremal vertices. If the extremal vertices are the endpoints of the curves then this equals also the weak Fréchet distance with endpoint restriction.

Thus, the distance between polygonal curves is simpler to compute if we weaken the constraints of the Fréchet distance and restrict the dimension of the curves. Interestingly, there are similar results for the Fréchet distance between surfaces. While computing the Fréchet distance between simplicial surfaces is NP-hard [5], the weak Fréchet distance between simplicial surfaces in 3D can be computed in polynomial time [1]. If the surfaces are restricted to lie in a plane and to not self-intersect, i.e., to be simple polygons, then even the Fréchet distance can be computed in polynomial time [4].

The weak Fréchet distance between curves in 1D is closely related to the *Mountain Climbing Problem.*

The Mountain Climbing Problem
Two climbers start at sea-level on opposite sides of a mountain range and want to meet at the highest peak without resting on the way. Can they travel in a way that they stay on equal altitude at all times?

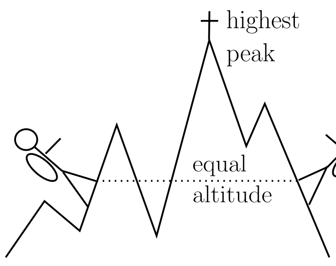


Figure 3: Mountain Climbing.

This problem is illustrated in Figure 3. It has been answered many times (see [8]). Homma [6] proved in 1952 that the climbers can stay at equal altitude if the mountains are locally non-constant. He also gave an example where it is not possible for the climbers, where one mountain range has a plateau while the other mountain range oscillates infinitely often.

In mathematical terms the problem asks for a characterization of the continuous functions $f_1, f_2 : [0,1] \to [0,1]$ with $0 = f_1(0) = f_2(0)$ and $1 = f_1(1) = f_2(1)$ for which there are continuous functions $g_1, g_2 : [0,1] \to [0,1]$ with $0 = g_1(0) = g_2(0)$ and $1 = g_1(1) = g_2(1)$ and

$$f_1 \circ g_1 = f_2 \circ g_2.$$

Weak Fréchet Distance in 1D The mountain climbing problem can be interpreted in terms of the weak Fréchet distance: Are there reparametrizations of f_1 and f_2 mapping endpoints to endpoints that realize a distance of 0?

The answer above directly yields that the weak Fréchet distance with endpoint restriction is 0 for curves f_1 and f_2 which take values in $[0,1]$, are locally non-constant, and satisfy $0 = f_1(0) = f_2(0)$ and $1 = f_1(1) = f_2(1)$. For curves $f_i,\ i = 1,2$, which take values in $[a_i, b_i]$, are locally non-constant, and satisfy $a_i = f_i(0)$ and $b_i = f_i(1),\ i = 1,2$, this implies that the weak Fréchet distance with endpoint restriction of f_1 and f_2 is $\max(|a_2 - a_1|, |b_2 - b_1|)$.

A characterization of the functions f_1, f_2 [7] for which such reparametrizations exist implies that the weak Fréchet distance between continuous functions with the same image is 0 even in the general case, i.e., where f_1, f_2 may be locally constant. In this case the "climbers" can maintain almost the same altitude.

Corollary 4 (Huneke [7]) *For any two continuous, surjective functions $f_1, f_2 : [0,1] \to [0,1]$ and for any $\varepsilon > 0$, there exist continuous, surjective functions $g_1, g_2 : [0,1] \to [0,1]$ such that for all $x \in [0,1]$*

$$|f_1 \circ g_1(x) - f_2 \circ g_2(x)| < \varepsilon.$$

Note that f_1 and f_2 no longer need to start at 0 and end at 1. For the weak Fréchet distance this implies:

Proposition 5 *Let $f_1, f_2 : [0,1] \to \mathbb{R}$ be continuous functions with $f_i([0,1]) = [a_i, b_i]$ for $i = 1,2$. The weak Fréchet distance between f_1 and f_2 is*

$$\max(|a_2 - a_1|, |b_2 - b_1|).$$

Theorem 2 directly follows from Proposition 5. If a_1, a_2, b_1, and b_2 are known, the weak Fréchet distance can even be computed in constant time.

5 Discussion

We presented an $\Omega(n \log n)$ lower bound for the decision problem for the Fréchet distance between polygonal curves in the plane. An open problem is to close the gap to the known quadratic upper bound. Furthermore, it is open whether the lower bound holds for underlying metrics other than the Euclidean metric.

We showed that the lower bound does not hold for the weak Fréchet distance between curves on a line. It remains to investigate the complexity of the Fréchet distance for curves on a line for which we only know the quadratic upper and trivial linear lower bound.

Acknowledgment

We would like to thank Helmut Alt and Rom Pinchasi for helpful discussions.

References

[1] H. Alt and M. Buchin. Can we compute the similarity between surfaces? In preparation.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[3] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[4] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons in polynomial time. In *Proc. 22nd Annu. ACM Sympos. Comput. Geom.*, pages 80–87, 2006.

[5] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions.* PhD thesis, Freie Universität Berlin, Germany, 1998.

[6] T. Homma. A theorem on continuous functions. *Kodai Math. Semin. Rep.*, 4(1):13–16, 1952.

[7] J. P. Huneke. Mountain climbing. *Trans. Am. Math. Soc.*, 139:383–391, 1969.

[8] V. Jiménez López. An elementary solution to the mountain climbers' problem. *Aequationes Math.*, 57(1):45–49, 1999.

Small Manhattan Networks and Algorithmic Applications for the Earth Mover's Distance

Joachim Gudmundsson* Oliver Klein† Christian Knauer† Michiel Smid‡

Abstract

Given a set S of n points in the plane, a Manhattan network on S is a (not necessarily planar) rectilinear network G with the property that for every pair of points in S the network G contains a path between them whose length is equal to the Manhattan distance between the points. A Manhattan network on S can be thought of as a graph $G = (V, E)$ where the vertex set V corresponds to the points of S and a set of Steiner points S'. The edges in E correspond to horizontal and vertical line segments connecting points in $S \cup S'$. A Manhattan network can also be thought of as a 1-spanner (for the L_1-metric) for the points in S.

We will show that there is a Manhattan network on S with $O(n \log n)$ vertices and edges which can be constructed in $O(n \log n)$ time. This allows us to to compute the L_1-Earth Mover's Distance on weighted planar point sets in $O(n^2 \log^3 n)$ time, which improves the currently best known result of $O(n^4 \log n)$. At the expense of a slightly higher time and space complexity we are able to extend our approach to any dimension $d \geq 3$. We will further show that our construction is optimal in the sense that there are point sets in the plane where every Manhattan network needs $\Omega(n \log n)$ vertices and edges.

1 Introduction

The problem to compute a minimum length Manhattan network is a well-researched area, see Gudmundsson et al. [7], Benkert et al. [1] and Chepoi et al. [3]. Even though the problem has received considerable attention the variant of minimizing the number of vertices and edges of the graph has not been considered (to the best of the authors' knowledge).

Here we will show that for every point set in the plane there is a Manhattan network with $O(n \log n)$ vertices and edges. This graph can be constructed in $O(n \log n)$ time. We will also show that this upper bound is tight, meaning that there are point sets where every Manhattan network on these points will need at least $\Omega(n \log n)$ vertices and edges. As it turns out, the Manhattan network constructed is not planar. We will show that if we force the network to be planar, there are point sets where every Manhattan network needs at least $\Omega(n^2)$ vertices and edges. Further we will show how to generalize the construction of the network to higher dimensions. Finally, we will show that one can reduce the time to compute the L_1-Earth Mover's Distance (EMD) for weighted point sets. The EMD is a useful distance measure for, e.g., shape matching, color-based image retrieval and music score matching, see Cohen and Guibas [4], Giannopoulos and Veltkamp [5], Graumann and Darell [6], and Typke, Giannopoulos, Veltkamp, Wiering and van Oostrum [11] for more information. Work on the optimization problem for the EMD under transformations has been done by Cabelloet al. [2] and Klein and Veltkamp [9]. An upper bound for the time to compute the EMD is $O(n^4 \log n)$ using a strongly polynomial minimum cost flow algorithm by Orlin [10]. Cabello et al. [2] gave a $(1 + \varepsilon)$-approximation algorithm with runtime $O(n^2 \varepsilon^{-2} \log^2(n\varepsilon^{-1}))$. Recently, Indyk [8] gave an $O(n \log^{O(1)} n)$-time randomized $O(1)$-approximation algorithm if the two point sets consist of an equal number of points in $\mathbb{R}^2$ of weight 1. Using the Manhattan network as a 1-spanner for the L_1-distance, we can compute the L_1-EMD in d dimensions in $O(n^2 \log^{2d-1} n)$ time using Orlin's algorithm on the reduced graph. This improves the previously best known runtime of $O(n^4 \log n)$ significantly. Further, it immediately leads to a $\sqrt{2}$-approximation with the same runtime for the important case when the EMD is based on the Euclidean distance. This algorithm is conceptually easier than the slightly faster $(1 + \varepsilon)$-approximation given by Cabello et al. [2].

2 Manhattan Networks

We will start formulating and proving the main result of this paper.

*National ICT Australia Ltd, Sydney, Australia. NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. joachim.gudmundsson@nicta.com.au

†Institut für Informatik, Freie Universität Berlin. This research was supported by the Deutsche Forschungsgemeinschaft within the European graduate program 'Combinatorics, Geometry and Computation' (No. GRK 588/3), {oklein,christian}@inf.fu-berlin.de

‡This research was supported by NSERC. School of Computer Science, Carleton University, Ottawa. michiel@scs.carleton.ca

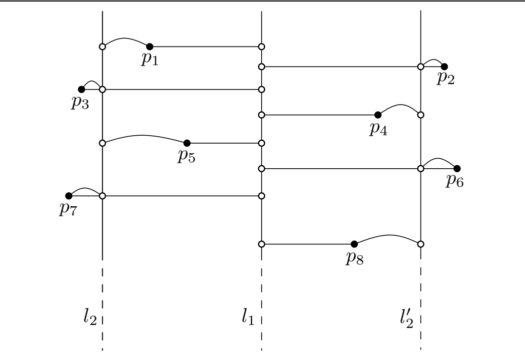


Figure 1: Construction of the Network.

Theorem 1 *Let S be a set of n points in the plane. Then, there is a Manhattan network on S with $O(n \log n)$ vertices and edges. It can be computed in $O(n \log n)$ time.*

Proof. Let S be a set of n points in the plane and assume that the points are sorted w.r.t. y-coordinates $S = \{p_1, \ldots, p_n\}$. Otherwise, the sorting can be done in $O(n \log n)$ time. In the following, L always denotes a list of points which is sorted by y-coordinate. The i-th point in L will be denoted by $L[i]$. Now, run the following routine on S.

```
Algorithm 1 (ConstructNetwork(L))
  1. Find median $p^*$ with respect to $x$-coordinate.
  2. Set $L_1 := \emptyset, L_2 := \emptyset$.
  3. For $i = 1, \ldots, |L|$ do
     (a) Construct vertex $v[i] := (p^*_x, L[i]_y)$.
     (b) Construct edge $e_h[i] := (L[i], v[i])$.
     (c) If $i \geq 2$:
         Construct edge $e_v[i] := (v[i-1], v[i])$.
     (d) If $L[i]_x \leq p^*_x$: add $L[i]$ at the end of $L_1$
         Else: add $L[i]$ at the end of $L_2$.
  4. ConstructNetwork($L_1$)
  5. ConstructNetwork($L_2$)
```

See Figure 1 for an illustration of the algorithm. We have to prove that the algorithm constructs a Manhattan network.

Let $p, q \in S$ be two arbitrary points. Let p^* be the first point chosen as a median in Step 1 with $p_x \leq p^*_x \leq q_x$. Clearly, p and q are both contained in L. W.l.o.g., let $p = L[i] =: p_i$ and $q = L[j] =: p_j$ with $i < j$. In Step 3, p_i is considered before p_j. Therefore, by construction, there are vertices $v[i], v[j]$, edges $(v[i], p_i)$, $(v[j], q_j)$ and a y-monotone sequence of vertices $v[i], \ldots, v[j]$. Now, the sequence $p_i, v[i], \ldots, v[j], p_j$ is an x- and y-monotone path consisting of two horizontal edges connected by a path of vertical edges and therefore a Manhattan path. This proves that the resulting graph is a Manhattan network on S.

The median in a list of $k := |L|$ numbers can be computed in $O(k)$. Steps (a) to (d) can be done in constant time. Therefore, the runtime of Algorithm 1 without the two recursive calls is $O(k)$. The insertion in the lists L_1, L_2 is done in sorted order with respect to the y-coordinate. No re-sorting is needed after the initial sorting step.

The overall runtime can be described by the recursion $T(n) = O(n) + 2 \cdot T(n/2)$, which gives $T(n) = O(n \log n)$. The number of Steiner points and edges in the construction obeys the same recursion, since in every recursive call of Algorithm 1 $O(k)$ vertices and edges are added. □

In practice, paths with a small number of links are often desirable. We will show how to construct a network such that for every pair of points there is a shortest path with a small number of links. Let $\alpha(n)$ denote the inverse of Ackermann's function, see [12].

Theorem 2 *Let S be a set of n points in the plane. Then, there is a Manhattan network on S with $O(n \log n)$ vertices and edges, where the number of edges on a shortest Manhattan path between any pair of points is bounded by $O(\alpha(n))$. The network can be computed in $O(n \log n)$ time.*

Proof. Consider one call of Algorithm 1. The Manhattan path between two input points p_i, p_j with $i < j$ always has the form $p_i, v[i], \ldots, v[j], p_j$, where $v[i], \ldots, v[j]$ is a sequence of Steiner points lying on a vertical line. Now, using a result of Yao [12], we can compute $O(k)$ edges in $O(k)$ time, each connecting two Steiner points, such that for any pair of Steiner points the number of links on the shortest path is $O(\alpha(k))$. That is, the total length of any path constructed is $O(\alpha(n))+2 = O(\alpha(n))$. Since we can compute these $O(k)$ edges in every recursive call in $O(k)$ time, the asymptotic runtime and number of Steiner points does not change. □

At the expense of a slightly higher runtime we can reduce the length of a shortest Manhattan path to a constant number of edges.

Theorem 3 *Let S be a set of n points in the plane. Then, there is a Manhattan network on S with $O(n \log^2 n)$, $O(n \log n \log \log n)$ and $O(n \log n \log^* n)$ vertices and edges where the number of edges on a shortest Manhattan path between any pair of points is bounded by $6, 7$ and 8, respectively. The runtimes are linear in the number of vertices and edges.*

Proof. The proof is analogous to that of Theorem 2, again using results of Yao [12]. □

The upper bound given in Theorem 1 is tight.

Theorem 4 *There are n-point sets in $\mathbb{R}^2$ where every Manhattan network needs $\Omega(n \log n)$ vertices and edges.*

Proof. We construct a point set P in general position, such that any Manhattan network for P consists of $\Omega(n \log n)$ vertices and edges. We assume that n is a power of two. Let ℓ be a vertical line separating P into two point sets $U := \{u_1, \ldots, u_{n/2}\}$ and $V := \{v_1, \ldots, v_{n/2}\}$, such that the points $u_1, v_1, u_2, v_2, \ldots, u_{n/2}, v_{n/2}$ are sorted by y-coordinates, from top to bottom, see Figure 2. For

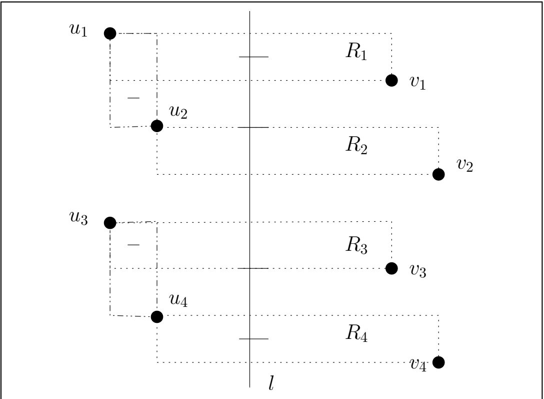


Figure 2: Lower Bound

$1 \leq i \leq n/2$, let R_i be the axes-parallel rectangle with top-left corner u_i and bottom-right corner v_i. Any Manhattan network on P must contain a path between u_i and v_i that crosses ℓ and is completely contained in R_i. Since the rectangles R_i are pairwise disjoint, it follows that any Manhattan network on P contains at least $n/2$ edges that cross ℓ. Observe that this remains true if we move the points of U and V horizontally, as long as U stays to the left of ℓ and V stays to the right of ℓ. Thus, we can move the points of U, such that they can be split into two subsets U_1 and U_2 that are separated by a vertical line ℓ' such that the sorted y-order alternates between a point in U_1 and a point in U_2. Any Manhattan network on P must contain at least $n/4$ edges that cross ℓ' and that are distinct from the above $n/2$ edges. Similarly, we can move the points of V, and split them into two subsets V_1 and V_2 that are separated by a vertical line ℓ'' in such a way that any Manhattan network on P must contain at least $n/4$ edges that cross ℓ'' and are distinct from the above $n/2 + n/4$ edges. We continue this moving in a recursive way and it can be shown that all these edges are distinct. We omit a rigorous proof due to space limitations. Then, it follows that the number $T(n)$ of vertices and edges in any Manhattan network on the final set P satisfies $T(n) \geq n/2 + 2 \cdot T(n/2)$, which proves that $T(n) = \Omega(n \log n)$. □

If the network is required to be planar the lower bound can be improved.

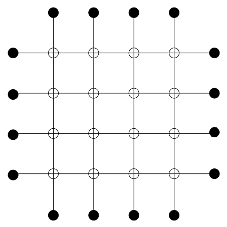


Figure 3: Planar Construction.

Theorem 5 *There are n-point sets in $\mathbb{R}^2$ where every planar Manhattan network needs $\Omega(n^2)$ vertices and edges.*

Proof. Let the set P of points in $\mathbb{R}^2$ be defined as follows, see Figure 3:

$$P := \bigcup_{i=1}^{n-1} (\{(\frac{i}{n}, 0)\} \cup \{(\frac{i}{n}, 1)\} \cup \{(0, \frac{i}{n})\} \cup \{(1, \frac{i}{n})\})$$

Let G be a Manhattan network for this point set. There must be a Manhattan path between every pair of points $(\frac{i}{n}, 0), (\frac{i}{n}, 1)$ and $(0, \frac{i}{n}), (1, \frac{i}{n})$. These paths have to be straight lines, since in the first case the x-coordinate and in the second case the y-coordinate is the same. This forces the $O(n^2)$ cross points of the straight lines to be Steiner points. □

A point set in general position giving the same lower bound can be constructed easily by perturbing the points slightly.

3 Higher Dimensions

The extension of the definition of a Manhattan path and therefore of a Manhattan network to dimensions $d \geq 3$ is straightforward. In dimension d we can use a similar divide-and-conquer approach as in the plane.

Theorem 6 *Let S be a set of n points in $\mathbb{R}^d$. Then, there is a Manhattan network on S with $O(n \log^{d-1} n)$ vertices and edges. It can be computed in $O(n \log^{d-1} n)$ time.*

Proof. Consider Algorithm 2 for point sets in $\mathbb{R}^d$:

```
Algorithm 2 (ConstructNetwork($L, d$))
  1. Find median $p^*$ with respect to the $d$-th coordinate.
  2. Project any point on the hyperplane containing $p^*$ and orthogonal to the $d$-th coordinate.
     Let $\mathcal{P}$ be the set of projected points.
  3. Add an edge between the original points and their projection.
  4. ConstructNetwork($\mathcal{P}, d-1$) (Compute the Manhattan network on this hyperplane).
  5. $L_1 := \{p \in L : p_d \leq p^*_d\}$. $L_2 := L \setminus L_1$.
  6. ConstructNetwork($L_1, d$)
  7. ConstructNetwork($L_2, d$)
```

Except for the recursive calls in the algorithm, any call can be done in $O(|L|)$ time. There are three recursive calls, one call of the routine for the same number of points in one dimension less and two calls for the number of points halved in the same dimension. Analogous to the earlier proof, the runtime of this can be expressed like

$$\begin{aligned} T(n,d) &= O(n) + T(n,d-1) + 2 \cdot T(n/2,d) \\ &= O(n \log^{d-1} n). \end{aligned}$$

The bound on the number of points and edges follows analogously. □

4 Earth Mover's Distance

We will now show that we can reduce the time to compute the L_1-Earth Mover's Distance on weighted point sets to $O(n^2 \log^{2d-1} n)$, which improves the previously best known result of $O(n^4 \log n)$.

A set $A = \{a_1, \ldots, a_n\}$ is called a weighted point set in $\mathbb{R}^d$ if $a_i = (p_i, \alpha_i)$ for $i = 1, \ldots, n$, where p_i is a point in $\mathbb{R}^d$ and $\alpha_i \in \mathbb{R}_0^+$ its corresponding weight; $W^A = \sum_{i=1}^n \alpha_i$ denotes the total weight of A. Now, let $A = \{(p_i, \alpha_i)_{i=1,\ldots,n}\}$ and $B = \{(q_j, \beta_j)_{j=1,\ldots,m}\}$ be two weighted point sets with total weights W^A, $W^B \in \mathbb{R}^+$ and $m \le n$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_0^+$ be a distance measure on $\mathbb{R}^d$. The D-EMD between A and B is defined as

$$D\text{-}EMD(A,B) = \frac{\min_{F \in \mathcal{F}} \sum_{i=1}^n \sum_{j=1}^m f_{ij} D(p_i, q_j)}{\min\{W^A, W^B\}},$$

where $F = \{f_{ij}\}$ is a feasible flow, i.e., for every $i = 1, \ldots, n$ and $j = 1, \ldots, m$ we have $f_{ij} \ge 0$, $\sum_{j=1}^m f_{ij} \le \alpha_i$, $\sum_{i=1}^n f_{ij} \le \beta_j$ and $\sum_{i=1}^n \sum_{j=1}^m f_{ij} = \min\{W^A, W^B\}$.

Theorem 7 *The L_1-EMD can be computed in $O(n^2 \log^{2d-1} n)$ time.*

Proof. Let A, B be weighted point sets. Using Theorem 6 we can construct a 1-spanner of the complete bipartite graph between the points of A and B for the L_1-metric in $O(n \log^{d-1} n)$ time. The number of points and edges in the resulting network is bounded by $O(n \log^{d-1} n)$. Now we proceed as in Cabello et al. [2]. By the standard method of doubling each edge and orienting the two copies in different directions we get a flow network where between any pair of points there is a directed path of minimum L_1-length. Now we can use the minimum cost flow algorithm by Orlin [10] on the 1-spanner. Given a network $G = (V, E)$, Orlin's algorithm solves the minimum cost flow problem in $O((|E| \log |V|)(|E| + \log |V|))$. Since the number of points and edges in our spanner is bounded by $|E| = |V| = O(n \log^{d-1} n)$, the overall runtime is bounded by $O(n^2 \log^{2d-1} n)$. □

Theorem 7 immediately leads to a $\sqrt{2}$-approximation with the same runtime for the important case when the EMD is based on the Euclidean distance. This algorithm is conceptually easier than the slightly faster $(1+\varepsilon)$-approximation given by Cabello et al. [2].

Acknowledgments

We thank Günter Rote for the idea in Theorem 5.

References

[1] M. Benkert, A. Wolff, F. Widmann, and T. Shirabe. The minimum Manhattan network problem: Approximations and exact solution. *Computational Geometry - Theory and Applications*, 2006.

[2] S. Cabello, P. Giannopoulos, C. Knauer, and G. Rote. Matching point sets with respect to the earth mover's distance. In *Proc. 13th ESA*, 2005.

[3] V. Chepoi, K. Nouioua, and Y. Vaxes. A rounding algorithm for approximating minimum manhattan networks. In *Proc. 8th APPROX*, 2005.

[4] S. D. Cohen and L. J. Guibas. The earth mover's distance under transformation sets. In *Proc. 7th IEEE Int. Conf. Comp. Vision*, 1999.

[5] P. Giannopoulos and R. C. Veltkamp. A pseudo-metric for weighted point sets. In *Proc. 7th Europ. Conf. on Comp. Vision*, 2002.

[6] K. Graumann and T. Darell. Fast contour matching using approximate earth mover's distance. In *Proc. 1991 IEEE Comp. Society Conf. on Comp. Vision and Pattern Recognition*, 2004.

[7] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Approximating a minimum Manhattan network. *Nordic J. Comput. 8*, 2001.

[8] P. Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost), to appear. In *Proc. 18th Symp. on Disc. Alg.*, 2007.

[9] O. Klein and R. C. Veltkamp. Approximation algorithms for the earth mover's distance under transformations using reference points. In *Proc. 16th ISAAC*, 2005.

[10] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research 41*, 1993.

[11] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. Oostrum. Using transportation distances for measuring melodic similarity. In *Proc. 4th Int. Conf. on Music Information Retrieval*, 2003.

[12] A. C. Yao. Space-time trade-off for answering range queries. In *Proc. 14th STOC*, 1982.

Applying graphics hardware to achieve extremely fast geometric pattern matching in low dimensional transformation space *

Dror Aiger† Klara Kedem‡

Abstract

We present a GPU-based approach to geometric pattern matching. We reduce this problem to finding the depth (maximally covered point) of an arrangement of polytopes in transformation space and describe hardware assisted (GPU) algorithms, which exploit the available set of graphics operations to perform a fast rasterized depth computation.

1 Introduction

A central problem in pattern recognition, computer vision, and robotics, is the question of whether two point sets P and Q *resemble* each other. One approach to this problem, first used by Huttenlocher et al.[8, 2] is based on the minimum Hausdorff distance between point sets in the plane under translation.

The Hausdorff distance between two point sets P and Q is defined as $H(P,Q) = \max(h(P,Q), h(Q,P))$ where $h(P,Q) = \max_{p \in P} \min_{q \in Q} d(p,q)$ and $d(\cdot,\cdot)$ is a standard metric on points.

In this paper we consider the following geometric pattern matching problem: Given two point sets P and Q in the plane, and some $\delta > 0$, find a 3-parameters transformation $T \in G$ that brings the largest subset S of P to distance $h(T(S), Q) \leq \delta$, where $h(.,.)$ is the directional Hausdorff distance with L_∞ as the underlying metric. We describe two alternatives, in one G is the set of translation + scale transformations, and in the other the set of rigid transformations. We reduce this problem to finding the depth (maximally covered point) of an arrangement of polytopes in transformation space. This reduction is quite common (see, e.g. [7, 9]).

In [3] we gave a randomized algorithm for approximating the *Pattern Matching* problem for point sets under similarity transformation. We refer the reader to that paper for description of related problems and previous work. The geometric pattern matching problem we solve in this paper is also termed the *Largest Common Pointset (LCP)* problem.

Based on the reduction to depth in a polytopes arrangement, we show that the problem can be solved very fast using a modern standard graphics hardware. Though translation + scale is a linear transformation rigid transformation (rotation and translation) is not. We will discuss approximating the regions in transformation space by a union of convex polytopes with linear boundaries. We use the so called *depth peeling* algorithm[1] implemented on the GPU to get the k-levels one after another while maintaining some structures. This enables us to get the deepest point in the arrangement (with a bounded error which depends on the rasterization) in $O(L)$ passes over the data where L is the number of levels in the arrangement.

2 The GPU as a stream computer

Recently, many GPU-based algorithms for geometry, image processing and other problems have been considered by researchers (see, e.g. [6, 4]). In particular, the GPU as a stream computer for geometric optimization was considered by [5]. In many applications, performing a computation on the graphics card is far faster than performing it on the CPU. The GPU provides spatial parallelism where each pixel on the screen can be viewed as a stream processor, enabling an application to be computed in highly parallel mode.

3 The Largest Common Point set problem (LCP) and its reduction to depth in an arrangement

We reduce the LCP problem to depth in arrangement as follows: All the transformations that bring a point p in P up to L_∞ distance δ from q in Q correspond to a region in transformation space [3]. The region in transformation space is the intersection of four constraints defined by the the square of size 2δ around q (see Figure 1). For linear transformations, one such region forms a convex polytope in transformation space where each polytope is the Minkowski sum of a $3D$ line by a planar square of side size 2δ. The $3D$ line in transformation space corresponds to all transformations that bring a point in P exactly to a point in Q and its Minkowski sum corresponds to all transformations that bring a point p in P to a

*This work was partly supported by the MAGNET program of the Israel Ministry of Industry and Trade (IMG4 consortium)

†Department of Computer Science, Ben Gurion University, Be'er Sheva, Israel and Orbotech LTD aiger@cs.bgu.ac.il,dror-ai@orbotech.com

‡Department of Computer Science, Ben Gurion University and Cornell University, klara@cs.bgu.ac.il

point q in Q such that $h(T(p), q) \leq \delta$ using the L_∞ metric. For rigid transformation, this region forms a Minkowski sum of an arc with the same square, since the transformation is not linear in the angle parameter. In this case we approximate the region with the union of convex polytopes. The solution is then approximated in the sense that the distance of the solution from the true optimal solution is bounded by the maximum error of our approximation (see Section 4.3).

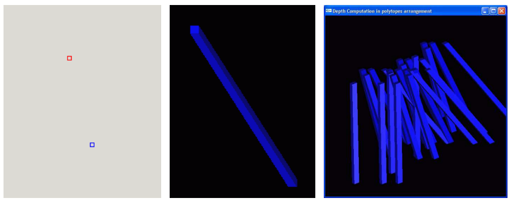

(a) (b) (c)

Figure 1: Polytopes in transformation space: (a) p in P and q in Q, (b) polytope created for a single match (c) arrangement of polytopes for a point p and all points of Q.

For each pair (p, q), $p \in P$ and $q \in Q$, there is a polytope. We assume that the polytopes that correspond to one point p in P are disjoint in their interior (if this is not the case, we can decompose the union of squares of size 2δ around points in Q to $O(|Q|)$ squares disjoint in their interiors as in [7]). We investigate the depth of the polytope arrangement: the depth of a point t in transformation space is the number of polytopes that cover t. A point of maximum depth corresponds to a transformation that brings the maximum number of points in P close to Q [7]. The regions in transformation space that are covered by $|P|$ of these $|P||Q|$ polytopes are thus the locus of transformations that bring all points of P close enough to a point of Q. Since we are looking for matching the largest subset S of P to points of Q, we will search for the maximal depth of that arrangement.

4 Computing the maximum depth

4.1 Using k-levels

In this section we show how we compute the maximum depth in $3D$ transformation space using the GPU.

Definition 1 *Given a set C of hyperplanes, a point p is said to be at k-level, if there are exactly k hyperplanes in C lying strictly below p.*

In an arrangement of convex polytopes, the depth is the number of objects that a ray down to $-\infty$ crosses only once (see Figure 2). The depth can be computed by going through the k-levels, one by one and counting the number of “entering into” and “exiting” polytope events. If we count and remember the maximum, we end up with the point of maximum depth.

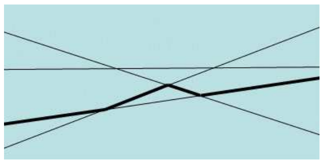


Figure 2: k-levels in arrangement of lines. The bold line is the 1-level.

4.2 Applying the GPU in the computation of k-levels

We use the GPU to compute a rasterized version of the k-levels for $3D$ transformation space. (The rasterization induces an error bounded by the pixel size.) Simple rendering using the depth buffer computes the 0-level using the standard z-buffer. The z-buffer performs the test “is $x < a$” in its standard process for hidden surface removal[10] while surfaces or planes are being rendered.

Our goal is to “peel” the k-levels one by one when our scene is built from all the polytopes defined above. During this process we count for every pixel all the objects that cover it by counting entry into and exit of polytopes. Using two depth buffers simultaneously plus the previous level, we can perform the test “is $a < x < b$” and thus we get the 1-level for all the pixels simultaneously. Similarly we follow to all k-levels.

This process is known as *depth peeling* and uses the shadow map as a secondary depth buffer (see [1] for the detailed algorithm). Throughout the process we count the number of entries and exits (from polytopes) for each pixel, thus getting the maximum depth in the polytope arrangement. By applying fragment program (FPs) [10] we compute the maximum depth at each pixel. The entire process requires L passes over the input scene when L is the number of levels and no CPU involvement is needed during the process.

Once we finish, we have the translation (t_x, t_y) at which the maximum depth is attained but we still have to compute the third parameter of the transformation (scale, or rotation in the rigid case).

For finding the right scale efficiently we have to find the scale coordinate for (t_x, t_y). We perform another pass similar to the one above but now only on the pixel (t_x, t_y) applying *selection mode* on that pixel. Selection mode is a mode of rendering in which the depth order of the objects within a given window is saved during the rendering and can be obtained by the application afterwards. The output of the *selection mode* are the ordered faces in the arrangement that meet the ray from (t_x, t_y) to $-\infty$. The scale coordinate of maximum depth in the arrangement is thus found.

4.3 Computing the depth for rigid transformations

The only difference between rigid transformations and scale + rotation transformations is that the rigid is not a linear transformation. Thus a linear constraint of the rectangle of size 2δ around points of Q in the plane does no longer correspond to a hyperplane in transformation space. The transformations that bring a point $p \in P$ to a point $q \in Q$ form an arc in $3D$ instead of a line as before. For a given error bound, we can approximate this arc by a set of segments.

By evaluating the allowed error we divide our rotation axis into a set of slices such that in any slice the arc is approximated by a line segment. The Minkowski sums of the polygonal lines approximating the arcs with a square of size 2δ yield the polytopes on which we apply the method described in the previous subsection.

4.4 Speeding up by randomization and oriented points

For simplicity we assume $|P| = |Q| = n$. The complexity of the peeling algorithm is $O(L)$ rendering passes over the data which contains n^2 polytopes where L is the number of levels. The number of levels is view dependent but can be $O(n^2)$. Thus we have runtime complexity of $O(n^4)$ (it is typically close to $O(n^3)$ in practical cases since $L = O(n)$ in most of the cases). A significant speed up can be achieved by applying randomization and oriented points, combined with stencil buffer and occlusion query. (The stencil buffer is used to mask a specific region in the frame buffer to which we want to restrict the rendering. Occlusion query is an Open GL operation [10] that enables the application to count the pixels in the frame buffer which have been updated during a specific rendering process.)

We decompose the problem to a set of smaller problems and use the GPU to quickly reject polytopes that cannot contribute to the optimal solution. Since our polytopes are the Minkowski sum of a $3D$ line (in the scale + translation case) and a planar square of side size 2δ, the polytopes are actually `sticks` and the intersection of a pair of them is expected to be a small region in $3D$ (especially if the angle between the $3D$ lines is large enough). To decompose the problem to smaller problems we use randomization as follows: Given $N = n \cdot n$ polytopes (each $p \in P$ transformed to a square around each point $q \in Q$), we know that the depth is bounded by n. If we assume that a constant fraction of the points of P are matched to points in Q, the probability that the intersection of two random polytopes contains the desired solution is $constant/N$. We sample $O(N \log N)$ pairs of polytopes to get high probability that the intersection of one pair contains the desired solution. We now have $O(N \log N)$ small regions in transformation space. For each region we mask the intersected region on the stencil, rendering all polytopes and using the occlusion query to reject polytopes that do not intersect these regions (their number is expected to be large).

The number of polytopes intersecting the regions is typically $O(n)$ instead of the original size $N = n \cdot n$, thus we need only $O(n)$ passes over $O(n)$ polytopes. In the worst case, we can still get a high number of polytopes if there is a large number of optimal solutions but this is rare in practice.

We can further speed up the process by using oriented points. In many real life applications we get orientation at each point (e.g. from edge detection). We construct only polytopes that correspond to validly matching orientations of points $p \in P$ and $q \in Q$ (when the difference between orientations after transformation, up to a threshold). The number of polytopes is thus reduced dramatically in the translation + scale problem. For rigid transformations, the size of the polytopes is reduced dramatically.

5 Experiments and results

The tests we present here are all under translation and scale. Rigid transformation is essentially the same and was not implemented. We implement the peeling algorithm with randomization and orientations on points. We used a PC running at 3Ghz with Windows XP and OpenGL. The GPU is the Nvidia GEforce 6600. Below we show a synthetic example where we graph the runtime as a function of pattern points and a real time object recognition example that demonstrates the power of the GPU in real life application.

5.1 Synthetic data

For the synthetic test, we randomly create 1000 oriented points in the range $[-1, 1]$ (the orientations were randomly selected from the range $[0,2\pi]$) as the set Q. For the set P we randomly select a subset of points of Q and perturb each point with a uniformly distribution in a small neighborhood of the selected point. For the square size around each point of Q we picked $\delta = 0.004$. For a pair (p, q) we create a polytope if the difference of orientations of p and q is up to 5 degrees. The raster resolution (pixel size) is 0.002. The scale was bounded to be in the range $[0.5, 2.0]$. We note that for the GPU implementation the runtime is almost not affected by the size of the search space.

The error on the various axes differs as in translation it is determined by pixel size and in scale axis it depends on the number of bits in each pixel in the depth buffer which is 32 bits in our implementation, and thus pretty precise.

The running times using this data are shown in Figure 3, where we picked P to be of 50, 100, 200 and 400 points. Notice that the graph shows linear dependency of runtime as a function of the number of points in P.

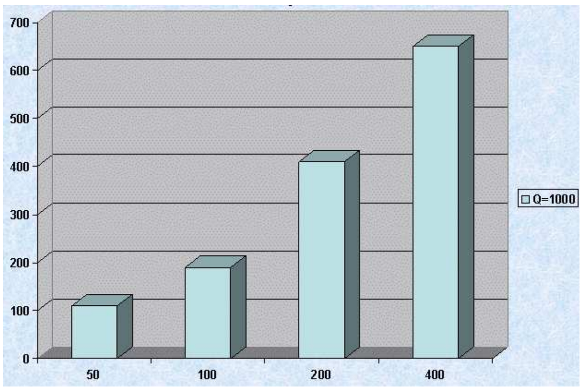


Figure 3: runtime(milliseconds) vs. different number of pattern points, while Q=1000. We used all the speed up methods to reduce the computation time

5.2 Application in real time object recognition

We tested our geometric matching method in a real application of model based object recognition. For a 512x512 gray level input image (Figure 4(a)) and a given geometry model (Figure 4(b)), an edge detection was first applied to the input image (Figure 4(c)), and oriented points were extracted from both the image and the model. For the similarly oriented points we constructed the polytope arrangement and applied our method to compute the region of maximum depth in this arrangement and then retrieved the best transformation from this region. The matching was performed allowing translation and scale change. The parameters are the same as in the synthetic case after normalization of the image data to the range [-1,1] as before. The results are shown in Figure 4(d).

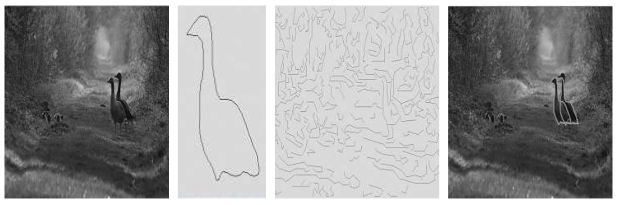


Figure 4: Model based object recognition: (a) input image, (b) model, (c) edge map, (d) detection results. The whole recognition (edge detection + matching) was done in 20 milliseconds while the matching alone took 4 milliseconds

6 Summary

Based on our theoretical work [3], on GPU capabilities and on some caveats we present practical real time algorithms and implementations of shape resemblance. The algorithms work for any three parameters transformation (rigid and scale + translation). Our future plans are to investigate the expansion to higher degree of transformation space and to allow the use of line segments instead of points.

References

[1] C. Everitt, "Interactive order-independent transparency". *Technical report*, NVIDIA Corporation, May 2001.

[2] D.P. Huttenlocher, K. Kedem, and M. Sharir, "The upper envelope of Voronoi surfaces and its applications", *Discrete and Computational Geometry*, 9(1993), pp 267–291.

[3] D. Aiger, Klara Kedem, "Exact and Approximate Geometric Pattern Matching for point sets in the plane under similarity transformations", submitted.

[4] Kenneth E. Hoff III, Andrew Zaferakis, Ming C. Lin, Dinesh Manocha, "Fast and simple 2D geometric proximity queries using graphics hardware". *SI3D 2001*: 145-148

[5] P. K. Agarwal, S. Krishnan, N. H. Mustafa, S. Venkatasubramanian, "Streaming Geometric Optimization Using Graphics Hardware". *ESA 2003*: 544-555.

[6] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krager, A. E. Lefohn, T. Purcell. "A Survey of General-Purpose Computation on Graphics Hardware". *Eurographics 2005*, State of the Art Reports, August 2005, pp. 21-51.

[7] L. P. Chew, K. Kedem, "Getting around a lower bound for the minimum Hausdorff distance", *Comput. Geom.* 10(3): 197-202 (1998).

[8] D.P. Huttenlocher and K. Kedem, "Computing the Hausdorff distance for point sets under translation", *Proceedings of the Sixth ACM Symposium on Computational Geometry*, 1990, pp 340–349.

[9] H. S. Baird, "Model Based Image Matching Using Location", *MIT press,* Cambridge, MA, 1985.

[10] The Industry's Foundation for High Performance Graphics - http://www.opengl.org

Computing Geodesic Disks in a Simple Polygon[*]

Magdalene G. Borgelt[†] Marc van Kreveld[†] Jun Luo[†]

Abstract

Let P be a simple polygon of n vertices and let S be a set of N points lying in the interior of P. A *geodesic disk* $GD(p,r)$ with center p and radius r is the set of points inside P that have a geodesic distance $\leq r$ from p (where the geodesic distance is the length of the shortest polygonal path connection that lies inside P) plus the set of points on the perimeter of P that have a geodesic distance of at most r from p. In this paper we propose an output sensitive algorithm for finding all N geodesic disks centered at the points of S, for a given value of r. Our algorithm runs in $O((n + (kn)^{\frac{2}{3}} + k)\log^c n)$ time, for some constant c and output size k.

1 Introduction

The *geodesic distance* between two points p and q inside P (or on its perimeter) is the length of the shortest path connecting p and q, such that no point of this path lies outside P. Let p be a point inside P or on the perimeter of P. Then a *geodesic disk* $GD(p,r)$ with radius r and center p is the set of points inside P that have a geodesic distance $\leq r$ from p plus the set of points on the perimeter of P that have a geodesic distance of at most r from p. It is easy to see that a geodesic disk is a shape that is bounded by circular arcs (not necessarily of the same radius) and pieces of the perimeter of P (see Figure 1).

Given a set $S = \{p_1, p_2, \cdots, p_N\}$ of N points inside or on the perimeter of a simple polygon $P = \{v_1, v_2, \cdots v_n\}$ with n vertices, and r a fixed real number, we present an output sensitive algorithm that computes all geodesic disks $GD(p_i, r)$, $i = 1, \ldots, N$. Our algorithm runs in $O((n + (kn)^{\frac{2}{3}} + k)\log^c n)$ time for some constant c and output size k. Note that $k = \Omega(N)$ and $O(n \cdot N)$.

To appreciate this result, note that a direct approach to computing a geodesic disk would treat each point $p \in S$ separately by computing the shortest path tree of p inside P, and then determining the circular arcs and boundary parts inside each funnel [6]. This procedure would take at least $O(n)$ time per geodesic disk. Therefore computing all N geodesic disks would take $\Theta(n \cdot N)$ time, which is proportional to the worst case output complexity.

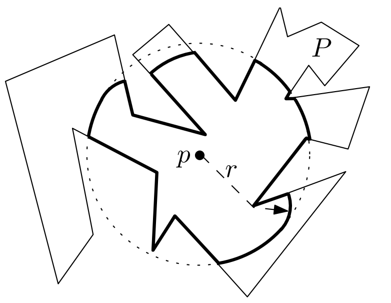


Figure 1: Example of a geodesic disk in a simple polygon. The dotted circle is the boundary of a normal disk centered at p with radius r.

There is a large variety of problems concerning the geodesic distance with respect to a given simple polygon that have been studied in the past. Among them are the computation of the geodesic center of a given simple polygon and its geodesic diameter [5, 7]. Given a simple polygon P with n edges in the plane and a set of point sites in its interior or on its perimeter, Aronov [4] studied computing the Voronoi diagram of the set of sites with respect to geodesic distance, and Toussaint [8] considered finding the geodesic convex hull of a set of points inside P.

2 Algorithm

Let vertices $v_1, v_2, ..., v_n$ of P be in clockwise order. The boundary of P is denoted as ∂P. For two points $x, y \in \partial P$, let $\partial P[v_i, v_j]$ be the part of boudary of P in clockwise order from v_i to v_j. A ray $\overrightarrow{pv_i}$ is a half line which starts at p and goes through v_i. For a point p in P that is visible from v_i, v_j, we have two rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$. We use $wedge(p, v_i, v_j)$ to denote the wedge which starts from ray $\overrightarrow{pv_i}$ and rotates around p clockwise until it reaches ray $\overrightarrow{pv_j}$. Before illustrating our algorithm to compute a geodesic disk $GD(p,r)$ in P, we assume there exist two query algorithms that use preprocessed data structures. We will discuss the details of the data structures and query algorithms in Section 3.

[*]This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503.

[†]Department of Information and Computing Sciences, Utrecht University, {magdalene,marc,ljroger}@cs.uu.nl

1. *CLSF*(p, v_i, v_j): the inputs are a point p in P and two vertices v_i, v_j of P, where p can see v_i, v_j. The query reports the closest line segment or vertex from p among the line segments and vertices of $\partial P[x_i, x_j]$ that are visible from p. If the output is a vertex, then we can use either line segment which is on $\partial P[x_i, x_j]$ and is incident to that vertex as the output.

2. *FVSP*(p, v_i): the inputs are a point p in P and a vertex v_i of P, the query reports the first vertex of the shortest path from p to v_i.

The inputs of the algorithm $\boldsymbol{GD}(p, v_i, v_j, r')$ are a point p inside P, two vertices v_i, v_j of P such that p is visible to v_i, v_j, and the radius r'. The output is the set of line segments of $\partial P[v_i, v_j]$ such that the shortest path distances from p to those line segments are $\leq r'$. To compute the geodesic disk itself, some straightforward extra work is needed; this is deferred to the full paper. At the beginning, p is some point of S, $r' = r$, and $v_i = v_j$, where v_i is a vertex of P that is visible to p, which means the query range is the whole boundary of P. The algorithm runs as follows: using *CLSF*(p, v_i, v_j) we find the closest line segment from p among all line segments of $\partial P[v_i, v_j]$. Let that closest line segment be $v_q v_{q+1}$. If the distance from p to $v_q v_{q+1}$ is larger than r, then we are done. Otherwise $v_q v_{q+1}$ is reported and there exists a closest point $a \in v_q v_{q+1}$.

Lemma 1 *The closest point a to p (by geodesic distance in P) of the line segment reported by CLSF(p, v_i, v_j) is visible from p.*

We assume without loss of generality that a is vertically above p (see Figure 2). The shortest path from p to v_q is a convex chain and its vertices are vertices of P. Let this convex chain be $pv_{l_1}v_{l_2}v_{l_3}\dots v_{l_k}v_q$. We know that $pv_{l_1}v_{l_2}v_{l_3}\dots v_{l_k}v_q$ is on the left side of the line though p, a. Similarly, we have the shortest path from p to v_{q+1} which is a convex chain $pv_{h_1}v_{h_2}v_{h_3}\dots v_{h_{k'}}v_{q+1}$ on the right side of the line through p, a.

$\partial P[v_i, v_j]$ is partitioned into several parts: $\partial P[v_i, v_{l_1}], \partial P[v_{l_1}, v_{l_2}], \partial P[v_{l_2}, v_{l_3}], \dots, \partial P[v_{l_k}, v_q]$, $\partial P[v_{q+1}, v_{h_{k'}}], \dots, \partial P[v_{h_2}, v_{h_1}], \partial P[v_{h_1}, v_j]$; see Figure 2. They give rise to subproblems that we solve sequentially and recursively. Since the subproblems on the left are the same as those on the right, we discuss the situation on the left. First, we solve subproblems $\boldsymbol{GD}(p, v_i, v_{l_1}, r)$ and $\boldsymbol{GD}(p, v_{h_1}, v_j, r)$ recursively. For all other parts $\partial P[v_{l_1}, v_{l_2}], \partial P[v_{l_2}, v_{l_3}], \dots, \partial P[v_{l_k}, v_q]$, as long as the shortest path distance from p to v_{l_t} $(1 \leq t \leq k)$ is $< r'$, we solve the subproblem $\boldsymbol{GD}(v_{l_t}, v_{l_t+1}, v_{l_{t+1}}, r' - (|pv_{l_1}| + \dots + |v_{l_{t-1}}v_{l_t}|))$ recursively. We don't need to compute the whole shortest path from p to v_q. We only need to find pv_{l_1} by *FVSP*(p, v_q). If the distance from p to v_{l_1} is $< r'$, then we find $v_{l_1}v_{l_2}$ by *FVSP*(v_{l_1}, v_q), and so on.

There is one special case. If the output of *CLSF*(p, v_i, v_j) is v_i or v_j, then there are two cases: (assume that v_i is the one reported by *CLSF*(p, v_i, v_j))

1. If v_{i+1} is inside *wedge*(p, v_i, v_j), then the algorithm continues normally.

2. If v_{i+1} is outside *wedge*(p, v_i, v_j), then we need to do a ray shooting query with $\overrightarrow{pv_i}$. Suppose $\overrightarrow{pv_i}$ first hits $v_k v_{k+1}$, which is a line segment of $\partial P[v_i, v_j]$. Then the shortest paths from p to v_k and v_{k+1} separate $\partial P[v_i, v_j]$ into several subparts and the problem becomes several subproblems. We can solve those subproblems sequentially and recursively as above.

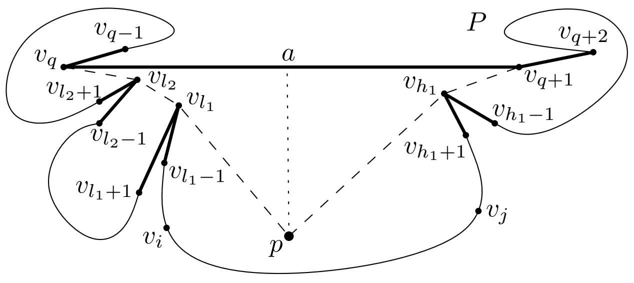


Figure 2: Illustration of the algorithm.

Lemma 2 *The algorithm $\boldsymbol{GD}(p, v_i, v_j, r')$ reports all line segments of $\partial P[v_i, v_j]$ that have geodesic distance at most r' from p at most once, and vertices at most twice.*

Proof (sketch). All recursive problems use a partition of the boundary into disjoint parts, except at the vertices $v_{l_1}v_{l_2}v_{l_3}...v_{l_k}$ and $v_{h_1}v_{h_2}v_{h_3}...v_{h_{k'}}v_{q+1}$ themselves. In all recursive problems of the form $\boldsymbol{GD}(v_{l_t}, v_{l_t+1}, v_{l_{t+1}}, r' - (|pv_{l_1}| + \dots + |v_{l_{t-1}}v_{l_t}|))$, all line segments of $\partial P[v_{l_t+1}, v_{l_{t+1}}]$ will have $v_{l_1}, \dots, v_{l_t}$ as the last vertices on the geodesic shortest path to p. Hence the recursive problem statements are valid. □

3 Data structures for *CLSF* and *FVSP*

In this section we describe the two data structures and query algorithms needed in the algorithm for geodesic disks. We also used a ray shooting data structure; this is standard with $O(\log n)$ query time in preprocessed simple polygons [3].

3.1 Closest boundary point in subpolygon queries

In this section we discuss how to find, for a given query point p and two vertices v_i and v_j of a simple polygon P, the point of the boundary of P between v_i and v_j that is closest to p. Vertices v_i and v_j can also be the answer to the query. We assume that pv_i and pv_j lie completely inside P (in other words: p sees v_i and v_j). To compute geodesic disks, we only need to solve the query problem with this restriction. The closest point q that is found must also be such that segment pq lies inside P.

Assume that some point q on the boundary of P between v_i and v_j is the point closest to p. Because pq lies inside P and p sees v_i and v_j, the angle of $\overrightarrow{pq}$ must be between the angles of $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$.

Without the restriction that pq must be inside P, we could have built a binary search tree T on $v_1, \ldots, v_n$, and construct a Voronoi diagram preprocessed for planar point location as associated structure with every internal node of T. A query would be answered by determining the search paths in T to v_i and v_j, and for all maximal subtrees strictly between these search paths, query the associated structure. But then a point may be found that does not see p inside the whole polygon P (see Figure 3).

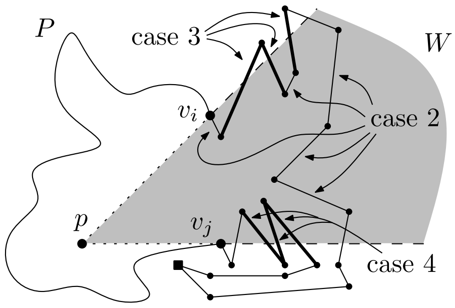


Figure 3: Edges of P of cases 2, 3, and 4 (cases 3 and 4 are shown thicker). Note that the vertex of $\partial P[v_i, v_j]$ closest to p, the square, is not in W.

The solution is to adapt the data structure so that we only query inside the wedge W bounded by the rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$. We have to take care to treat edges that lie partially inside this wedge correctly. We use five different data structures to handle all cases. In each case the main tree T is a binary search tree on $v_1, \ldots, v_n$, and the final associated structure is a planar point location structure on some Voronoi diagram. The first few associated structures (levels between the main tree and the point location structure) allow us to select the vertices or edges to which the case applies [1, 2]. The five cases are the following.

- Vertices inside W.
- Edges of which both endpoints lie inside W.
- Edges that intersect the ray $\overrightarrow{pv_i}$, have one endpoint inside W, and whose angle with $\overrightarrow{pv_i}$ is less than $\pi/2$, measured inside the wedge and closer to p (see Figure 3).
- Edges that intersect the ray $\overrightarrow{pv_j}$, have one endpoint inside W, and whose angle with $\overrightarrow{pv_j}$ is less than $\pi/2$, measured inside the wedge and closer to p (see Figure 3).
- Edges that intersect both rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$, and both angles are less than $\pi/2$.

For the first case we use a partition tree as the main tree. For the second case we use two levels of partition trees. We treat the third case in more detail, the fourth case is the same and the fifth case can be treated in the same manner.

For the third case, let T be the main tree with $v_1, \ldots, v_n$ in the leaves. An internal node μ corresponds to a subchain $v_s, \ldots, v_t$ of the boundary of P. Let $E_\mu = \{v_s v_{s+1}, \ldots, v_{t-1} v_t\}$ be the edges in this subchain. To be able to select all edges that have one endpoint in the wedge W we take one endpoint of each edge and use a partition tree as the second level structure. To select the edges that intersect $\overrightarrow{pv_i}$ among these, we store the other endpoints in a partition tree as well as the third level structure, and the points dual to the supporting lines of the edges as the fourth level structure. In the fifth level structure we select further on the angle condition. This can be done using a binary search tree on the orientations of the edges. The sixth and last level structure is the point location structure on the Voronoi diagram of the edges. The fourth and fifth cases use similar multi-level trees.

For any query wedge, we can use the levels of the tree to select the edges for which each of the cases apply, and query in the Voronoi diagram to find the closest one. Each of the five cases may give an answer, and we can simply take the closest one as the actual closest vertex or edge. All structures use storage $O(n \log^c n)$ and query time $O(\sqrt{n} \log^c n)$ for some constant c. Combinations of cutting trees and partition trees allow us to get faster query times at the expense of storage and preprocessing [1, 2]. For any $n \leq m \leq n^2$, we can get storage and preprocessing time of $O(m \log^c n)$ and query time $O((n/\sqrt{m}) \log^c n)$.

3.2 First vertex of the shortest path queries

We partition P into $O(n)$ geodesic triangles in $O(n)$ time [3]. The three vertices of a geodesic triangle are vertices of P. The three edge chains are three shortest concave paths inside P. In [3] Chazelle *et al.* show that any line segment interior to P crosses at most $O(\log n)$ geodesic triangles. So the line segment pa in Figure 2 crosses at most $O(\log n)$ geodesic triangles. Suppose pa crosses a geodesic triangle bcd. If we walk along pa from p to a, we first hit an edge of geodesic triangle bcd. Assume this edge is the shortest path c–d. There are two cases for the other intersection point of pa with geodesic triangle bcd:

1. pa intersects the shortest path b–c (see Figure 4). Suppose the line segment of the shortest path from b to c intersecting pa is $b'b''$, and b' is on the same side of the line through p, a as v_q. So b' is the only possible vertex that can be the first vertex of the shortest path from p to v_q. Given the shortest path b–c, we can find $b'b''$ in $O(\log n)$ time.

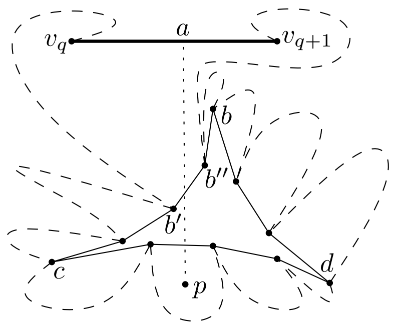


Figure 4: pa intersects the shortest path b–c.

2. pa intersects the shortest path b–d (see Figure 5). The only candidate for the first vertex is the one on a line through p that is tangent to the shortest path b–c. Given the shortest path b–c, we can find that tangent vertex in $O(\log n)$ time.

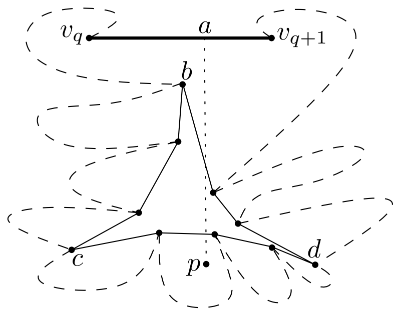


Figure 5: pa intersects the shortest path b–d.

To decide whether a vertex y is the first vertex of the shortest path from p to v_q, we do a ray shooting query with $\overrightarrow{py}$. The first vertex of the shortest path from p to v_q is y if and only if py does not intersect any other line segment of ∂P and the first intersection point of $\overrightarrow{py}$ after y with ∂P is on $v_q v_{q+1}$. This can also be done in $O(\log n)$ time. Since there are $O(\log n)$ geodesic triangles we need to check, the total running time of $FVSP(p, v_q)$ is $O(\log^2 n)$.

4 Complexity analysis

If the output size of N geodesic disks is $O(k)$, then the algorithm will perform $O(k)$ $FVSP$ and $CLSF$ queries. The preprocessing is $O(n)$ time, plus the time needed to build the multi-level trees of Subsection 3.1. We observed that a preprocessing time/query time trade-off exists: $O(m \log^c n)$ preprocessing time leads to $O((n/\sqrt{m}) \log^c n)$ query time. Assume we know k in advance. Then we can choose m to be such that the total query time and preprocessing time are of the same order: $k \cdot (n/\sqrt{m}) \log^c n = m \log^c n$, giving $m = (kn)^{\frac{2}{3}}$ (provided $n \le m \le n^2$).

Unfortunately, the output size k is not known, so we can not balance query time and preprocessing time. To overcome this problem, we will guess k, run the algorithm, and if it turns out that the guess was too low, we double our guess of k and start again: We build a data structure with slightly higher preprocessing time and slightly faster queries. Our initial guess is $k' = \max(n^{\frac{1}{3}}, 2N)$, since we know that $k \ge N$. Since we also know that $k \le n \cdot N$, we will restart the algorithm at most $\log_2 n$ times. The running time is $O((n + (k'n)^{\frac{2}{3}} + k') \log^c n)$ in each round. Summation over the rounds yields $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c.

References

[1] P.K. Agarwal. Range searching. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 36, pages 809–838. Chapman & Hall/CRC, Boca Raton, 2nd ed., 2004.

[2] P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J.E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray Shooting in Polygons Using Geodesic Triangulations. *Algorithmica* 12:54–68, 1994

[4] B. Aronov. On the Geodesic Voronoi Diagram of Point Sites in a Simple Polygon. *Algorithmica* 4:109–140, 1989.

[5] T. Asano, and G. Toussaint. Computing the Geodesic Center of a Simple Polygon. In *Perspectives in Computing: Discrete Algorithms and Complexity,* D.S. Johnson, A. Nozaki, T. Nishizeki, and H. Willis (eds.), pages 65–79, 1987.

[6] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear-time Algorithms for Visibility and Shortest Path Problems inside Triangulated Simple Polygons. *Algorithmica* 2:209–233, 1987.

[7] R. Pollack, M. Sharir, and G. Rote. Computing the Geodesic Center of a Simple Polygon. *Discrete and Computational Geometry* 4:611–626, 1989.

[8] G. Toussaint. Computing geodesic properties inside a simple polygon. *Revue D'Intelligence Artificielle,* 3(2):9–42, 1989.

Generalized Source Shortest Paths on Polyhedral Surfaces

Marta Fort* J. Antoni Sellarès*

Abstract

We present an algorithm for computing shortest paths and distances from a single generalized source (point, segment, polygonal chain or polygon) to any query point on a possibly non-convex polyhedral surface. The algorithm also handles the case in which polygonal chain or polygon obstacles on the polyhedral surface are allowed. Moreover, it easily extends to the case of several generalized sources to compute the (implicitly represented) Voronoi diagram of a set of generalized sites on the polyhedral surface.

1 Introduction

Computing shortest paths on polyhedral surfaces is a fundamental problem in computational geometry with important applications in computer graphics, robotics and geographical information systems [5].

Let $\mathcal{P}$ be a, possibly non-convex, polyhedral surface represented as a mesh consisting of n triangular faces. The single point source *shortest path* problem consists on finding a shortest path in the Euclidean metric from a source point to any target point such that the path stays on $\mathcal{P}$.

Mitchell et al. [4] present an algorithm for solving the single point source shortest path problem by developing a "continuous Dijkstra" method which propagates distances from the source to the rest of $\mathcal{P}$. The algorithm constructs a data structure that implicitly encodes the shortest paths from a given source point to all other points of $\mathcal{P}$ in $O(n^2 \log n)$ time. The structure allows single-source shortest path queries, where the length of the path and the actual path can be reported in $O(\log n)$ and $O(\log n + k)$ time respectively, k is the number of mesh edges crossed by the path. Different improvements of this algorithm have been proposed. In [6] a simple way to implement the algorithm is described and shown to run much faster on most polyhedral surfaces than the $O(n^2 \log n)$ theoretical worst case time. In [1], Chen and Han, using a rather different approach improved this to an $O(n^2)$ time algorithm. Their algorithm constructs a search tree and works by unfolding the facets of the polyhedral surface. The algorithm also answers single-source shortest path queries in $O(\log n + k)$ time. In [2], Kaneva and O'Rourke implemented Chen and Han's algorithm and reported that the implementation is difficult for non-convex polyhedral surfaces and that memory size is a limiting factor of the algorithm. In [3], Kapoor presented an algorithm following "continuous Dijkstra" paradigm that computes a shortest path from a source point to a target point in $O(n \log^2 n)$ time. The difficulties of such an algorithm make complicated its implementation.

In this paper we present an algorithm for computing exact shortest paths, and consequently distances, from single generalized sources (points, segments, polygonal chains or polygonal regions) on a possibly non-convex polyhedral surface represented as a triangular mesh. We also study the case in which obstacles represented by polygonal chains or polygons on the polyhedral surface are allowed. The algorithm easily extends to the case of several generalized sources and gives an implicit representation of the Voronoi diagram of a set of generalized sites on the polyhedral surface. Our algorithm extends the ideas developed in [6] for the efficient implementation of the algorithm of Mitchell et al. to the general source case.

2 Previous Work

The shortest path distance function defined by a source point p on $\mathcal{P}$ is a function D_p such that for any point $q \in \mathcal{P}$, $D_p(q)$ is the Euclidean length of the shortest path along $\mathcal{P}$ from q back to point p. A geodesic is a path that is locally a shortest path, thus, shortest paths are all geodesics, but the converse need not hold. Geodesics on non-convex triangulated surfaces have the following characterization [4]: 1) in the interior of a triangle the shortest path is a straight line; 2) when crossing an edge a shortest path corresponds to a straight line if the two adjacent triangles are unfolded into a common plane; 3) shortest paths can go through a vertex if and only if it is a boundary vertex or its total angle is at least 2π (*saddle* vertex). The basic idea of the algorithm of Mitchell et al. [4] for solving the shortest path problem from a mesh vertex v, as implemented by Surazhsky et al. [6], is to track together groups of shortest paths by partitioning each triangle edge into a set of intervals (windows) so that all shortest paths that cross a window can be encoded locally using a parameterization of the distance function D_v. After an initialization step, where windows encoding D_v in the edges of the triangles containing v are created, the distance function

*Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain, {mfort,sellares}@ima.udg.es. Partially supported by grant TIN2004-08065-C02-02. Marta Fort is also partially supported by grant AP2003-4305.

is propagated across mesh triangles in a "continuous Dijkstra" fashion by repeatedly using a window propagation process. A complete intrinsic representation of D_v is obtained when the propagation process ends. From this representation the shortest path from any point q to the vertex source v is computed by using a "backtracing" algorithm.

3 Exact Generalized Shortest Paths Computation

From now on, a generalized source s on $\mathcal{P}$ refers to a point, segment, polygonal chain or polygon. The shortest path distance function defined by a generalized source s is a function D_s such that for any point $q \in \mathcal{P}$, $D_s(q)$ is the length of the shortest path from q back to source s. Since $D_s(q) = min_{p \in s} D_q(p)$, the shortest path from the generalized source s to any point destination q has the same characterization as the shortest path between two points that we described previously. Notice that if s' is a subsegment of a generalized source s contained in a triangle t of $\mathcal{P}$, the part of a shortest path interior to t starting at an interior point of s' is orthogonal to s'.

3.1 Point Source

For a punctual source p contained in a triangle t we basically use the algorithm developed in [6] for a vertex source with only a few changes in the initialization step. When p is interior to t we create windows encoding the distance function on the edges of t. If p is on an edge e of t and possibly another triangle t', we create: two windows on e going from p to each endpoint of e, respectively; windows on the remaining edges of t and possibly the edges of t'.

3.2 Segment Source

To compute the distance function D_s for a segment source s, as is done in the case of a source point, we track together groups of geodesic paths by partitioning the edges of $\mathcal{P}$ into windows. Geodesic paths that cross a window go through the same triangles and bend at the same vertices of $\mathcal{P}$. In the initialization step, windows defining D_s in the triangle(s) containing s are created. Then the distance field is propagated across triangles in a Dijkstra-like sweep in such a way that over each window, D_s can be represented compactly using an appropriate parameterization.

3.2.1 Distance Function Codification

Consider a shortest path from the source segment s to some point q on an edge e, and let us assume that this path does not bend at any mesh vertex. When all the triangles intersecting the path are unfolded in a common plane, the path forms a straight line. The set of neighboring points of q on e whose shortest paths back to s pass through the same sequence of triangles form: a) a pencil of rays emanating from an endpoint of s; b) a pencil of orthogonal rays emanating from the interior of s (see Figure 1). In both cases we represent the group of shortest paths over a window of the edge e.

Suppose now that the shortest path from $p \in s$ to q bends on one or more vertices on its way to the source s, and let v be the nearest such vertex to q. Again, consider the set of neighboring points on e whose shortest paths back to v go through the same strip of triangles. In the unfolding of the strip between e and v, these shortest paths will form a pencil of rays emanating from the *pseudosource* vertex v, as seen in Figure 1. As before, we represent this group of shortest paths over a window on the edge e.

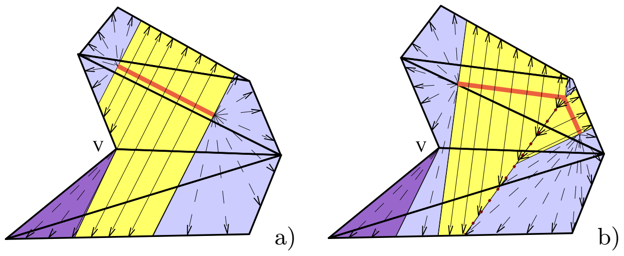


Figure 1: An unfolded strip of triangles with: a) a segment source; b) a polygonal chain source.

Windows representing a group of geodesics emanating from a punctual source p (an endpoint of s or a mesh pseudosource vertex) are called *p-windows*, and windows representing a group of geodesics emanating from interior points of s are called *s-windows*.

p-windows

Following the strategy of Surazhsky et al. [6], the group of geodesics associated to a p-window w originated on a point p (an endpoint of s or a pseudosource vertex) is locally encoded by using a 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. Where b_0, $b_1 \in [0, |e|]$ measure the Euclidian distance from the endpoints of w to the origin of e (the lexicographically smallest endpoint of e). Distances d_0 and d_1 measure the Euclidean distance from the endpoints of w to p, direction τ specifies the side of e on which p lies, and σ gives the distance from p to s. From the 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$ it is easy to position the source p and to recover the distance function within w. It is done by considering the planar unfolding adjacent to e in the rectangular coordinate system $\mathcal{S}_e$ that aligns e with the x-axis as it is shown in Figure 2 a.

s-windows

The group of geodesics associated to an s-window w is locally encoded by using a 5-tuple $(b_0, b_1, d_0, d_1, \phi)$. Where $b_0, b_1 \in [0, |e|]$ are the distances of the endpoints of w to the origin of e, d_0 and d_1 measure the distance of the endpoints of w to s, finally the angle determined by e and the rays emanating from s is stored in $\phi \in [0, 2\pi]$. From the 5-tuple $(b_0, b_1, d_0, d_1, \phi)$ it is easy to position the part s' of

s from which geodesics to w emanate (the visible part of s through the unfolding). Again it is done by considering the planar unfolding adjacent to e in the rectangular coordinate system $\mathcal{S}_e$ that aligns e with the x-axis as it is shown in Figure 2 b. Notice that we do not need d_1 to position s' but it is useful in order to obtain the distance function within w.

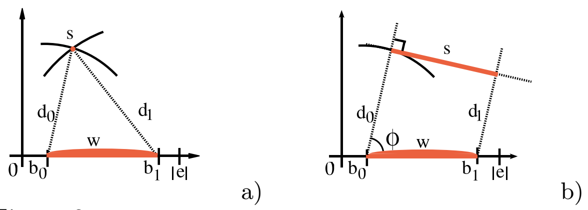


Figure 2: The source s is positioned using the information stored in a: a) p-window, b) s-window.

3.2.2 Window Propagation

We propagate the distance function D_s encoded in a window on an edge e to the next adjacent triangle t by creating new (potential) windows on the two opposing edges of t. They are potential windows because they may overlap previously computed windows. Consequently, we must intersect the potential window with previous windows and determine the combined minimum distance function.

Given a p-window or s-window w on an edge e, we propagate D_s by computing how the pencil of straight rays representing geodesics associated to w extends across one more unfolded triangle t adjacent to e. New potential windows can be created on the opposing edges of t (see Figures 3 a and 3 b). To encode D_s in a new potential window e' first, we obtain the position of the source in the coordinate system $\mathcal{S}_{e'}$. Then, we consider the rays emanating from the source through the endpoints of w to determine the new window interval $[b'_0, b'_1]$ on e'. New distances d'_0 and d'_1 from the window endpoints to the source are computed. For p-windows σ' does not change, and for s-windows the angle ϕ' is the angle defined by the rays and edge e'. When the window w is adjacent to a boundary or saddle vertex v, geodesics may go through it. Vertex v is a new pseudosource and generates new potential p-windows with σ' given by $d_0(d_0+\sigma)$ or $d_1(d_1+\sigma)$ for s-windows(p-windows) (see vertex v of Figure 3 c).

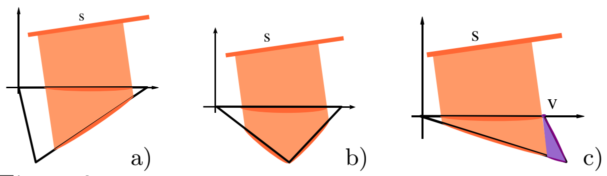


Figure 3: Examples of s-window propagation resulting in a: a) new single window, b) two new windows, c) a new single window and a pseudosource vertex v.

After the propagation, each new potential window w' on edge e' may overlap with previously created windows. Let w be a previously created window which overlaps with w', notice that both w and w' can be either s-windows or p-windows. We have to decide which window defines the minimal distance on the overlapped subsegment $\delta = [b_0, b_1] \cap [b'_0, b'_1]$. To correctly obtain the windows we have to compute the point ξ in δ where the two distance functions coincide. We are discarding the geodesics encoded in w and w' that cannot be shortest paths.

In order to obtain ξ, we define the rectangular coordinate system $\mathcal{S}_{e'}$ that aligns e' with the x-axis in the planar unfolding adjacent to e'. When w and w' are:

- p-windows: we obtain the position of their pseudosources v and v' in $\mathcal{S}_{e'}$ and solve the equation $|v-p|+\sigma = |v'-p|+\sigma'$, as it is done in [6].
- an s-window and a p-window: we obtain the position of the segment source s and pseudosource v on $\mathcal{S}_{e'}$, and solve $d(s,p) = |v-p|+\sigma'$, where $d(s,p)$ is the Euclidean distance from p to s.
- s-windows: we obtain the position of both segment sources s and s' in $\mathcal{S}_{e'}$ and solve the equation $d(s,p) = d(s',p)$.

In all three cases the resulting equation has a single solution.

3.2.3 Continuous Dijkstra

The algorithm uses a Dijkstra-like propagation strategy. In the initialization step, we create windows encoding the distance function on the edges of the triangle(s) containing the segment source s. Those points closer to points in the interior of the segment source are contained in s-windows. On the other hand, those points whose closest point of s is an endpoint of s are contained in p-windows. When windows are created, they are stored in a priority queue which contains both s-windows and p-windows. Windows are stored by increasing distance to the source. The minimum distance from an s-window to source s is $min(d_0, d_1)$. For p-windows we use $min(d_0, d_1)+\sigma$ as weight in the priority queue, although it may not be the minimal distance. It can be done because the obtained solution does not depend on the order in which windows are removed from the queue. However, this weight yields good experimental results according to [6].

The first window of the priority queue is selected, deleted and propagated. Next, overlays are checked, intersections are computed and the new windows are added to the priority queue. Notice that when there is an overlay, windows may be modified or deleted and the priority queue has to be updated accordingly.

3.3 Polygonal Sources

The distance function defined by a polygonal chain is obtained by simultaneously considering all the segments of the polygonal chain in the initialization step. For each segment s of the polygonal chain we create potential s-windows in the face(s) containing s and for each vertex we create potential p-windows. We handle one segment/point after the other and the new potential windows are intersected with the already created ones to ensure that they define the actual distance function. The other parts of the algorithm do not need changes. The distance function defined by a polygonal region r, a connected region of $\mathcal{P}$ whose boundary is a closed polygonal chain, is the distance function of its boundary in the complementary of r, and is 0 in the interior of r. We compute the distance function produced by its boundary polygonal chain creating, in the initialization step, windows only in the complementary of r.

4 Shortest Paths With Polygonal Obstacles.

Given a triangulated non-convex polyhedral surface $\mathcal{P}$ (which may represent a terrain) we consider some obstacles modeled as polygonal chains or polygonal regions (which may represent rivers, lakes, etc). Paths cannot traverse the polygonal obstacles, however, we let paths go along them. Now, geodesic paths can go through a vertex not only if it is a saddle or boundary vertex but also when it is an obstacle vertex. To compute shortest paths we only need to make some modifications in the window propagation process. On the one hand, windows on an obstacle edge are not propagated. On the other, obstacle vertices are new pseudosource vertices regardless of their total angle.

5 Shortest Path Construction

When the propagation of the distance field has finished, the shortest path from any point q on a face f to the source can be obtained by using a backtracing technique similar to the one described in [6]. First, the window w' on the edges of f defining the minimum distance to q is chosen. The distance to q given by a window w is $d_w(q) = d_w(q') + |q - q'|$ with $q' \in w$ the closest point to q and d_w the distance function generated by w. From w', we jump to the adjacent face f' by using the direction τ when w' is a p-window or the angle ϕ when w' is an s-window. We keep on jumping to the adjacent face until we get s.

6 The Case of Several Generalized Sources

The algorithm extends naturally to the case of several generalized sources. In this case we obtain a generalized distance function, which for any point gives the shortest path distance to its nearest source. It is only necessary to change the initialization step where we generate windows for each single source and store them in a unique priority-queue. Thus, we propagate the distance field defined by each single source simultaneously. We automatically obtain a codification of the generalized distance function that yields an implicit representation of the Voronoi diagram of the set of generalized sources.

7 Complexity analysis

Let D be the distance function for a set of generalized sources on a non-convex polyhedral surface $\mathcal{P}$ with polygonal obstacles represented as a mesh consisting of n triangles. It can be proven, following the discussion given in [4], that D can be intrinsically obtained in $O(N^2 \log N)$ time and $O(N^2)$ space, where N is the maximum of n and the total number of segments conforming the generalized sources.

A distance query reporting the length of the shortest path from a point of $\mathcal{P}$ to its nearest source, can be solved by standard methods in $O(\log N)$ time. Finally, the shortest path can be obtained in additional $O(k)$ time, when the path crosses k triangles.

8 Final Comments

We are finishing the implementation of the algorithm for the special case of polyhedral terrains. We expect that in practice the algorithm will run in subquadratic time as in [6]. We are also designing an algorithm to obtain an explicit representation of the Voronoi diagram of a set of generalized sources on the polyhedral surface that is heavily based on the algorithm that computes generalized distance functions.

Finally, we want to mention that the algorithm of Chen and Han can also be adapted to support generalized sources without increasing the time and storage complexity of the original algorithm because the "one angle one split" observation holds. We have chosen to follow Surazhsky et al. strategy because of its easier implementation and good performance in practice.

References

[1] J. Chen, Y. Han. *Shortest paths on a polyhedron.* Int.J. Comput. Geom. Appl., 1996, Vol. 6., 127–144.

[2] B. Kaneva, J. O'Rourke. *An Implementation of Chen & Han's Shortest Paths Algorithm.* Proc. of the 12th Canadian Conf. on Comput. Geom., 2000, 139–146.

[3] S. Kapoor. *Efficient Computation of Geodesic Shortest Paths.* Proc. 32nd Annu. ACM Sympos. Theory Comput., 1999 770–779.

[4] J. Mitchell, D. Mount, H. Paparimitriou. *The discrete geodesic problem.* SIAM J. Computation, **16(4)**, 1987, 647–668.

[5] J. S. B. Mitchell. *Geometric shortest paths and network optimization.* In Jörg-Rüdiger Sack and Jorge Urrutia, editors, Handbook of computational geometry, Elsevier, 2000, pages 633–701.

[6] V, Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, H. Hoppe. *Fast Exact and Approximate Geodesics on Meshes.* In ACM Transactions on Graphics (TOG), Proc. of ACM SIGGRAPH, **24:3**, 2005, 553 - 560.

Improved Algorithms for length-minimal one-sided boundary labeling

Marc Benkert* Martin Nöllenburg*

Abstract

We present algorithms for labeling n points that are contained in a rectangle R by labels that lie on one side of R. The points are connected to their labels by non-intersecting curves (leaders) that each have at most one bend. We consider two types of curves: rectilinear leaders, called *po-leaders*, and leaders that consist of a horizontal and a diagonal segment, called *do-leaders*. To obtain a good readability of the labeling we minimize the total leader length. For the *po*-leaders we give an $O(n \log n)$ and for *do*-leaders an $O(n^2)$-time algorithm. This type of labeling has applications for geographic maps or illustrations in medical atlases in which the labels should not be inserted directly because labels would obscure important information or if points lie too dense.

1 Introduction

Our work ties up to a work of Bekos et al. [2]: for n points contained in a rectangle R and n labels that lie either on one, two or all four sides of R they gave several algorithms for different types of polygonal lines that are allowed to connect the points with the labels. For a good readibility of the labeling they demand that the leaders should be non-intersecting. Further criteria that serve for a good quality are minimizing the total leader length and the total number of bends. One of the leader types that they introduced are the *po*-leaders: the leader starts with a (possibly empty) vertical segment followed by a horizontal segment that connects to the label, see Figure 1(a). For the case that the labels are located only on one side of R, they gave a quadratic algorithm that computes the length-minimum labeling with *po*-leaders. Ali et al. [1] propose several heuristic labeling methods using straight-line and rectilinear leaders. They first compute an initial labeling and then eliminate intersections between leaders. In Section 2 we improve the *po*-leader result of Bekos et al. by giving an $O(n \log n)$ algorithm. Furthermore, we introduce the notion of a *do*-leader. The only difference to a *po*-leader is that the leader starts with a diagonal segment of fixed angle oriented towards the label, see Figure 1(b). To our best knowledge, there is no literature yet that algorithmically deals with *do*-leaders. In practice the *do*-leaders seem to produce nicer labelings because their smoother shape makes the comprehension of the assignment from points to labels easier. In Section 3 we present a quadratic algorithm that computes the length-minimum *do*-labeling with labels on one side.

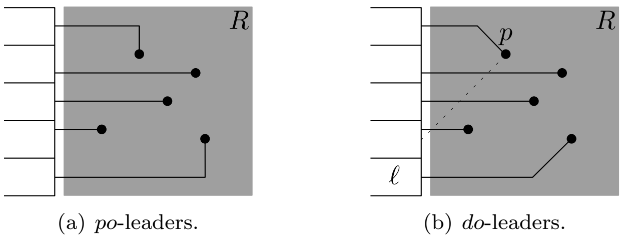


Figure 1: Valid labelings for labels on the left side.

2 *po*-leaders

For simplicity we assume that the labels are uniform and located on the left side of R. We briefly sketch Bekos et al. quadratic algorithm [2] as we will prove the correctness of our algorithm by showing that it produces exactly the same labeling.

Their algorithm proceeds in two steps: first, they produce a non-crossing-free labeling that obviously minimizes the total leader length. This labeling is simply found by sorting the points according to their y-coordinate and then assigning the bottommost point to the bottommost label, the second bottommost point to the second bottommost label and so on. Then, in the second step, they purge all crossings by changing the assignment of two points at a time in an appropiate order. In this second step the total leader length does not change, see Figure 2. Hence, their output is a valid length-minimum labeling. However, in the second step their algorithm potentially has to deal with a quadratic number of crossings which is the bottleneck for the running time.

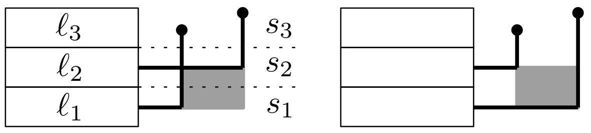


Figure 2: Exchanging the labels of two intersecting leaders without changing the total length.

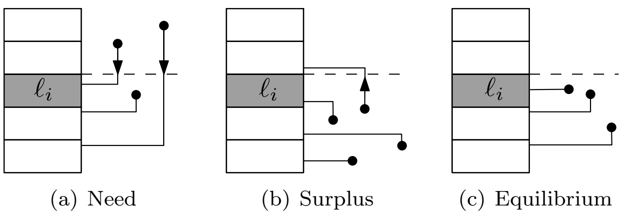
*Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany, `http://i11www.iti.uni-karlsruhe.de/research/geonet/`, GeoNet is supported by grant WO 758/4-2 of the DFG.

The key idea for bringing the running time down to $O(n \log n)$ is to couple these two steps and to not make an assignment for a point until we know that this assignment will not produce any crossings in the remainder of the algorithm. We accomplish this by a sweep-line algorithm.

Let $\ell_1, \ldots, \ell_n$ be the numbering of the labels from bottom to top and let $s_1, \ldots, s_n$ denote the horizontal strips to the right of the according labels, see Figure 2. In a preprocessing step that takes $O(n \log n)$ time we determine point lists $P_1, \ldots, P_n$, where P_i contains exactly the points in s_i, and sort each list according to increasing y-coordinate. Throughout the algorithm we maintain a list L of points that are ordered according to increasing x-coordinate, L contains the points that, for some state of the algorithm, all have to be labeled by a label above or below the current sweep line, initially L is empty. For the sweep itself, there is one event point for every label, namely the horizontal line through the upper horizontal side of the label rectangle. We denote the event point for ℓ_i by $\hat{\imath}$ and the number of points in $s_1 \cup \cdots \cup s_i$ by $n_i^{\cup}$. We distinguish three possible states for an event point $\hat{\imath}$:

Need	if $i > n_i^{\cup}$,
Surplus	if $i < n_i^{\cup}$ and
Equilibrium	if $i = n_i^{\cup}$.

The three states are illustrated in Figure 3. *Need* means that there are too few points in $s_1 \cup \cdots \cup s_i$ for labeling $\ell_1, \ldots, \ell_i$. *Surplus* means that there are too many points in $s_1 \cup \cdots \cup s_i$ for labeling $\ell_1, \ldots, \ell_i$ and *Equilibrium* means that there are exactly i points in $s_1 \cup \cdots \cup s_i$. Note that then, in the length-minimum labeling, these points are assigned to the labels $\ell_1, \ldots, \ell_i$. We can see this by Bekos et al.' algorithm: in their first step the i points are obviously assigned to $\ell_1, \ldots, \ell_i$, meaning that they could have crossings amongst each other but never with other points above. Since Bekos et al. switch labels only of points whose leaders intersect, this means that in the end, these i points remain assigned to $\ell_1, \ldots, \ell_i$.



(a) Need (b) Surplus (c) Equilibrium

Figure 3: The three possible states at event point $\hat{\imath}$.

According to the state that we found at $\hat{\imath}$ we do the following: For *Need* we do nothing at all and proceed with the next event point. The label ℓ_i will get its assigned point during a *backtracking* later on.

For *Surplus* we check the state of the previous event point. If it was *Equilibrium* we assign the bottommost point in P_i to ℓ_i, insert the remaining points of P_i in the then empty list L, remove the first point of L and assign it to ℓ_{i+1}. If it was *Surplus*, label ℓ_i will already have its point assigned by the processing of $\hat{\imath} - 1$. We insert the points P_i into L. After that L must be non-empty because either there were points in P_i or L was still non-empty after processing $\hat{\imath} - 1$. Again, we remove the first point of L and assign it to ℓ_{i+1}. If it was *Need*, we check for which point p in s_i we have an 'artificial' *Equilibrium* and assign p to ℓ_i. We insert the points below p into L and *backtrack*: we set a counter c to $i-1$, as long as L is not empty we do the following: we assign the first point of L to ℓ_c and delete it from L. Then we insert the points P_c into L and decrease c by one. After the backtrack we insert the points above p into L and proceed as above.

For *Equilibrium* we look at P_i. If $P_i = \emptyset$ the previous state was *Surplus* and ℓ_i will already have its point assigned. we have nothing to do. If $|P_i| = 1$, we assign the point in P_i to ℓ_i. If $|P_i| > 1$, the previous state was *Need*, we assign the topmost point of P_i to ℓ_i, insert the remaining points in P_i into the list L and *backtrack* as described above.

Theorem 1 *For labels on one side, the length-minimum labeling using po-leaders can be computed in $O(n \log n)$ time requiring $O(n)$ space.*

Proof. Obviously, each point is inserted in and deleted from L at most once, which establishs the running time since an insertion in the ordered list L is in $O(\log n)$. The linear space requirement is also obvious. It remains to prove that the algorithm finds a valid labeling and that this labeling has indeed minimum length. As mentioned earlier we do this by showing that our algorithm computes exactly the same labeling as Bekos et al.' algorithm. We omit the details but point out that taking the first point of L, i.e. the point with minimum x-coordinate in L, for assigning it to ℓ_c (backtracking) or to ℓ_{i+1} (treating a *Surplus* state) is necessary for the prevention of producing crossings in the further run of the algorithm. □

3 *do*-leaders

For simplicity we assume that the labels are uniform and located on the left side of R. We note that the algorithm will work for any fixed angle for the diagonal segments between 0° and 90° to the x-axis.

We cannot use the same approach as for the *po*-leaders simply as not every point can connect to any label by a *do*-leader, look e.g. back to Figure 1(b) where p cannot connect to ℓ. Roughly speaking we will use a generalization of Bekos et al.' algorithm for the *po*-leaders that takes these restrictions into account.

We start by introducing necessary conditions for the existence of a *do*-labeling and show how to algorithmically make use of them. In the end, we constructively get that the conditions are even sufficient.

Each label ℓ induces a funnel-shaped subregion R_ℓ in which all points that could be assigned to this label are located. The arrangement of all these regions defines $O(n^2)$ cells, see Figure 4. All points in the same cell of this arrangement can connect to the same set of labels and these sets are distinct for any two cells.

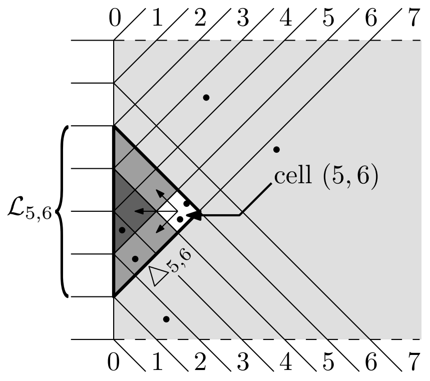


Figure 4: The cell arrangement.

A cell itself is the intersection of an ascending and a descending diagonal strip and after numbering these strips we can index each cell, e.g. the white cell in Figure 4 has index $(5,6)$, when we take the index of the descending strip as first coordinate. For a cell (i,j) we denote the label set that can be reached by $\mathcal{L}_{i,j}$ and the smallest triangle bordering to $\mathcal{L}_{i,j}$ and containing (i,j) by $\triangle_{i,j}$, see Figure 4. Now, a necessary condition for the existence of a *do*-labeling is obviously that the number $n_{i,j}$ of points in $\triangle_{i,j}$ does not exceed the number of labels in $\mathcal{L}_{i,j}$ which is $i+j-n$. Otherwise there will be unassigned points left over in $\triangle_{i,j}$ that cannot connect to any other labels beside $\mathcal{L}_{i,j}$. We say that $i+j-n$ is the *level* of cell (i,j).

Lemma 2 *There can only be a valid do-labeling if for each k-level cell (i,j) it holds that $n_{i,j} \leq k$.*

We can check these necessary conditions in $O(n^2)$ time. For this we have to compute all numbers $n_{i,j}$: initially we set each $n_{i,j}$ to zero. For each input point we determine its containing cell (i,j) and increment $n_{i,j}$ by one. Then, each $n_{i,j}$ gives the number of points in the cell (i,j) but we aim for the number of points in $\triangle_{i,j}$. We traverse the cells in increasing order of their levels. Apparently, all 1-level cells already contain the desired values, for all other cells $n_{i,j}$ is updated based on three predecessor values (see Figure 4):

$$n_{i,j} \leftarrow n_{i,j} + n_{i,j-1} + n_{i-1,j} - n_{i-1,j-1}.$$

This counts each point in $\triangle_{i,j}$ exactly once. The time complexity per cell is obviously constant.

Now, we present our algorithm to compute the length-minimum labeling. We assume that we computed the numbers $n_{i,j}$ in a preprocessing and neither of the necessary conditions has been violated. For a k-level cell (i,j) for which $n_{i,j}$ is k we say that $\triangle_{i,j}$ is *full*, meaning that in any valid *do*-labeling each of the $n_{i,j}$ points in $\triangle_{i,j}$ connects to a label in $\mathcal{L}_{i,j}$. For the algorithm we generalize the above definition: a triangle $\triangle_{i,j}$ is full if the numbers of points in $\triangle_{i,j}$ and labels in $\mathcal{L}_{i,j}$ that have not been assigned yet match. From now on we call such points and labels *open*.

The algorithm traverses the cells in increasing order of their levels and for each level from bottom to top. Whenever we find a k-level cell (i,j) for which $\triangle_{i,j}$ is full, we call the subroutine *complete*$(\triangle_{i,j})$ which computes a length-minimum valid labeling for the remaining open items in $\triangle_{i,j}$. Then, $\triangle_{i,j}$ is marked as completed. Eventually the traversal will examine the n-level cell (n,n) and if not all points have been assigned yet, an assignment for the remaining open points and labels will be found.

In the procedure *complete*$(\triangle_{i,j})$ we process the open labels from bottom to top. Basically for each open label ℓ the point that we assign to ℓ is the first open point that we find when we sweep $R_\ell \cap \triangle_{i,j}$ by a horizontal line from bottom to top. If the placement of the leader inserts any crossing with earlier drawn-in leaders we purge the crossings by flipping assigned labels without changing the total leader length.

However, we have to pay attention during the completion of a triangle $\triangle_{i,j}$: each time we assign a point that does not lie in a 1-level cell, we artificially shift this point into the 1-level cell adjacent to the assigned label, see Figure 7. This decreases the number of open labels for incompleted subtriangles of $\triangle_{i,j}$ while the number of open points in them stays the same, thus, these triangles can become full. If this happens we have to bring the completion of these recently filled subtriangles forward to the usual completion of $\triangle_{i,j}$.

For describing the full operation mode of *complete*$(\triangle_{i,j})$ we have to distinguish two cases:

complete$(\triangle_{i,j})$**:** First, we traverse the incompleted cells of $\triangle_{i,j}$ by a breadth-first search starting from (i,j). This yields lists of the remaining open points and labels in $\triangle_{i,j}$. If the lists of points and labels are empty we mark (i,j) as completed and are done, otherwise we sort both lists according to increasing y-coordinate.

Together with the BFS we purge redundant cells: due to already completed subtriangles of $\triangle_{i,j}$ cells can have become equivalent in the sense that they now can reach the same set of open labels. We merge these equivalent cells and assign the number and level of the topmost-level cell to the newly emerged cell. Obviously, this maintains the number of points and labels in the triangle associated with the new cell, see Figure 5. This step is indispensable for the maintenance of a quadratic running time as the update of the cell entries that we have to do when we make an assignment later on would cause the runtime to get super-quadratic if the number of cells was quadratic.

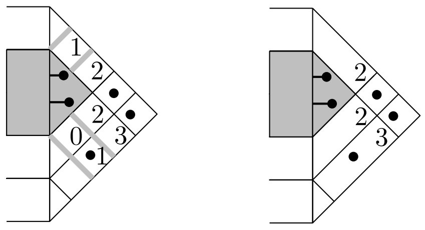


Figure 5: Merging redundant cells.

After finishing these initilizations we start with assigning points to the open labels. For this, we sweep the labels from bottom to top. For an open label ℓ we do the following: we traverse the list of open points and assign the first point p that we find and that is in R_ℓ to ℓ. We remove ℓ and p from the lists of open labels and points. If the leader from p to ℓ intersects earlier drawn-in leaders we take the leader of the topmost label among them and flip the assigned points with ℓ, we repeat this step until there are no crossings anymore, see Figure 6.

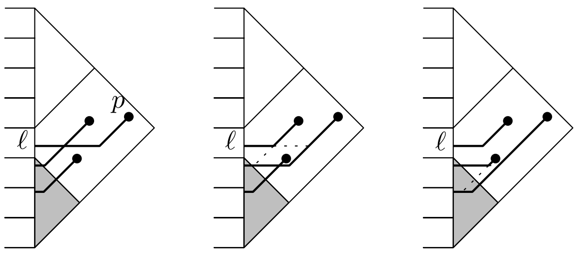


Figure 6: Purging crossings after assigning $p \rightarrow \ell$.

After making an assignment, we update the cell structure and data of $\triangle_{i,j}$. For cells that have become redundant by the assigment this works analogously as for the initilization. For the numbers $n_{i',j'}$ we have shifted the assigned point from its original cell to the cell c_ℓ adjacent to ℓ. We trace the leader from ℓ back to p and update the affected cells accordingly, see Figure 7. This update can cause subtriangles (i',j') to become full. If this happens we have to bring their completion forward. For this, we prepare the lists of open labels and points in $\triangle_{i',j'}$ and start the subroutine *subtriangle-complete*($\triangle_{i',j'}$). Then, we mark (i',j') including the according points and labels as completed and proceed with the completion of $\triangle_{i,j}$.

Finally, after the last open label in $\triangle_{i,j}$ is assigned we mark $\triangle_{i,j}$ as completed.

subtriangle-complete($\triangle_{i',j'}$)**:** As before, only the lists of open points and labels are handed over by the overall procedure.

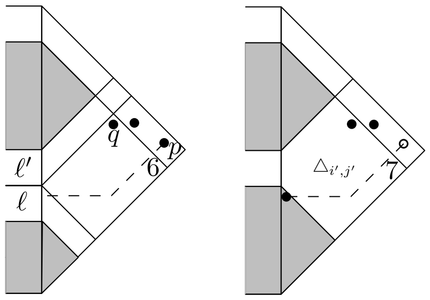


Figure 7: $\triangle_{i',j'}$ becomes full by assigning $p \rightarrow \ell$. The open point q in $\triangle_{i',j'}$ now has to be assigned to ℓ' in order to find a valid labeling.

Theorem 3 *For labels on one side, a valid length-minimum labeling using do-leaders can be computed in $O(n^2)$ time requiring $O(n^2)$ space, if there is any.*

Proof. We assume that the necessary conditions from Lemma 2 hold, otherwise we report infeasibility after the $O(n^2)$-time preprocessing.

For the correctness of the described algorithm it is obviously sufficient to show that the procedure *complete*($\triangle_{i,j}$), not called within the completion of a supertriangle, computes a valid length-minimum labeling within $\triangle_{i,j}$. We do this in the following stages: after *complete*($\triangle_{i,j}$) has finished it holds that ...

1. ... each of the labels $\mathcal{L}_{i,j}$ is assigned to a distinct point in $\triangle_{i,j}$.
2. ... the computed labeling is valid. Any flip that is performed to purge crossings leaves the total leader length unchanged.
3. ... the computed labeling is length-minimum.

For space reasons we have to omit the details of 1.–3. and conclude the proof by showing that the algorithm requires quadratic time and space. Storing the cell structure dominates the space consumption and is quadratic. A call to *complete*($\triangle_{i,j}$), where $\triangle_{i,j}$ has κ open points, requires at most $O(\kappa^2)$ time: after the lists of open points in $\triangle_{i,j}$ have been generated and sorted in $O(\kappa \log \kappa)$ time, finding the point for an open label and updating the list of remaining items is in $O(\kappa)$. For the crossing purges we have to deal with at most $O(\kappa^2)$ crossings in total. Since each point appears as an open point for exactly one full triangle this settles the total running time to $O(n^2)$. □

References

[1] K. Ali, K. Hartmann, and T. Strothotte. Label layout for interactive 3D illustrations. *J. of WSCG*, 13:1–8, 2005.

[2] M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry: Theory & Applications*, 36:215–236, 2007.

Optimal Higher Order Delaunay Triangulations of Polygons [*]

Rodrigo I. Silveira[†] Marc van Kreveld[‡]

Abstract

This paper presents an algorithm to triangulate polygons optimally using order-k Delaunay triangulations, for a number of quality measures. The algorithm uses properties of higher order Delaunay triangulations to improve the $O(n^3)$ running time required for normal triangulations to $O(k^2 n \log k + kn \log n)$ expected time, where n is the number of vertices of the polygon. An extension to polygons with points inside is also presented.

1 Introduction

One of the best studied topics in computational geometry is the triangulation. When the input is a point set P, it is defined as a subdivision of the plane whose bounded faces are triangles and whose vertices are points of P. In this paper we focus mainly on triangulations of polygons. The goal is to decompose the polygon into triangles by drawing diagonals. For a given point set or polygon, many triangulations exist, so it is possible to try to find one that is the best according to some criterion that measures some property of the triangulation. Typical properties are local properties of the triangles (like area, height or minimum angle) or global properties of the triangulation (such as total edge length or maximum vertex degree).

For a given set of points P, a well-known triangulation is the Delaunay triangulation. It is defined as a triangulation where the circumcircle of the three vertices of any triangle does not contain any other point of P. It is unique when no four points are cocircular, and can be computed in $O(n \log n)$ time for n points. The Delaunay triangulation optimizes several measures, like max min angle or min max smallest enclosing circle, among others. This is the reason why its triangles are said to be well-shaped. Gudmundsson et al. [6] define *higher order Delaunay triangulations*, a class of well-shaped triangulations where a few points are allowed inside the circumcircles of the triangles. A triangulation is *order-k Delaunay* if the circumcircle of the three vertices of any triangle contains at most k other points (see Figure 1).

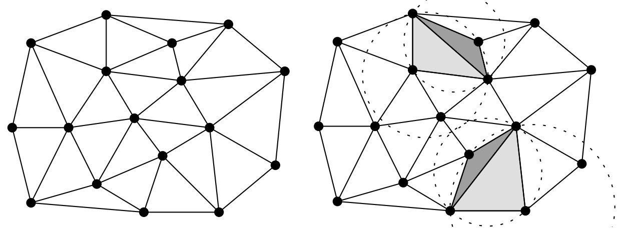


Figure 1: A Delaunay triangulation ($k = 0$) (left), and an order-2 triangulation (right). Light gray triangles are first order, the medium grey ones are second order.

When the goal is to optimize only one criterion, optimal polygon triangulations can be computed in polynomial time for many measures. The main tool for this is a dynamic programming algorithm attributed to Klincsek [9], that allows to find in $O(n^3)$ time an optimal triangulation of a simple polygon for any *decomposable* measure. Intuitively, a measure is decomposable if the measure of the whole triangulation can be computed efficiently from the measures of two pieces, together with the information on how the pieces are glued together. Decomposable measures include the following: min / max angle, min / max circumcircle, min / max length of an edge, min / max area of triangle, and the sum of the edge lengths. The algorithm by Klincsek can be extended to other measures that are not decomposable, like maximum vertex degree. For convex polygons, the min/max area of triangle measures can be optimized faster, in $O(n^2)$ time [8]. A few methods exist for optimally triangulating point sets, and can be applied also to polygons. The edge insertion paradigm [1] can be used to optimize several (decomposable) measures in $O(n^2 \log n)$ or $O(n^3)$ time (depending on the measure). A triangulation of a point set minimizing the maximum edge length can be computed in $O(n^2)$ time [4]. The *greedy triangulation*, which lexicographically minimizes the sorted vector of length edges, can be constructed in $O(n^2)$ time [10].

Our problem is more involved, since we aim at optimizing a measure over higher order Delaunay triangulations, therefore enforcing well-shaped triangles at the same time as optimizing some other measure. There are not many results on optimal higher order Delaunay triangulations. For the case $k = 1$, the

[*]This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503 (GADGET) and under the project GOGO.

[†]Department of Information and Computing Science, Utrecht University, the Netherlands, rodrigo@cs.uu.nl

[‡]Department of Information and Computing Science, Utrecht University, the Netherlands, marc@cs.uu.nl

triangulations have a special structure that allows a number of measures (for example min / max area triangle, total edge length, number of local minima in a terrain) to be optimized in $O(n \log n)$ time [6]. When $k > 1$, fewer results are known. Minimizing local minima in a terrain is NP-hard for orders at least n^{ε}, where ε is any positive constant [3].

In this paper we extend the algorithm of Klincsek [9] to allow to optimize a decomposable measure for a simple polygon over order-k Delaunay triangulations. A straightforward extension of Klincsek's algorithm leads to $O(n^3 \log n)$ running time. Our main contribution is improving this time to $O(k^2 n \log k + kn \log n)$, by exploiting properties of this special class of triangulations. This represents an important improvement given that the small values of k are the ones that are most interesting [3]. An extension to triangulate polygons with points inside is also presented.

2 Higher Order Delaunay Triangulations

We begin by presenting some basic concepts on order-k Delaunay triangulations, together with some results that will be needed later. We assume non-degeneracy of the input set P: no four points are cocircular.

Definition 1 *A triangle $\triangle uvw$ in a point set P is* order-k Delaunay *if its circumcircle $C(u, v, w)$ contains at most k points of P. A triangulation of a set P of points is an* order-k Delaunay triangulation *if every triangle of the triangulation is order-k.*

Definition 2 *For a set P of points, the* order *of an edge $\overline{pq}$ between two points $p, q \in P$ is the minimum number of points inside any circle that passes through p and q. The* useful order *of an edge is the lowest order of a triangulation that includes that edge.*

For brevity, we will sometimes write *order-k* instead of *order-k Delaunay*, and *k-OD edge* instead of *order-k Delaunay edge.*

Lemma 1 *(from [6]) Let $\overline{uv}$ be a k-OD edge, let s_1 be the point to the left of $\overrightarrow{vu}$, such that the circle $C(u, s_1, v)$ contains no points to the left of $\overrightarrow{vu}$. Let s_2 be defined similarly but to the right of $\overrightarrow{vu}$. Edge $\overline{uv}$ is a useful k-OD edge if and only if $\triangle uvs_1$ and $\triangle uvs_2$ are k-OD triangles.*

Lemma 2 *(from [6]) Let $\overline{uv}$ be any Delaunay edge. The number of useful k-OD edges in a triangulation T that intersect $\overline{uv}$ is $O(k)$.*

We extend these results with one more lemma.

Lemma 3 *Let $\overline{uv}$ be a useful k-OD edge. Then there are at most $O(k)$ order-k triangles that have $\overline{uv}$ as one of their edges.*

We provide only a sketch of the proof here. We slide a circle in contact with u and v until it touches a first point r_1 to the right of edge $\overrightarrow{uv}$. This could be a *third point* of a triangle incident to $\overrightarrow{uv}$ because it is possible for $C(u, v, r_1)$ to contain less than $k + 1$ points. We slide the circle again until it touches a second point r_2. Now $C(u, v, r_2)$ contains at least one point (r_1). Continuing in this way it can be seen that $C(u, v, r_{k+1})$ contains at least k points, hence no further point can be a *third point*. An identical argument applies to the left side.

Next we show that all order-k triangles can be computed efficiently.

Lemma 4 *Let P be a set of n points in the plane. In $O(k^2 n \log k + kn \log n)$ expected time one can compute all order-k triangles of P.*

We provide a sketch of the algorithm. There are $O(kn)$ useful k-OD edges [6], with $O(k)$ incident order-k triangles each (Lemma 3), therefore the total number of order-k triangles is $O(k^2 n)$.

All the useful edges can be computed in $O(k^2 n + kn \log n)$ expected time [6]. Moreover, within the same running time we can store for every useful edge $\overrightarrow{uv}$, the two sets of points that are contained in the two circles that determine its usefulness (see Lemma 1). There are at most k of these points on each side. For each side, we will sort the points according to the order in which they are touched when sliding a circle in contact with u and v (as in the proof of Lemma 3). This can be done in $O(k \log k)$ time by sorting the centers of the circles. By going through these sorted lists from left to right, we can count the number of points inside all the circumcircles in $O(k)$ time.

Still, some of these triangles may contain points inside, so we need to discard them. Let $\triangle uvx$ and $\triangle uvy$ be two triangles, and let α_u (α_v) denote the angle of $\triangle uvx$ at u (at v), and β_u (β_v) the same for $\triangle uvy$. It is easy to see that $\triangle uvx$ contains point y if and only if $\beta_u < \alpha_u$ and $\beta_v < \alpha_v$. Each triangle can be represented by a point in the plane using its angles at u and at v. The empty triangles are the ones laying in the lower-left staircase of the point set, and can be found in $O(k \log k)$ time by a sweep line algorithm.

The time needed to find the triangles for one useful edge is $O(k \log k)$. Hence the useful edges and order-k triangles can be found in $O(k^2 n \log k + kn \log n)$ time.

3 Triangulating polygons

As mentioned in the introduction, Klincsek's algorithm allows to triangulate polygons in an optimal fashion for a large number of measures using dynamic programming. In this paper we have the additional requirement that the triangulation must be order-k, therefore only order-k triangles can be used.

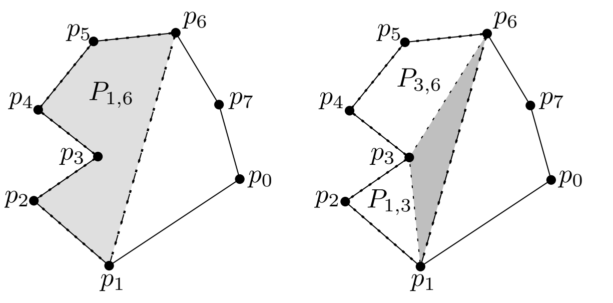


Figure 2: Dynamic programming method. The value of an optimal triangulation of $P_{1,6}$ is stored in $L[1,6]$.

The input is a polygon P defined by its vertices in clockwise order $\{p_0, p_1, ..., p_{n-1}\}$. The output is a k-OD triangulation of optimum cost.

The dynamic programming algorithm finds an optimal solution by combining solutions of smaller problems in a systematic way. The typical algorithms use an $n \times n$ matrix L, which in our problem would have the following meaning: $L[i, i+j]$ contains the cost of the optimal k-OD triangulation of the polygon $P_{i,i+j}$, defined by the edges between p_i and p_{i+j}, plus edge $\overline{p_{i+j}p_i}$. See Figure 2 for an example.

The matrix can be filled in a recursive way. The simplest entries are the ones of the form $L[i, i+1]$, which have cost 0. The formula for $L[i, i+j]$ is:

$$L[i, i+j] = \min_{q=1,2,\ldots,j-1} (Cost(p_i, p_{i+q}, p_{i+j}) + L[i, i+q] + L[i+q, i+j]) \quad (1)$$

Where $\overline{p_ip_{i+q}}$ and $\overline{p_{i+q}p_{i+j}}$ must not intersect the boundary of P. The expression $Cost(p_i, p_{i+q}, p_{i+j})$ is the cost of the triangle $\triangle p_ip_{i+q}p_{i+j}$. Triangles that are not k-OD have cost $+\infty$. Checking the order of the triangle would take $O(\log n + k)$ time [6], but if we precompute all the order-k triangles for each useful edge and store them in a perfect hashing table [5], we can answer the question in $O(1)$ time.

The idea behind the formula is that if edge $\overline{p_ip_{i+j}}$ is part of an optimal triangulation, then it must also contain some triangle incident to it, dividing the polygon into two subpolygons that can be solved independently, see Figure 2. Note that measures of the type $\min\max(\ldots)$ can also be optimized with the same method by a recursive formula similar to (1).

This algorithm has $O(n^3)$ running time because if the matrix is computed in the right way, we always have the required elements of L calculated when computing $L[i, i+j]$. Filling in one cell takes $O(n)$ time, leading to $O(n^3)$ time for the whole matrix.

The special properties of order-k Delaunay triangulation allow to reduce the running time significantly.

First of all, for a given edge $\overline{p_ip_j}$, the number of possible *third points* is not $O(n)$ but $O(k)$ (see Lemma 3). If for every edge we precompute these $O(k)$ points, we can improve the running time to $O(kn^2)$, after spending $O(k^2n \log k + kn \log n)$ in the precomputation of the useful edges and the order-k triangles (Lemma 4).

Secondly, matrix L has $O(n^2)$ cells, each corresponding to one edge. However, only $O(kn)$ can be useful, so it is not necessary to keep a data structure of quadratic size. We will use *memoization* instead. We will apply the recursive formula (1) to compute $L[0, n-1]$, but to avoid solving the same subproblem $L[i, j]$ more than once, we will use a perfect hashing table to store all the subproblems already solved. Since each subproblem $L[i, j]$ is associated with a *k-OD* edge $\overline{p_ip_j}$, only $O(kn)$ subproblems will be computed and stored. Each particular problem is solved in $O(k)$ time, yielding a total running time of $O(k^2n)$, plus $O(k^2n \log k + kn \log n)$ preprocessing time.

Finally, every edge considered must not intersect the polygon boundary and must not lie outside the polygon. This can be checked in $O(\log n)$ time per edge using an algorithm for ray shooting in polygons [7]. This adds an $O(kn \log n)$ preprocessing term, which does not increase the asymptotic running time.

Theorem 5 *An optimal order-k Delaunay triangulation of a simple polygon with n vertices that optimizes a decomposable measure can be computed in $O(k^2n \log k + kn \log n)$ expected time.*

4 Triangulating polygons with points inside

In this section we consider the more general problem of finding an optimal triangulation of a simple polygon that contains $h \geq 1$ components in its interior. A component can be either a point or a connected component made of several points connected by edges. We can reuse the algorithm from the previous section if we connect each component to some other vertex in order to remove all the loose parts. To find the optimal triangulation we must try, in principle, all the possible ways to make these connections.

In principle, there are $O(n)$ ways to connect each component. However, since what we need is to find only one edge to connect the component to the polygon, it is enough to try only $O(k)$ edges. The reason is as follows. Let u be the top-most point inside the polygon. Everything above u is part of the polygon boundary. Let $\overline{uv}$ be a Delaunay edge of the constrained Delaunay triangulation of the polygon and its components, with v above u. By Lemma 2, a Delaunay edge can be crossed by only $O(k)$ useful *k-OD* edges. If $\overline{uv}$ is not part of the optimal triangulation, at least one of the $O(k)$ edges that cross it must be. Let $\overline{xy}$ be the first of these edges (the first one encountered when going from u to v along $\overline{uv}$), then triangle $\triangle uxy$ must be part of the optimal triangulation. This implies that edges $\overline{ux}$ and $\overline{uy}$ are part of it as well. Then in our algorithm we can try $\overline{uv}$ and for each of the $O(k)$ possible edges $\overline{xy}$, either $\overline{ux}$ or $\overline{uy}$ (any of them as long as it connects u to a higher point). At least one of these edges must be part of any optimal triangulation,[1] and connects v to the triangulated part.

Trying every possible way to connect each inner component to the boundary of the polygon involves trying $O(k^h)$ cases. Let $P_1, \cdots, P_\eta$ be the $O(k^h)$ different polygons that are tried, and let H_i be the set of new boundary edges of P_i. For each P_i, besides computing the boundary, we must compute the intersections between the $O(kn)$ useful edges and the new h edges in H_i, which were added to remove the loose components. The computation of these intersections can be done once and maintained between successive polygons without increasing the asymptotic running time. Though we omit the details here, we can compute during the preprocessing phase an intersection graph for the useful k-OD edges, in $O(kn \log n + k^3 n)$ time, and then update it in $O(k^2)$ time per polygon by iterating through the polygons in a certain order.

Triangulating each generated polygon using the algorithm from the previous section takes $O(k^2 n)$ time, yielding a total time of $O(kn \log n + k^3 n + k^h(k^2 + k^2 n))$ $= O(kn \log n + k^{h+2} n)$ (because $h \geq 1$).

Theorem 6 *An optimal order-k Delaunay triangulation of a simple polygon with n boundary vertices and $h \geq 1$ components inside that optimizes a decomposable measure can be computed in $O(nk \log n + k^{h+2} n)$ expected time.*

5 Application to point sets

Any point set can be triangulated using the results from the previous sections if it is seen as a polygon made of its convex hull with points inside. In general, this will lead to a running time exponential in n, so this is of no practical use. For higher order Delaunay triangulations, the situation might be better. Given a point set and an order k, there is always a set of *fixed edges* that are present in any order-k triangulation, and partition the convex hull of the point set into a number of polygons with components inside. For $k = 1$, it is known that these components are always empty triangles or quadrilaterals [6]. For $k > 1$ this is not true anymore, but preliminary experimental results suggest that in practice, for small values of k, the polygons created contain only a few components (Figure 3). Therefore it is possible that the algorithms presented here can be used in practice to triangulate point sets for small values of k.

[1]Actually, it may happen that none of these edges are in any optimal triangulation. That can occur only if $\overline{uv}$ crosses a boundary edge of the polygon that is not useful order-k. Therefore our algorithm we will also consider that boundary edge.

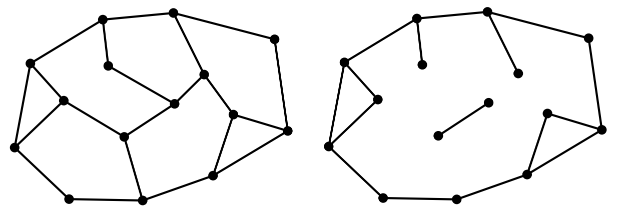


Figure 3: Fixed edges: $k = 2$ (left) and $k = 4$ (right).

6 Discussion

We studied algorithms to find higher order Delaunay triangulations for polygons that optimize a decomposable measure. An existing algorithm for polygon triangulation was extended to optimize over this special class of triangulations. Their specific properties allowed to improve the running time substantially, reducing a $O(n^2)$ factor to $O(k^2)$, a considerable improvement since k will be, in general, significantly smaller than n [3]. Even though not discussed here, other measures, like maximum vertex degree or number of local minima in a terrain, can also be optimized using a similar approach.

References

[1] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T. S. Tan. Edge insertion for optimal triangulations. *Discrete Comput. Geom.*, 10(1):47–65, 1993.

[2] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 590–600, 1988.

[3] T. de Kok, M. van Kreveld, and M. Löffler. Generating realistic terrains with higher-order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 36:52–65, 2007.

[4] H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation. *SIAM J. Comput.*, 22:527–551, 1993.

[5] M. L. Fredman, J. Komlos, and E. Szemeredi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, July 1984.

[6] J. Gudmundsson, M. Hammar, and M. van Kreveld. Higher order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 23:85–98, 2002.

[7] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[8] J. M. Keil and T. S. Vassilev. Algorithms for optimal area triangulations of a convex polygon. *Comput. Geom. Theory Appl.*, 35(3):173–187, 2006.

[9] G. T. Klincsek. Minimal triangulations of polygonal domains. *Discrete Math.*, 9:121–123, 1980.

[10] C. Levcopoulos and A. Lingas. Fast algorithms for greedy triangulation. *BIT*, 32(2):280–296, 1992.

Notes on the Dynamic Bichromatic All-Nearest-Neighbors Problem

Magdalene G. Borgelt* Christian Borgelt†

Abstract

Given a set S of n points in the plane, each point having one of c colors, the bichromatic all-nearest-neighbors problem is the task to find (in the set S) a closest point of different color for each of the n points in S. We consider a dynamic variant of this problem where the points are fixed but can change color. More precisely, we consider restricted problem instances, which allow us to improve over the time needed for solving the problem from scratch after each color change. In these variants we maintain, in $O(n)$ time per color change, a data structure of size $O(cn)$ or $O(n)$, with which the closest neighbor of different color of any point in S can be found in time $O(\log n)$, and the restrictions allow us to bound the number of look ups that are necessary in each step.

1 Introduction

Point proximity problems have been studied intensely over the years. One of the simplest instances is the *closest pair problem*, which consists in the task to find a closest pair of points from a set S of n points in d-dimensional space. It can be solved in $O(n \log n)$ time [5]. Its colored counterpart can be solved by computing a Euclidean minimum spanning tree, thus bounding the time complexity by the one of computing such a tree [10]. Better results are known for the special cases of 2 colors and 2 as well as 3 dimensions [1].

There are two dynamic versions of this problem. In the uncolored variant, the set S is modified by inserting or deleting points. [6] presents a data structure of size $O(n)$ that maintains a closest pair of S in $O(\log n)$ update time per insertion or deletion of a point. In the colored variant, the points in S are fixed, but they can change color dynamically. [10] showed that for an arbitrary number of colors the bichromatic closest pair can be maintained in $O(d + \log\log n)$ update time per color change with a data structure of size $O(n)$.

A natural extension of the closest pair problem is the *all-nearest-neighbors problem*, which is the task of finding the nearest neighbors for all points of a set S of n points. It can be solved, for arbitrary dimension d, in $O(n \log n)$ time [18]. The colored variant of this problem consists in the following task: given a set S of n points, each having one of c colors, find (in the set S) a closest point of different color for each of the n points in S. It can be solved in the plane in $O(n \log n)$ time for an arbitrary number of colors [2]. The static colored all-nearest-neighbor problem has applications in spatial databases and data mining [15, 13, 8, 19] and in information retrieval [7, 9].

Again there are two dynamic versions of this problem. In the uncolored variant, S is modified by inserting or deleting points. It can be solved by maintaining a Voronoi diagram for the set of points, thus bounding the time complexity to that of updating a Voronoi diagram (which in 2 dimensions takes $O(n)$ time per insertion or deletion, see Section 4). In the colored version the points in S are fixed, but they can change color dynamically. We consider this problem only in the plane ($d = 2$) and confine ourselves to restricted problem instances, which allow us to improve on the time that would be needed for solving the problem from scratch for each color change (leading to $O(n \log n)$ time per color change, since the static, colored all-nearest-neighbors problem can be solved in the plane with this time complexity).

2 A Closer Look at the Problem

Suppose that we obtained, by some algorithm, the closest bichromatic neighbor for each point in S for a given coloring of the points. What makes it difficult to maintain these closest neighbors under color changes? Obviously it is not that a point that changed from color i to color j is now eligible as the closest bichromatic neighbor of (other) points of color i. This type of update could easily be achieved in linear time: traverse all points of color i and replace their previously closest bichromatic neighbor if the point that changed color is closer. The difficulty comes from the complementary situation (see Figure 1): if the point that changed color from i to j was the closest bichromatic neighbor of some points of color j, a new bichromatic closest neighbor has to be found for each of these points. This is difficult, because all points not having color j are candidates for the new bichromatic closest neighbor, and simply searching them would take linear time per point that lost its closest neighbor.

The first idea to handle this problem is to maintain a data structure for all differently colored neighbors of a point, so that the next closest can be found quickly

*Department of Information and Computing Sciences, Utrecht University, Netherlands, magdalene@cs.uu.nl

†European Center for Soft Computing, Edificio Cientifico-Tecnologico, c/ Gonzalo Gutiérrez Quirós s/n, 33600 Mieres, Asturias, Spain, christian.borgelt@softcomputing.es

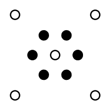


Figure 1: If the white point in the middle changes color, all black points need a new closest neighbor.

if the closest becomes invalid due to a color change. A natural choice for such a data structure would be a heap—like a binomial heap or a Fibonacci heap—for each point, which contains the bichromatic neighbors of that point. Unfortunately, this would not lead to a time complexity per color change that is better than solving the problem from scratch: in the worst case a linear number of heaps have to be updated by deleting points from them, and even the amortized time complexity of a deletion is $O(\log n)$, thus leading to a time complexity of $O(n \log n)$ per color change. In addition, the total size of all heaps would be $O(n^2)$.

3 Restricted Problem Instances

The efficiency of solutions to dynamic point proximity problems is measured by parameters like the preprocessing time, the space needed to store the data structures, the time needed to update the data structures, and the time needed to answer a user-defined query. When we studied the dynamic all-nearest-neighbor problem, we discovered that some options concerning the last two items in this list had not been investigated fully, and that there was room for improvement in certain restricted problem instances. Our rationale is that not all applications require solutions to the general problem and therefore faster solutions to restricted instances may be useful in practice.

Our ideas mainly rest on the insight that the standard problem can be seen as the task to maintain a data structure that allows a user to query for the closest, differently colored neighbor of any point in constant time. Unfortunately, this turns out to be surprisingly difficult. However, we found that it is actually fairly easy to maintain, in linear time, data structures that allow a user to query for the closest neighbor of any point in S in time $O(\log n)$ per point. Hence, if we have an application in which a user queries for the closest neighbor of, say, only a constant number of points after each color change, such a data structure is already very useful.

In addition, if we actually want to maintain, at any point in time, knowledge about the closest neighbor of each point in S, then such a solution can lead to an improvement under constraints. For example, if we face a restricted problem instance in which a color change only invalidates a sub-linear number of bichromatic neighbors (at least on average), having to look up the new neighbors still improves on the time complexity of solving the problem from scratch.

This reasoning led us to two restricted problem instances, in which improvements are possible. Suppose, in the first place, that the number of points that may have the same color is limited to $\frac{kn}{c}$, where n is the total number of points, c the number of colors, and k a real-valued constant greater than 1. Intuitively, this means that the color distribution may not deviate (upwards) more than a factor k from a uniform one. In this case we show that the update, maintaining knowledge of all closest neighbors (i.e., lookup in constant time per point), can be done in $O(\frac{n}{c} \log n)$ time per color change, while solving the problem from scratch still incurs $O(n \log n)$ time.

Alternatively, suppose that the sequence of points that change color is a concatenation of arbitrary permutations of the point set. In other words, in each section of n color changes that starts at index kn, $k \in \mathbb{N}_0$, each point appears at most once. Thus on average each point changes color every n steps, with 0 being the minimum and $2n-2$ being the maximum number of steps. For this case we show that an update takes $O(n)$ time on average, even though it can be as bad as $O(n^2)$ in the worst case. (It should be noted that, in principle, it is possible to generalize this result beyond concatenations of permutations, provided that on average a point still changes color every n steps. However, we confine ourselves to the more restricted version as this simplifies the analysis.)

4 Preliminaries

The data structures we use are always sets of Voronoi diagrams in the plane. Therefore we recall some properties of Voronoi diagrams that we need to show the properties of the update algorithms.

Given a set S of points in the plane, the *Voronoi region* $\mathrm{VR}(p)$ of a point p in S is the set of all points in the plane that are closer to the point p than to any other point in S. The *Voronoi diagram* $\mathrm{VD}(S)$ of S is the collection of all Voronoi regions of the points in S. Two points in S are said to be *Voronoi neighbors* if the closures of their regions have more than one point in common. These common points form a so-called *Voronoi edge*. Sometimes a Voronoi diagram is also defined as the union of all Voronoi edges.

Observation 1 *The number of Voronoi neighbor pairs in a Voronoi diagram for a set S of n points is at most $3n-6$ if $n > 3$.*

This observation follows immediately from the fact that the Voronoi diagram of a set S of points is closely related to the Delaunay triangulation of S: each midperpendicular of an edge in the Delaunay triangulation gives rise to at most one Voronoi edge. Therefore the neighbor graph of the Voronoi diagram (which is obtained by connecting all Voronoi neighbors in S by an edge) is a subgraph of the Delaunay triangulation. As a consequence the neighbor graph has at most as many edges as the Delaunay triangulation, which, as any triangulation, has $3n-6$ edges.

Observation 2 *The sum of degrees in the neighbor graph, over all points in S, is at most $6n - 12$.*

This observation follows trivially from the preceding one, since each edge (Voronoi neighbor pair) contributes to the degree of exactly two points.

The Voronoi diagram of n points can be computed in $O(n \log n)$ time and stored in $O(n)$ space and these bounds have been shown to be optimal in the worst case [16]. They can also easily be preprocessed for point location queries. The methods of [14] and [11], when combined with a linear time algorithm for triangulating simple polygons [17], need $O(n)$ preprocessing time. After preprocessing the Voronoi diagram, a point location query can be answered in $O(\log n)$ time [4]. It was also shown that n-point Voronoi diagrams can be updated in $O(n)$ worst case time per insertion or deletion of a point [12]. Consequently, a data structure allowing for point location queries in Voronoi diagrams can be maintained in $O(n)$ update time per insertion or deletion of a point.

5 Constrained Color Distribution

For the cases where the user queries only for the closest neighbors of a restricted number of points after each color change or the number of points having the same color is bounded by $\frac{kn}{c}$, we form the sets $S_i = \{p \in S \mid p \text{ has color } i\}$, $i = 1, \dots, c$, and $C_i = S - S_i$ (that is, the complement sets of points for each color). Then we build the Voronoi diagrams $\mathrm{VD}(C_i)$, $i = 1, \dots, c$, which takes $O(n \log n)$ time per Voronoi diagram and thus $O(cn \log n)$ total time. Preprocessing these Voronoi diagrams for point location adds $O(cn)$ time and thus does not change the overall time complexity. The total size of the Voronoi diagrams is $O(cn)$, since each has worst case size $O(n)$.

In order to maintain this data structure when a point p changes color from i to j, we have to update two Voronoi diagrams, namely $\mathrm{VD}(C_i)$ and $\mathrm{VD}(C_j)$. The former has to be updated by inserting p, since p now has color j and thus is in the complement of S_i, the latter by deleting p, since p is now in S_j. These two updates take $O(n)$ time each (see Section 4).

Of course, updating the two Voronoi diagrams only yields a data structure that allows us to query for the closest neighbor of each point in $O(\log n)$ time. However, suppose we queried the initial Voronoi diagrams with all points and recorded the closest bichromatic neighbor of each point. In order to update this direct knowledge about closest bichromatic neighbors, we distinguish again the two situations discussed in Section 2. For all points in C_i (except p), we check whether p is now the closest bichromatic neighbor and update accordingly. This obviously takes at most $O(n)$ time. For all points in S_j that had p as theirs closest bichromatic neighbor (which may be all points of S_j in the worst case), we have to look up new closest bichromatic neighbors. Due to the restricted problem instance we consider, we know that $|S_j| \leq \frac{kn}{c}$ and thus that we have to execute at most $O(\frac{n}{c})$ queries, each of which takes $O(\log n)$ time. Therefore the total update time is $O(\frac{n}{c} \log n)$ if we maintain a data structure in which the closest neighbor of any point can be found in constant time.

6 Constrained Update Sequence

For the case where the sequence of points that change color changes is a concatenation of arbitrary permutations of all points in S, we use the algorithm of [2] for the static colored all-nearest-neighbors problem. This algorithm builds $c + 1$ Voronoi diagrams: one for all points in S (that is, $\mathrm{VD}(S)$) and one for each of the c colors. The latter are formed for the sets $T_i = \{p \in C_i \mid p \text{ has a Voronoi neighbor in } S_i\}$, where "Voronoi neighbor" is meant w.r.t. $\mathrm{VD}(S)$. The reason is that in order to find a closest neighbor of different color for each point, it suffices to locate every point $p_i \in S_i$ in the Voronoi diagram of T_i [2]. Hence the Voronoi diagrams $\mathrm{VD}(T_i)$, $i = 1, \dots, c$, are built. In addition, since we need this information for the update, we record for each (occurrence of a) point in each set T_i how many Voronoi neighbors it has in S_i.

Due to Observation 2 we have $\sum_{i=1}^{c} |T_i| \in O(n)$ and thus building all Voronoi diagrams $\mathrm{VD}(T_i)$, $i = 1, \dots, c$, takes $O(n \log n)$ time. Preparing them for the queries takes $O(n)$ time, hence the overall time complexity is still $O(n \log n)$. The total size of all Voronoi diagrams is $O(n)$, and the total number of counters for the Voronoi neighbors in $\mathrm{VD}(S)$ is also $O(n)$.

In order to maintain this data structure when a point p changes color from i to j, we have to update two Voronoi diagrams, namely $\mathrm{VD}(T_i)$ and $\mathrm{VD}(T_j)$. Let $\mathrm{VN}(p)$ be the set of all Voronoi neighbors (w.r.t. $\mathrm{VD}(S)$) of p in S_i, that is, $\mathrm{VN}(p) = \{q \in S \mid q \text{ is a Voronoi neighbor of } p\}$. Since p is now in S_j, we have to add to T_j all points of $\mathrm{VN}(p)$ that are not yet in T_j (and must update $\mathrm{VD}(T_j)$ accordingly), and since p is no longer in S_i, we have to remove those points in $\mathrm{VN}(p)$ that do not have any other Voronoi neighbor in S_i from T_i (and must update $\mathrm{VD}(T_i)$ accordingly) — see Figure 2 for an example. Especially for the latter operation the Voronoi neighbor counters are important, because they make it easy to determine whether a point has another Voronoi neighbor that entitles it to stay in the set T_i.

Updating the Voronoi diagrams takes $O(n)$ time per point that has to be added or deleted. Since, in the worst case, a point can have $O(n)$ Voronoi neighbors, all of which may have to be added or deleted, the worst case time complexity of an update due to a color change is $O(n^2)$. However, one may also consider a scheme where the Voronoi diagrams are rebuild if too many points (more than $k \log n$ for some constant k)

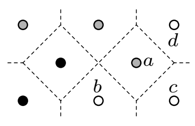


Figure 2: If a changes color from grey to black, b and c are removed from $\mathrm{VD}(T_{\mathrm{grey}})$ and c and d are inserted into $VD(T_{\mathrm{black}})$.

have to be added or deleted, thus reducing the worst case time complexity to $O(n \log n)$.

However, if the sequence of points that change color is a concatenation of arbitrary permutations of all points in S, we obtain a much better average time complexity. In order to demonstrate this, we consider one permutation of all points from the sequence of color changes, say, the section from step kn to step $(k+1)n-1$, $k \in \mathbb{N}_0$. Let p_s be the point that changed color in step s. Then we know from Observation 2 that $\sum_{s=1}^{n} \mathrm{VN}(p_s) \in O(n)$. Consequently, at most a linear number of points have to be inserted into and deleted from Voronoi diagrams in n steps. Since each of these updates take $O(n)$ time each, the total update time in n steps is bounded by $O(n^2)$. Therefore the average update time is linear per color change.

Of course, this procedure only maintains a data structure with which the closest bichromatic neighbor can be found in $(\log n)$ time. In order to improve the situation for a constant time lookup, we have to draw again on the restriction of the number of points that may have the same color. However, the advantage of this procedure is the smaller size of the data structure that needs to be maintained (namely $O(n)$ space instead of $O(cn)$).

7 Conclusions

We described two restrictions that may be introduced for the dynamic version of the all-nearest neighbor problem, in which points are fixed, but can change color. These restrictions allowed us to improve the time complexity of the update operations compared to that needed for solving the problem from scratch after each color change.

Acknowledgments

Partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503.

References

[1] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs. *Discrete and Computational Geometry* 6:407–422. 1991

[2] A. Aggarwal, H. Edelsbrunner, P. Raghavan, and P. Tiwari. Optimal Time Bounds for Some Proximity Problems in the Plane. *Information Processing Letters* 42:55–60. 1992

[3] F. Aurenhammer. Voronoi Diagrams — A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys* 23(3):345–405. 1991

[4] F. Aurenhammer and R. Klein. Voronoi Diagrams. In: J.-R. Sack and J. Urrutia, eds. *Handbook of Computational Geometry*, 210–290. Elsevier Science, Amsterdam, Netherlands, 2000

[5] J. Bentley and M. Shames. Divide and Conquer in Multidimensional Space. *Proc. 8th Ann. ACM Symp. Theory of Computing*, 220–230. 1976

[6] S. Bespamyatnikh. An Optimal Algorithm for Closest Pair Maintenance. *Proc. 11th Ann. Symp. Computational Geometry*, 152–161. 1995

[7] W. Burkhard and R. Keller. Some Approaches to Best-match File Searching. *Communications of the ACM*, 230–236. 1973

[8] F. Cazals. Effective Nearest Neighbours Searching on the Hyper-cube, with Applications to Molecular Clustering. *Proc. 14th Ann. ACM Symp. Computational Geometry.* 1998

[9] L. Devroye and T. Wagner. Nearest Neighbor Methods in Discrimination. In: P.R. Krishnaiah and L.N. Kanal, eds. *Handbook of Statistics, Vol. 2.* North-Holland, Netherlands, 1982

[10] A. Dumitrescu and S. Guha. Extreme Distances in Multicolored Point Sets. *Proc. Int. Conf. Computational Science (ICCS 2002), Part III*, LNCS 2331, 14–25. Springer-Verlag, Heidelberg, Germany 2002

[11] H. Edelsbrunner, L. Guibas, and J. Stolfi. *Optimal Point Location in a Monotone Subdivision.* Report 2, Digital Systems Research Center, Palo Alto, Califonia, USA 1984

[12] I. Gowda, D. Kirkpatrick, D. Lee, and A. Naamad. Dynamic Voronoi Diagrams. *Information Theory, IEEE Transactions* 29:724–731. 1983

[13] T. Hastie and R. Tibshirani. Discriminant Adaptive Nearest Neighbor Classification. *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining*, 142–149. 1995

[14] D. Kirkpatrick. Optimal Search in Planar Subdivisions. *SIAM Journal Comp. Sci.* 28–35. 1983

[15] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, USA 1989

[16] M. Shamos and D. Hoey. Closest Point Problems. *Proc. 16th Ann. IEEE Symp. Foundations of Computer Science*, 151–162. 1975

[17] R. Tarjan and C. van Wyk. A Linear-time Algorithm for Triangulating Simple Polygons. *Proc. 18th Ann. ACM Symp. Theory of Computing.* 1985

[18] P. Vaidya. An $O(n \log n)$ Algorithm for the All-Nearest-Neighbors Problem. *Discrete and Computational Geometry* 4:101–115. 1989

[19] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-Nearest-Neighbors Queries in Spatial Databases. *Proc. 16th Int. Conf. Scientific and Statistical Database Management*, 297–306. 2004

Improving the Construction of the Visibility–Voronoi Diagram

Mojtaba Nouri Bygi[*] Mohammad Ghodsi[†]

Abstract

Ron Wein et al. [4] introduced the Visibility-Voronoi diagram for clearance c, denoted by $VV^{(c)}$, which is a hybrid between the visibility graph and the Voronoi diagram of polygons in the plane. It evolves from the visibility graph to the Voronoi diagram as the parameter c grows from 0 to ∞. This diagram can be used for planning natural-looking paths for a robot translating amidst polygonal obstacles in the plane. They also proposed an algorithm that is capable of preprocessing a scene of configuration-space polygonal obstacles and constructs a data structure called the VV-complex.

As [4] used a straightforward approach for constructing $\mathrm{VV}^{(c)}$-diagram, its construction time is $O(n^2 \log n)$, which is not optimum. In this paper we improve this time to $O(k \log n)$, where k is the number of visibility edges, based on the method used for the preprocessing of the VV-complex in [4].

1 Introduction

Ron Wein et al. [4] have studied the problem of planning a natural-looking collision-free path for a robot with two degrees of motion freedom moving in the plane among polygonal obstacles. By "natural- looking" they mean that the robot should select a path that will be as close as possible to the path a human would take in the same scene to reach the goal configuration from the start configuration. They introduced a new data structure, called the $\mathrm{VV}^{(c)}$-diagram, yielding natural-looking motion paths, meeting all three criteria mentioned above. It evolves from the visibility graph to the Voronoi diagram as c grows from 0 to ∞, where c is the preferred amount of clearance. Beside the straightforward algorithm for constructing the $\mathrm{VV}^{(c)}$-diagram for a given clearance value c, they also propose an algorithm for preprocessing a scene of configuration-space polygonal obstacles and constructing a data structure called the VV-complex. The VV-complex can be used to efficiently plan motion paths for any start and goal configuration and any given c.

In this paper we reduce the time needed to construct the $\mathrm{VV}^{(c)}$-diagram for a given c-value to $O(k \log n)$ where k is the number of visibility edges of the polygons, based on the method used for the preprocessing of the VV-complex in [4].

2 Constructing the $\mathrm{VV}^{(c)}$-Diagram

Here we give the outline of our algorithm for constructing the $\mathrm{VV}^{(c)}$-diagram of an input set $\mathcal{P}$ of pairwise interior-disjoint polygons with n vertices in total for a given c-value, say c_m, in $O(k \log n)$ where k is the number of visibility edges of the initial visibility graph of the polygons.

Our approach is to construct the visibility graph of the polygons, grow c while stopping at critical events that would change the visibility graph, until we get to c_m. This approach is very similar to that of [4] in preprocessing of VV-complex, in which they spend $O(n^2 \log n)$ time in computing the validity range of any possible edge of VV-diagram, while we only need the valid edges at c_m.

We start with a set of visibility edges containing all pairs of the polygonal obstacles. We also include the original obstacle edges in this set, and treat our visibility edges as directed, such that if the vertex u *sees* the vertex v, we will have two directed visibility edges $\vec{uv}$ and $\vec{vu}$.

As c grows larger than zero, each of the original visibility edges potentially spawns as many as four bitangent visibility edges. These edges are the bitangents to the circles $B_c(u)$ and $B_c(v)$ (where $B_r(p)$ denotes a circle centered at p whose radius is r) that we name $\vec{uv}_{ll}$, $\vec{uv}_{lr}$, $\vec{uv}_{rl}$ and $\vec{uv}_{rr}$, according to the relative position (left or right) of the bitangent with respect to u and to v.

Let α_{uv} be the angle between the vector $\vec{uv}$ and the x-axis, and $d(u, v)$ the Euclidean distance between u and v, then it is easy to see that the two edges $\vec{uv}_{ll}$ and $\vec{uv}_{rr}$ retain the same slope α_{uv} for increasing c-values. The slope of the other two edges changes as c grows: $\vec{uv}_{rl}$ rotates counterclockwise and $\vec{uv}_{lr}$ rotates clockwise by the same amount, both around the midpoint $\frac{1}{2}(u + v)$ of the original edge, so their slopes

[*]Department of Computer Engineering, Sharif University of Technology, P.O. Box 11365-9517, Tehran, Iran, nouribaygi@ce.sharif.edu

[†]Department of Computer Engineering, Sharif University of Technology, and IPM School of Computer Science (No. CS1382-2-02), P.O. Box 19395-5746, Tehran, Iran, ghodsi@sharif.edu

become $\alpha_{uv} + \varphi_{uv}(c)$ and $\alpha_{uv} - \varphi_{uv}(c)$, respectively, where $\varphi_{uv}(c) = \arcsin(\frac{2c}{d(u,v)})$.

Note that for a given c-value, it is impossible that all four edges are valid (at most three can be valid, and the ll- and rr-edges can never be valid simultaneously). Our goal is to proceed in growing c while keeping track of valid visibility edges until we reach the final c_m value.

If an edge is valid, then it must be tangent to both circular arcs associated with its end-vertices. There are several reasons for an edge to change its validity status:

- The tangency point of e to either $B_c(u)$ or to $B_c(v)$ leaves one of the respective circular arcs.
- The tangency point of e to either $B_c(u)$ or to $B_c(v)$ enters one of the respective circular arcs.
- The visibility edge becomes blocked by the interior of a dilated obstacle.

The important observation is that at the moment that a visibility edge $\vec{uv}$ gets blocked, it becomes tangent to another dilated obstacle vertex w, so essentially one of the edges associated with $\vec{uv}$ becomes equally sloped with one of the edges associated with $\vec{uw}$. The first two cases mentioned above can also be realized as events of the same nature, as they occur when one of the $\vec{uv}$ edges becomes equally sloped with $\vec{uw}_{lr}$ (or uw_{rl}), when v and w are neighboring vertices in a polygonal obstacle.

This observation stands at the basis of the algorithm we devise for constructing the $\mathrm{VV}^{(c)}$-diagram: We sweep through increasing c-values, stopping at critical visibility events, which occur when two edges become equally sloped, until we reach the goal clearance c_m. We note that the edge $\vec{uv}_{ll}$ (or $\vec{uv}_{lr}$) can only have events with arcs of the form $\vec{uw}_{ll}$ or $\vec{uw}_{lr}$, while the edge $\vec{uv}_{rl}$ (or $\vec{uv}_{rr}$) can only have events with arcs of the form $\vec{uw}_{rl}$ or $\vec{uw}_{rr}$. Hence, we can associate two circular lists $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$ of the left and right-edges of the vertex u, respectively, both sorted by the slopes of the edges. Two edges can have an event at some c-value only if they are neighbors in the list for infinitesimally smaller c. At these event points, we should update the validity of the edges involved, and also update the adjacencies in their appropriate lists, resulting in new events.

In the rest of the paper, we will use the notation $\vec{uv}$ to represent any of the four edges $\vec{uv}_{ll}$, $\vec{uv}_{lr}$, $\vec{uv}_{rl}$ or $\vec{uv}_{rr}$. Moreover, we will use $\mathcal{L}(u)$ to denote either $\mathcal{L}_l(u)$ or $\mathcal{L}_r(u)$ (whether we choose the left or the right list depends on the type of edge involved).

As mentioned in [4], an endpoint of a visibility edge in the $\mathrm{VV}^{(c)}$-diagram may also be an intersection point of dilated obstacle boundaries, which by definition also lies on the Voronoi diagram. Such an endpoint that lies on the Voronoi diagram is called a chain point, as it can be associated with a Voronoi chain in fact, as a Voronoi chain is either monotone or has a single point with minimal clearance, we can associate at most two chain points with every Voronoi chain. Our algorithm will also have to compute the validity for edges connecting a chain point with a dilated vertex or with another chain point. For that purpose, we will have a list $\mathcal{L}(p)$ of the outgoing edges of each chain point p, sorted by their slopes (notice that we do not have to separate the "left" edges from the "right" edges in this case).

2.1 Construction

2.1.1 Initialization

Our algorithm start as follows:

1. Compute the visibility graph of the polygonal obstacles. This can be done in $O(k + n \log n)$ [3] where k is the number of visibility edges, but a simpler $O(k \log n)$ algorithm [3] is also sufficient for our algorithm.
2. Examine each bitangent edge in the visibility graph: For an infinitesimally small c only one of the four edges it spawns is valid - assign `true` to be the value of the validity of this edge.
3. Initialize an empty event queue $\mathcal{Q}$, storing events by their increasing c-order. Hereafter, we only add events with c-order less than c_m to the queue.
4. For each obstacle vertex u:
   (a) Construct $\mathcal{L}_l(u)$ and $\mathcal{L}_r(u)$, based on the edges obtained in step 2. This can be done in total $k \log n$ time.
   (b) Examine each pair of the neighboring edges e_1, e_2 in $\mathcal{L}_l(u)$ and in $\mathcal{L}_r(u)$, compute the c-value at which e_1 and e_2 become equally sloped, if one exists. If the computed c is less than c_m, insert the *visibility event* $\langle c, e_1, e_2 \rangle$ to $\mathcal{Q}$. There are $O(k)$ of such events and updating the queue takes $O(k \log n)$ time.
5. Compute the Voronoi diagram of the polygonal obstacles ($O(n \log n)$)
6. For each *non-monotone* Voronoi chain, locate the arc a that contains the minimal clearance value $c_{\min}$ of the chain in its interior, and insert the *chain event* $\langle c_{\min}, a \rangle$ to $\mathcal{Q}$.

2.1.2 Event Handling

While the event queue is not empty, we proceed by extracting the event in the front of Q, associated with minimal c-value, and handle it according to its type. We note that the visibility events (created, for example, by step 4b of the initialization stage) always come in pairs - that is, if $\vec{uv}$ becomes equally sloped with $\vec{uw}$, we will either have an event for the opposite edges $\vec{vu}$ and $\vec{vw}$, or for the opposite edges $\vec{wu}$ and $\vec{wv}$. We therefore handle a pair of visibility events as a single event:

Visibility event: The edges $\vec{uv}$ and $\vec{uw}$ become equally sloped for a clearance value c', and at the same time the edges $\vec{vu}$ and $\vec{vw}$ become equally sloped.

1. The edges $\vec{uv}$ and $\vec{vu}$ are blocked. Delete them from the edges of the visibility graph.

2. Remove the other event involving $\vec{uv}$ (based on its other adjacency in $\mathcal{L}(u)$) from $\mathcal{Q}$, and delete this edge from $\mathcal{L}(u)$. Examine the new adjacency created in $\mathcal{L}(u)$ and insert its visibility event if is less than c_m into the event queue $\mathcal{Q}$.

3. Repeat step 2 for the opposite edge $\vec{vu}$.

4. If the edge $\vec{uv}$ used to be valid before it was deleted and the edges $\vec{uw}$ and $\vec{vw}$ do not have a true validity value yet, assign it to true, because these edges have become bitangent for this c-value.

The operations above can be done in $O(k \log n)$ because each $O(k)$ edges of the visibility graph can be blocked at most once and deleting such edge would produce $O(1)$ new events.

Chain event: The value c equals the minimal clearance of a Voronoi chain χ_a, obtained on the arc a, which is equidistant from an obstacle vertex u and another obstacle feature. Let z_1 and z_2 be a's endpoints.

1. Initiate two chain points $p_1(\chi_a)$ and $p_2(\chi_a)$ associated with the Voronoi chain χ_a. As c grows, $p_1(\chi_a)$ moves toward z_1 and $p_2(\chi_a)$ moves toward z_2.

2. For all edges $e = \vec{ux}$ incident to u, compute the c-value c' for which e becomes incident to one of the chain points $p_i(\chi_a)$ of a. If c' is less than c_m and is within the range of the Voronoi arc a, then insert the tangency event $\langle c', e, p_i(\chi_a)\rangle$ to the event queue.

3. If a is equidistant to u and to another obstacle vertex v, repeat the last step for the edges incident to v.

4. Let c_1 and c_2 be the clearance values of z_1 and z_2, respectively. Insert the endpoint events $\langle c_1, p_1(\chi_a), z_1\rangle$ and $\langle c_2, p_2(\chi_a), z_2\rangle$ to the event queue.

As each edge of the visibility graph is became incident to a chain point at most twice (one for each of its vertices), so the total time spent in processing chain events is $O(k \log n)$.

When dealing with a chain event, we introduced two additional types of events: *tangency events* and *endpoint events*. We next explain how we deal with these events.

Tangency event: An edge $e = \vec{ux}$ (the endpoint x may either represent a dilated vertex or a chain point) becomes tangent to $B_c(u)$ at a chain point $p(\chi_a)$ associated with the Voronoi arc a.

1. Remove all events involving the edge e from $\mathcal{Q}$.

2. The edge e is blocked, so remove this edge from $\mathcal{L}(u)$. Note that it is possible to disregard the new adjacency created in u's list.

3. Insert a reincarnate of e to $\mathcal{L}(p(\chi_a))$, and assign its validity value to true. Examine the new adjacencies in $\mathcal{L}(p(\chi_a))$ and insert new visibility events, if they are smaller than c_m, into $\mathcal{Q}$.

4. Replace the edge $\vec{xu}$ in $\mathcal{L}(x)$ by $\vec{xp(\chi_a)}$ and recompute the critical c-values of the visibility events of this edge with its neighbors. Modify the corresponding visibility events in Q.

5. In case x is a dilated obstacle vertex, we may have another tangency event in the queue, associated with $\vec{xu}$, which was computed under the (false) assumption that tangency point of the edge on x coincides with a chain point before the one on u does. In this case, we have to locate the tangency event from Q that is associated with $\vec{xu}$ and recompute the c-value associated with it.

Again, each edge of the visibility graph leaves a $\mathcal{L}(u)$ list and enters a $\mathcal{L}(\chi_a)$ list at most twice, so the total time spent in processing tangency events is $O(k \log n)$.

Endpoint event: A chain point $p(\chi_a)$ reaches the endpoint z of a. If z is a local maximum of the clearance function, there are multiple event points associated with it, so we should just assign a false validity value to all edges in the edge lists of all chain points coinciding with z and delete them from the visibility graph. If z is not a local maximum, we have to deal with one of the following two cases:

- z is incident only to two Voronoi arcs a and a' belonging to the same chain ($\chi_a = \chi_{a'}$). In this case the chain point $p(\chi_a)$ is transferred from a to a', and we only have to examine the adjacencies in $\mathcal{L}(p(\chi_{a'}))$ and modify the corresponding visibility events in the queue (as the slopes of these arc become a different function of c from now on). We also have to deal with the opposite edges, as we did in step 4 of the tangency-event procedure. If one of the polygon features associated with the new arc a' is a vertex u, iterate over all edges incident to u and check whether each edge has a tangency event in the range of the new Voronoi arc a' – if so, add this event to the queue $\mathcal{Q}$. If a' is associated with two vertices u and v, repeat the procedure above for v as well.

- z is the endpoint of the chain χ_a (i.e. a Voronoi vertex) and it is not a local maximum of the clearance function. In this case we may have several chains $\chi_1, \chi_2 \ldots$ ending at z, having a synchronous endpoint event, and a single monotone chain $\hat{\chi}$ beginning at z:

  1. Create a new chain point $p(\hat{\chi})$ associated with the monotone chain.
  2. Assign the validity value of each edge in $\mathcal{L}(p(\chi_1)), \mathcal{L}(p(\chi_2)), \ldots$ to false at clearance c, where c is the clearance value at z. Remove all visibility events associated with these edges from $\mathcal{Q}$.
  3. Insert reincarnates of all edges from $\mathcal{L}(p(\chi_1))$ into $\mathcal{L}(p(\hat{\chi}))$, and assign their validity value to true. Examine all adjacencies in $\mathcal{L}(p(\hat{\chi}))$ and add the appropriate visibility event to Q. We also have to deal with the opposite edges, as we did in step 4 of the chain-event procedure.

Note that in the last step all edge lists of the chain points ending at z should be equal ($\mathcal{L}(p(\chi_1)) = \mathcal{L}(p(\chi_2)) = \ldots$), thus we consider only one of these lists. This event should be dealt with before any visibility event occurring at the same c-value, in order to avoid handling visibility events involving duplicate edges. In fact, when we have several events occurring at the same c-value, we deal with endpoint events first, then with visibility events, then chain events and finally tangency events.

As we stated earlier, the complexity of each obstacle is $O(1)$, so the complexity of each Voronoi chain is $O(1)$. Therefore the number of chain events is $O(n)$ in total and $O(1)$ for each Voronoi chain, and the number of times an edge would participate in a chain event is $O(1)$, so the time needed to process all the chain events is $O(k \log n)$.

2.2 Complexity Analysis and Proof of Correctness

As we mentioned before, the algorithm described in section 2 is based on the algorithm given in [4] for preprocessing stage of computing VV-complex, so the proof of correctness would be similar.

Proposition 1 *The construction takes $O(k \log n)$ in total, where n is the total number of obstacle vertices and k is the number of visibility edges in the visibility graph.*

Proof: We first have to compute the visibility graph, which can be performed in $O(k + n \log n)$ time [1], though the $O(k \log n)$ is sufficient for us. This also accounts for the time needed to construct the initial edge lists $\mathcal{L}(u)$ for each obstacle vertex u (there are k edges and n edge lists) and label the valid visibility edges. The construction of the Voronoi diagram can be performed in $O(n \log n)$, and the complexity of the diagram (the number of arcs) is linear.

After the initialization, the priority queue $\mathcal{Q}$ contains $O(1)$ events associated with each of the $O(k)$ visibility edges, and in addition $O(n)$ chain events. Any operation on the event queue thus takes $O(\log n)$. The initialization takes $O(k \log n)$ time in total.

As the construction algorithm proceeds, it starts handling events: In total we have $O(k)$ visibility events, each of them can be handled in $O(\log n)$ time. There are $O(n)$ chain events, and as we said earlier, they can be handled in $O(k \log n)$ time in total. Each visibility edge can participate in a tangency event at most twice , so in total there are $O(k)$ tangency events, each of them can be handled in $O(\log n)$ time. Finally, there are $O(n)$ endpoint events and they will deal with at most $O(k)$ visibility edges, so we need $O(k \log n)$ time to handle them in total.

3 Conclusion

It this paper we studied VV$^{(c)}$-diagram designed by Wein et al. [4] for finding natural-looking paths amid polygonal obstacles. We presented an algorithm for constructing this data structure that runs in $O(k \log n)$ time, where n is the number of vertices of the polygons and k is the number of visibility edges of the visibility graph of the polygons, which is an improvement to current result that works in $O(n^2 \log n)$ time.

It seems that the VV$^{(c)}$-diagram for a fixed c-value may be constructed in $O(k + n \log n)$ time, based on the work of [2], so it may seem we do not need any preprocessing stage, and it is better to construct the VV$^{(c)}$-diagram from scratch whenever we are given a preferred clearance value.

References

[1] S. K. Ghosh and D. M. Mount. *An output sensitive algorithm for computing visibility graphs.* Proc. 28th Annual IEEE Sympos. Found. Comput. Sci., 1987.

[2] M. Pocchiola and G. Vegter. *The visibility complex.* International J. Comput. Geom., 1996.

[3] M. Pocchiola and G. Vegter. *Computing the visibility graph via pseudo-triangulation.* In Proc. Annu. ACM Sympos. Comput. Geom., 1995.

[4] R. Wein, J. P. van den Berg, and D. Halperin. *The Visibility–Voronoi Complex and Its Applications.* In Proc. Annu. ACM Sympos. Comput. Geom., 2005.

Polar Diagram with respect to a Near Pole

Bahram Sadeghi Bigham [*] Ali Mohades [†]

Abstract

Polar diagram of a set of points on the plane, the dual and applications of which has been introduced recently[1, 2]. In this paper we define the Near pole polar diagram and survey some properties of it. Then we present an optimal algorithm to obtain it and discuss the complexity of the algorithm. Also we introduce applications and future works.

Keywords: Polar Diagram, Near Pole Polar Diagram, Voronoi Diagram, Computational Geometry.

1 Introduction

The Voronoi diagram is one of the most fundamental concepts in Computational Geometry and its algorithms and applications have been studied extensively[6]. This concept has also been generalized in a variety of directions by replacing the Euclidean distance with other metrics such as L_p-distance, weighted distances [9], the geodesic distance[3], the power distance[8, 10], and a skew distance. However, some of them are difficult to compute. As the solution to many problems in computational geometry requires some kind of angle processing of the input, some other generalizations of Voronoi diagram based on angle have been studied in [2, 1]. Grima et al. propose a new locus approach for problems processing angles, the polar diagram. For any position q in the plane (represented by a point) the site with smallest polar angle, is the owner of the region where q lies in. Using this portion, it can be solved by an $O(n)$ search problem in an optimal $O(logn)$ location problem and so the polar diagram principle can be used in some important problems requiring angle processing in computational geometry. Grima et al. [2] proved that polar diagram, used as preprocessing, can be applied to many problems in computational geometry in order to speed up their processing times. Some of these applications are the convex hull, visibility problems, and Path Planning problems. Jarvis's March approach can be improved to become an optimal time process and visibility problems can take advantage of polar diagram principles as well. Also Sadeghi et al. introduced in [1] the dual of polar diagram and some properties and applications.

Polar Diagram is the plane partition with similar features to those of the Voronoi diagram. In fact, the polar diagram can be seen in the context of the generalized Voronoi diagram. The *Polar Angle* of the point p with respect to s_i, denoted as $ang_{s_i}(p)$, is the angle formed by the positive horizontal line of p and the straight line linking p and s_i.

Given a set S of n points in the plane, the locus of points having smaller positive polar angle with respect to $s_i \in S$ is called *Polar Region* of s_i. Thus,

$$\mathcal{P}_S(s_i) = \{(x,y) \in E^2 | ang_{S_i}(x,y) < ang_{S_j}(x,y); \forall j \neq i.$$

The plane is divided into different regions in such a way that if the point $(x,y) \in E^2$ lies into $\mathcal{P}_S(s_i)$, it is known that s_i is the first site found performing an angular scanning starting from (x,y). We can draw an analogy between this angular sweep and the behavior of a radar. Figure 1 depicts the polar diagram of a set of points in the plane and the final division constructed using the smallest polar angle criterion.

Although the polar diagram of n points in the plane is not a graph, we define its dual as well as a dual of graph. We said, two points (sites) are joined by the edge e^* in the dual of polar diagram if and only if their corresponding faces are separated by the edge e in polar diagram. So we may have some parallel edges or loops in the dual of a polar diagram. If we omit the loops and replace the parallel edges with one edge, then we will have another graph named Extracted Dual of polar diagram($\mathcal{EDPD}$) (Figure 2). There is an optimal algorithm to draw $\mathcal{EDPD}$ in [1].

In this paper, we define a new extension of Voronoi diagram and call it *Near Pole Polar Diagram ($\mathcal{NPPD}$).* Then we present an optimal algorithm to find it and apply $\mathcal{NPPD}$ for some applications.

This paper is structured as follows: in Section 2 we introduce the problem Near Pole Polar Diagram and present an optimal algorithm to draw it. In Section 3 we present some applications and finally in Section 4 we state some feature works and open problems.

[*]Department of Applied Mathematics, Faculty of Mathematics and Computer Science, Amirkabir University of Technology, No.424, Hafez Ave., Tehran, Iran, `b_sadeghi_b@aut.ac.ir Corresponding author(B.Sadeghi B.) Tel:+982164542545, Fax: +982144468109`

[†]Department of Applied Mathematics, Faculty of Mathematics and Computer Science, Amirkabir University of Technology, No.424, Hafez Ave., Tehran, Iran, `mohades@aut.ac.ir`

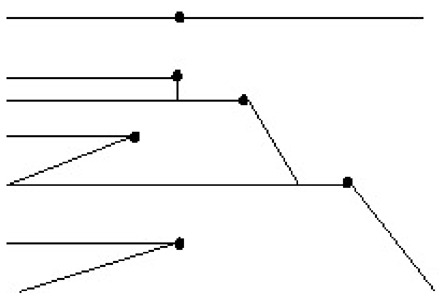


Figure 1: Polar diagram of 6 point sites.

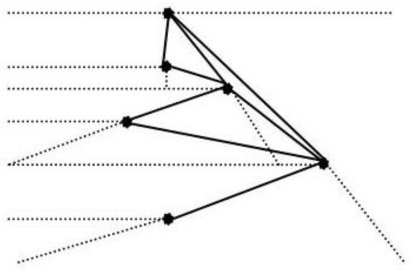


Figure 2: Extracted dual of polar diagram.

2 Polar diagram with respect to a near pole

In this section we introduce the Near pole polar diagram of a set of n point sites in the plane. Also we are going to present an optimal algorithm to draw it. Our approach is an incremental approach.

2.1 Near Pole Polar Diagram ($\mathcal{NPPD}$)

We introduce the Near pole polar diagram with similar features to those of the Polar diagram that can be seen in the context of the generalized Voronoi diagram [6]. As described in [1, 2], the pole mentioned lies on the left hand side of the plane at $-\infty$. In the polar diagram with respect to a near pole, it is assumed that the pole is located on the left hand side of the sites close to them. This allows us to find more applications for the problem. For example, the pole can be considered as the center of vision (eye) of a robot.

In addition to the given point sites, the point p in the plane is also given as a pole, and the partitioning of the plane will depend on the position of p. W.L.O.G assume that the pole p is located on the left-hand side of the sites. Figure 3 shows an example of $\mathcal{NPPD}$ for 7 points with respect to pole p.

In short, a Near pole polar diagram can be described as follows. Initially there is a radar at each of the point sites looking at the pole. They simultaneously start to rotate in counterclockwise direction and scan their periphery. The region in the plane observed by radar p_i before other radars will be called the region of p_i's $\mathcal{NPPD}$.

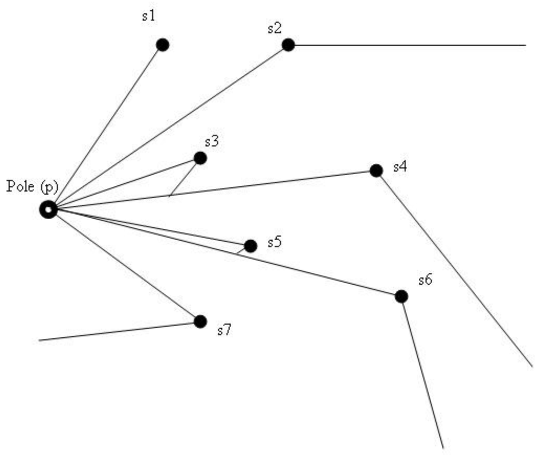


Figure 3: Near pole polar diagram of 7 points.

In Figure 4, we are given sites s_1 and s_2, pole p and point x in the plane. Since $\widehat{ps_1x} < \widehat{ps_2x}$, in the plane's partition, x will belong to the region of s_1. However this partitioning will produce disconnected regions with curved boundaries and makes the problem more complex and drawing the corresponding diagram more difficult. In this paper we have made an assumption which not only makes the problem simpler, but also increases its applications.

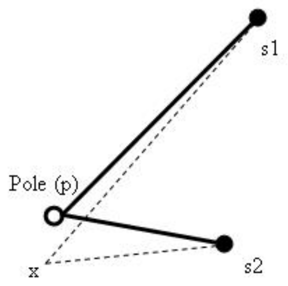


Figure 4: Our assumption: the line segment ps_2 blocks s_1's line of view.

2.2 Incremental approach

In this paper, we are going to present an incremental algorithm for drawing $\mathcal{NPPD}$, working in optimal time. In this paradigm, we first compute the tangent of the line segments $s_ip : i = 1, ..., n$ and sort them. Then a straight half line starting at pole p rotates in clockwise direction around the pole p and sweeps the plane. The $\mathcal{NPPD}$ region of s_i is built according to the following theorem.

Theorem 1 *Let S'_i denote the set of processed points when point s_i is reached, $S'_i = S'_{i-1} \cup s_i$. If $s_i \in$*

$\mathcal{NPPD}_S(s_k), s_k \in S'_{i-1}$ then the near pole polar region of s_i is the angular sector defined by the half line from s_i to the pole p and the half line defined by s_i and s_k, which does not contain s_k.

Proof. Consider point x within the region mentioned in the theorem. Since according to our initial assumption the line segment ps_i blocks other sites' (and specially s_k's) line of view, then the above mentioned region in $\mathcal{NPPD}$ belongs to s_i. Also as $\widehat{s_kpy} < \widehat{s_ipy}$, there can not exist any other points such as y within s_i's region (see Figure 5). □

Theorem 1 is the key to compute the $\mathcal{NPPD}$ using the Incremental method. Algorithm 1 describes the process: $S = \{s_0, s_1, ..., s_{n-1}\}$ is given. $TS = \{ts_0, ts_1, ..., ts_{n-1}\}$ is sorted from the largest member to the smallest one. The $\mathcal{NPPD}$ region of s_i ($\mathcal{NPPD}_S(s_i)$) is computed when $\mathcal{NPPD}_S(s_0)$, $\mathcal{NPPD}_S(s_1)$,...,$\mathcal{NPPD}_S(s_{i-1})$ have been already processed according to theorem 1.

In what follows an algorithm for drawing $\mathcal{NPPD}$ in the plane is presented which takes optimal $\theta(nlogn)$ time.

```
Algorithm 1
Input: A set $S = \{s_0, s_1, ..., s_{n-1}\}$ of $n$ point sites
and a point $p$ as pole in $E^2$.
Output: $\mathcal{NPPD}(S, p)$.
Begin
    Step 1: Calculate the tangent of all lines $s_ip$
     and make new set $TS$.
    Step 2: Sort $TS$ by decreasing order obtaining
     $TS = \{ts_0, ts_1, ..., ts_{n-1}\}$
    Step 3: Let be $TS' := \{ts_0\}$
    Step 4: $TS := TS - \{ts_0\}$
    Step 5: While $\{TS \neq \emptyset\} Do$
        (a): Let $ts_i$ be the maximum value of $TS$
        (b): Do $TS' := TS' \cup \{ts_i\}$ and
       $TS := TS - \{ts_i\}$
        (c): Construct $\mathcal{NPPD}_S(s_i)$ according to
        theorem 1.
        (d): Discard all edges inside $\mathcal{NPPD}_S(s_i)$
     Endwhile
End.
```

We can solve a sorting operation in $O(nlogn)$ time and theorem 1 ensures this time complexity after computing all near pole polar regions.

Assume that n numbers $x_i; i = 1, ..., n$ are given. We can find a function to map the numbers into interval $[-1, 1]$ and calculate n new numbers $x'_i; i = 1, ..., n$. Lets locate n point sites in the plane on $(1, x'_i); i = 1, ..., n$ coordination and the pole p on the $(0, 0)$ (Figure 6). Now using $\mathcal{NPPD}$ for these point sites with respect to the pole p we can sort the points $(1, x'_i); i = 1, ..., n$. So in this way we can sort

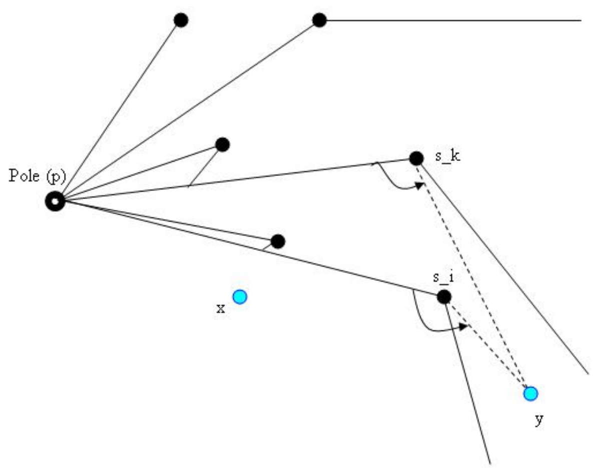


Figure 5: Incremental approach to drawing $\mathcal{NPPD}$.

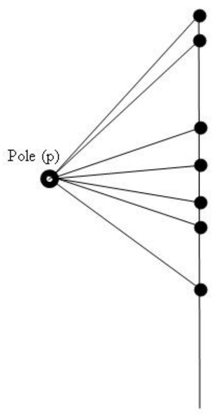


Figure 6: Contradiction: sorting n given numbers at a time less than $O(nlogn)$.

n given numbers at such time and this is a contradiction. Therefore this time complexity is a lower bound and we have the following:

Theorem 2 *The near pole polar diagram of a set of n points in the plane with respect to a given pole p must be computed in $\theta(nlogn)$.*

3 Some applications

In this section we are going to briefly address a number of applications for $\mathcal{NPPD}$. But before that, it is worth mentioning that as in [1], it is possible to define and solve the extracted dual of $\mathcal{NPPD}$. It can also be defined for other objects in the plane such as for line segments, convex polygons and circles.

Let S be a set of n disjoint line segments in the plane, and let p be a point not on any of the line segments of S. Using the $\mathcal{NPPD}$ of the line segments with respect to pole p and the extracted dual of it, we can find all line segments of S that p can see, that is, all line segments of S that contain some point q so that the open segment $\overline{pq}$ does not intersect any line segment of S.

In addition to the applications of $\mathcal{NPPD}$ to computer graphics, visibility and path planning problems, it is also possible to draw decorative patterns by assigning certain points in the plane and drawing $\mathcal{NPPD}$ in two directions (with point sites lying on the left-hand side or the right-hand side of the pole), with application in architecture. A sample of such patterns is shown in Figures 7, 8 in which the sites lie on some concentric circles.

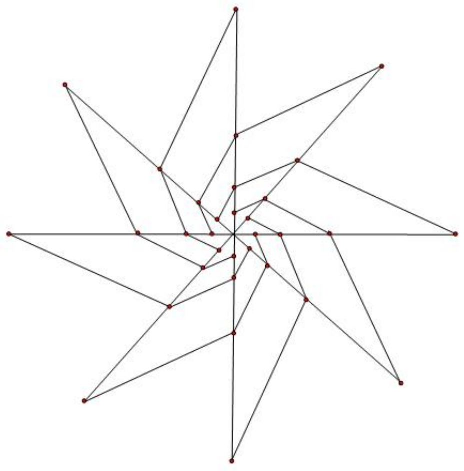


Figure 7: $\mathcal{NPPD}$ for some sites which lie on some concentric circles.

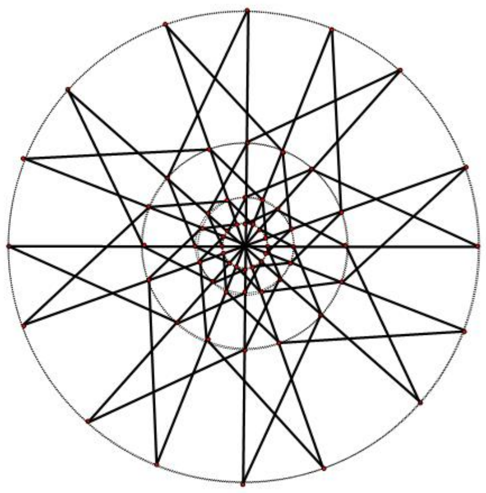


Figure 8: $\mathcal{NPPD}$ (in two directions) for some sites which lie on some concentric circles.

4 Conclusion and future works

We defined in this paper the Near pole polar diagram ($\mathcal{NPPD}$) and presented an optimal algorithm to find it. The algorithm runs in $\theta(nlogn)$ time for n given point sites and a given pole p in the plane. We also briefly introduced some applications of $\mathcal{NPPD}$.

Some problems related to robots' vision can be modeled using $\mathcal{NPPD}$ and its extracted dual. $\mathcal{NPPD}$ is also closely related to the collision detection problems, allowing one to employ methods similar to those in [4].

References

[1] B. Sadeghi Bigham, A. Mohades, *The dual of polar diagram and its extraction.* in: International Conference of Computational Methods in Sciences and Engineering (ICCMSE), Greece, 2006.

[2] C. I. Grima, A. Marquez and L. Ortega, *A new 2D tessellation for angle problems: The polar diagram.* Computational Geometry, 34 ,2006, 58-74.

[3] Zahra Nilforoushan, Ali Mohades, *Hyperbolic Voronoi diagram.* ICCSA (5), 2006, 735-742.

[4] L. Ortega, F. Feito, *Collision detection using Polar diagram.* Computers and Graphics 29, 2005, 726-737.

[5] C.I. Grima, A. Mrquez, *Computational Geometry on Surfaces.* Kluwer Academic Publishers, 2001.

[6] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, *SpatialTessellationsConcepts and Applications of Voronoi diagrams*, second ed., Wiley, Chichester, 2000.

[7] C.I. Grima, A. Mrquez, L. Ortega, *Polar diagrams of geometric objects'* in: 15th European Workshop in Computational Geometry, Antibes, France, 1999.

[8] F. Aurenhammer, *Power diagrams-properties, algorithms and applications.* SIAM J. Comput. 16 ,1987, 78-96.

[9] P.F. Ash, E.D. Bolker, *Generalized Dirichlet tessellations.* Geom. Dedicata 20 ,1986, 209-243.

[10] H. Imai, M. Iri, K. Murota, *Voronoi diagram in the Laguerre geometry and its applications.* SIAM J. Comput. 14 ,1985, 93-105.

Net-aware Critical Area extraction for VLSI opens via Voronoi diagrams

Evanthia Papadopoulou*

Abstract

We address the problem of computing critical area for opens in a circuit layout in the presence of loops and redundant interconnects. The extraction of critical area is the main computational problem in VLSI yield prediction for random manufacturing defects. Our approach first models the problem as a graph problem and solves it efficiently by exploiting its geometric nature. The approach expands the Voronoi critical area computation paradigm [10, 7] with the ability to accurately compute critical area for missing material defects in a net-aware fashion. Generalized Voronoi diagrams used in the solution are combinatorial structures of independent interest.

1 Introduction

Catastrophic yield loss in integrated circuits is heavily attributed to random particle defects interfering with the manufacturing process resulting in functional failures such as open or short circuits. Random defect yield loss has been studied extensively resulting in several yield models (see e.g. [12, 11, 1]). The focus of all random defect yield models is the concept of critical area, a measure reflecting the sensitivity of a design to random defects during manufacturing. Reliable critical area extraction is essential for IC manufacturing especially when *design for manufacturability* (DFM) initiatives are under consideration.

The critical area of a circuit layout on a layer A is defined as

$$A_c = \int_0^\infty A(r)D(r)dr$$

where $A(r)$ denotes the area in which the center of a defect of radius r must fall in order to cause a circuit failure and $D(r)$ is the density function of the defect size. Critical area analysis is typically performed on a per layer basis and results are combined to estimate total yield. The defect density function has been estimated as follows [1, 5, 11, 14]:

$$D(r) = \begin{cases} cr^q/r_0^{q+1}, & 0 \le r \le r_0 \\ cr_0^{p-1}/r^p, & r_0 \le r \le \infty \end{cases} \quad (1)$$

where p, q are real numbers (typically $p = 3, q = 1$), $c = (q+1)(p-1)/(q+p)$, and r_0 is some minimum optically resolvable size. Using typical values for p, q, and c, the widely used defect size distribution is derived, $D(r) = r_0^2/r^3$. (r_0 is typically smaller than the minimum feature size thus, $D(r)$ is ignored for $r < r_0$). Following a common practice to facilitate critical area computation, a defect of size r is modeled throughout this paper as a square of radius r i.e., a square of side $2r$. Modeling defects as squares corresponds to computing critical area in the L_∞ metric. A formal bound for critical area between square and circular defects is given in [7]. For computational simplicity [9] the L_∞ metric is used throughout the paper.

In this paper we focus on critical area extraction for *opens* resulting from broken interconnects. The results are a generalization of the results presented in [7]. Opens are *net-aware*, that is, a defect forms a fault if it is actually *breaking* a net. A net is said to be broken if terminal points of the net get disconnected. An open circuit may also be caused by a *via-block* i.e., a defect on a via or contact layer that entirely destroys a connection (a via or cluster of vias) between neighboring conducting layers (see [7, 8]). In order to increase design reliability and reduce the potential for open circuits designers are increasingly introducing redundant interconnects creating loops that may span over a number of layers (see e.g. [4]). Redundant interconnects reduce the potential for opens at the expense of increasing the potential for shorts. The ability to perform trade-offs is important requiring accurate critical area computation for both. In this paper we accurately compute critical area for opens even in the presence of loops. Note that a loop reduces the potential for open faults but does not necessarily provide immunity to opens: a defect may still create an open by breaking two or more wires disconnecting the loop. The problem is modeled as a graph problem and solved efficiently by exploiting the geometric nature of it. A Voronoi diagram of the layout modeling opens allows accurate critical area computation. The algorithms are being integrated into the IBM Voronoi Critical Area Analysis tool (*Voronoi CAA*) currently used in production mode for chip manufacturing by IBM microelectronics (see [6] for results on the industrial use of the tool).

2 Problem Formulation

From a layout perspective a net is a collection of interconnected shapes spanning over a number of layers. Some of those shapes are designated as *terminal*

*IBM T.J. Watson Research Center, Yorktown Heights NY 10598, `evanthia@acm.org`

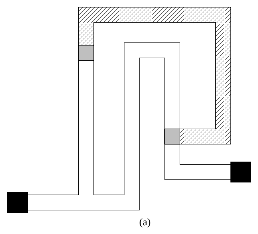


Figure 1: (a) A net N spanning over two layers. (b) Dark defects create opens while transparent defects are no faults.

shapes representing the entities that must remain interconnected. A net remains *functional* as long as all terminal shapes comprising the net are interconnected. Otherwise the net is said to be *broken*. Figure 1(a) illustrates a simple net spanning over two metal layers (say M1 and M2, where M2 is illustrated shaded). The two contacts illustrated as black squares have been designated as terminal shapes. Figure 1(b) illustrates defects that create opens by breaking the net as dark squares and defects causing no fault as transparent squares in dashed lines.

Given a net N, the portion of N on a layer X, denoted $N_X = N \cap X$, consists of a number of connected components. Every connected component is a collection of overlapping shapes that can be unioned into a single polygon (a simple one or one with holes). A compact graph representation for N, denoted $G(N)$, can be defined as follows. There is a graph node for every connected component of N on a conducting layer such as Metal, Poly etc. A node containing terminal shapes is designated as a terminal node. Two graph nodes are connected by a graph edge iff there is at least one contact or via connecting the respective components of N. Given a layer A where critical area analysis needs to be performed, the *extended graph of* N on layer A denoted as $G(N, A)$ can be obtained from $G(N)$ by expanding all components of N_A by their medial axis. For every via or contact that connects a component of N_A to a component of N_B, $B \neq A$, introduce an approximate point along the medial axis representing that via or contact, referred to as a *via-point*. In addition, introduce an edge in $G(N, A)$ connecting the via-point with the respective node of N_B. If a contact or via has been designated as terminal shape, designate also the corresponding via point as terminal. Any portion of the medial axis induced by edges of terminal shapes is also identified as terminal. For the purposes of this paper we assume that $G(N)$ can be readily available by a built in net tracing capability. Then $G(N, A)$ can be easily obtained using well known Voronoi algorithms (e.g. [9] for L_∞). Net extraction is a well known topic beyond the scope of this paper.

We use the extended net graph $G(N, A)$ to detect loops. For this purpose we partition $G(N, A)$ into *bi-*

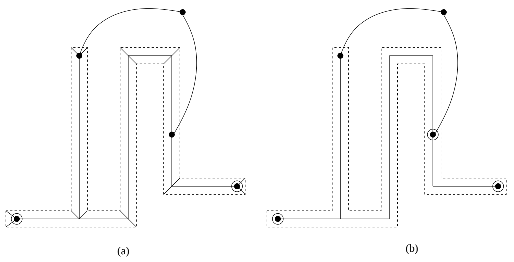


Figure 2: The net graph of Figure 1 before (a) and after (b) cleanup of trivial parts.

connected components, bridges and *articulation points* using *depth-first search* (DFS) as described in [13, 3]. For our problem we only need to keep some additional terminal information to determine whether the removal of a vertex or edge actually *breaks* $G(N, A)$ i.e., whether it disconnects $G(N, A)$ leaving terminals in both sides. Any bridges or articulation points that do not disconnect terminals are called *trivial* and can be removed form $G(N, A)$. Figure 2(b) illustrates the net graph after cleaning all trivial parts. Circles indicate terminal and articulation points.

Definition 1 *A defect D is a* minimal break *of a simple shape P if D breaks P into at least two pieces and D has minimal size, that is, if D is shrunk by $\epsilon \geq 0$ then D will be entirely contained in the interior of P. A piece of P may trivially consist of a single edge. A minimal break is called* strictly minimal *if it contains no other minimal break in its interior. A* break *is any defect totally overlapping a minimal break.[7].*

Definition 2 *A* minimal open *is a defect D that breaks a net N and D has minimal size, that is, if D is shrunk by $\epsilon > 0$ then D no longer breaks N. D breaks N if any two terminal shapes of N get disconnected or if a terminal shape itself gets destroyed. A minimal open is called* strictly minimal *if it contains no other minimal open in its interior. An* open *is any defect entirely covering a minimal open.*

Definition 3 *The center of an open D is a* generator point *for D weighted by the size (radius) of D. A generator point is of order $k, k \geq 1$, if D creates an open by breaking k polygonal paths (wires). A segment formed as a union of (kth order) generator points is called a (kth order)* generator segment.

Definition 4 *The* core *of the extended net graph $G(N, A)$ on layer A, denoted $core(N, A)$, is the set of all medial axis vertices, including articulation, via, and terminal points, and all medial axis edges except the standard 45° edges*[1]. *$G(N, A)$ is assumed to have*

[1]The 45° edges of the L_∞ medial axis bisecting axis parallel polygon edges are referred to as *standard* 45°

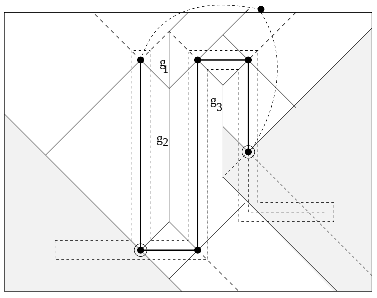


Figure 3: $V(C_i)$.

been cleaned up from any trivial components, trivial bridges, or trivial articulation points.

Lemma 1 *The 1st order generators for strictly minimal opens on layer A for net N, denoted $Gen_1(N,A)$, are the bridges, terminal edges, articulation points, and terminal points of $G(N,A) \cap core(N,A)$.*

2.1 Determining higher order generators for opens

Consider a bi-connected component of $G(N,A)$ denoted as C_i. Let $core(C_i) = C_i \cap core(N,A)$. For the purposes of this problem the endpoints and the open portion of a core segment are treated separately giving higher priority to the endpoints. Let the (weighted) Voronoi diagram of $core(C_i)$ be denoted $V(C_i)$, where any point p along a core segment s is weighted with $w(p) = d(p, e_l) = d(p, e_r)$, where e_l, e_r are the polygonal elements inducing s. Any Voronoi region equidistant from a core segment and its endpoint is assigned to the endpoint. Regions belonging to articulation or terminal points are colored red. See Figure 3.

We can identify the higher order generators for opens using higher order Voronoi diagrams of $core(C_i)$. In particular we can use $V^i(C_i), i \geq 1$, $(V^1(C_i) = V(C_i))$ to identify the $(i+1)$-order generators. It is not hard to see that a 2nd order strictly minimal open involving C_i must be generated by a point along a non-red, non-standard-45 Voronoi edge or a non-red Voronoi vertex of $V(C_i)$. The potential 2nd order generators of our example are illustrated in Figure 3 as $\{g_1, g_2, g_3\}$.

Let g be a Voronoi edge or vertex of $V(C_i)$ representing a potential opens generator. Let $core(g)$ be the set of core elements inducing g. There are three possible outcomes with respect to the connectivity of C_i after removing the elements of $core(g)$: 1) Removing $core(g)$ does not disconnect C_i; label *no-disconnect.* 2) Removing $core(g)$ disconnects the component but does not break it; label *trivial-disconnect.* 3) Removing $core(g)$ breaks the component; label *break.* If potential generator g is labeled break then g must be a 2nd order opens generator. Otherwise, g can not be a pure 2nd order generator (however portions of g might still be $k > 2$-order generators). The set of 2nd order generators determined form $V(C_i)$ by being labeled break is denoted $Gen_2(C_i)$. In figure 3 $Gen_2(C_i) = \{g_1, g_2\}$. All elements of $Gen_2(C_i)$ are colored red.

To determine $(k > 2)$-order generators (if any) we can repeat this process until all regions are colored red. In particular $V^k(C_i)$ can be obtained from $V^{k-1}(C_i)$ with an iterative process computing a slight modification of order k Voronoi diagrams. Red regions of $V^{k-1}(C_i)$ remain red in $V^k(C_i)$ and no further k-order subdivision is performed within. For every non-red region of $V^{k-1}(C_i)$ we compute the (slightly modified) k-order Voronoi subdivision as follows: Let H^{k-1} be the set of core elements owning the region of $V^{k-1}(C_i)$ under consideration $(reg(H^{k-1}))$. Let $s(H^{k-1})$ be the superset of H^{k-1} including all core segments that are incident to core points in H^{k-1}. Let $N(s(H^{k-1}))$ denote the union of core elements owning Voronoi regions incident to regions of core segments in $s(H^{k-1})$ excluding any core elements already in $s(H^{k-1})$. Compute the (weighted) L_∞ Voronoi diagram of $N(s(H^{k-1}))$ and truncate it within the interior of $reg(H^{k-1})$. This gives the kth order subdivision within $reg(H^{k-1})$. Edges of $V^{k-1}(C_i)$ whose neighboring k-order regions have the same owners get removed from $V^k(C_i)$.

Given $V^k(C_i)$, $k \geq 1$, the set of potential $(k+1)$-order generators for strictly minimal opens must be the set of non-red, non-standard-45° Voronoi edges and vertices of $V^k(C_i)$. Thus, the set $Gen_{k+1}(C_i)$ of $(k+1)$-order generators for opens can be derived using a direct generalization of the process for $k = 1$.

To determine the labeling of Voronoi edges bounding a Voronoi region $reg(H)$ of $V^k(C_i)$, we can simply remove from C_i the tuple H of core elements owning $reg(H)$, and determine new bi-connected components, bridges and articulation points. In the case of 2nd order generators $(k = 2)$ more advanced algorithmic techniques could be employed to derive a faster algorithm (see e.g. [2]). We do not attempt any such improvement in this simple version however. The time bound is $O(kn^2 + k^2 n \log n)$. In practice most biconnected components are sparse graphs due to the standard goal of minimizing wire length, in their majority just simple cycles, thus in any realistic situation k will be kept small. In case of biconnected components forming simple cycles the algorithm can easily simplify to $O(n \log n)$.

3 The Voronoi diagram for opens

Once generators for opens on a layer A are determined we can compute their Voronoi diagram and obtain a subdivision of the layout that allows fast critical area computation for opens.

Theorem 2 *Let $V(G)$ be the (weighted) L_∞ Voronoi* *diagram of the set G of all of generators for opens on layer A. The critical radius*[2] *for opens for any point t on layer A is $r_c(t) = d_w(t, s)$, where $t \in reg(s)$ in $V(G)$.*

$V(G)$ is a generalization of the Voronoi diagram for breaks and via-blocks introduced in [7]. In L_∞ it is equivalent to the weighted Voronoi diagram of additively weighted segments, where the weight function along any segment s corresponds to an L_∞ distance from a line. For a Manhattan layout, generators are axis parallel segments and points with constant weights. The Voronoi region of a generator need not be connected, thus the size of $V(G)$ need not be linear.

Let K denote the number of *interacting* pairs of generators. A generator s_i is said to be *interacting* with generator s_j if $w(s_i) < w(s_j)$ and $R(s_i)$ intersects $R(s_j)$ in a non-trivial way, where $R(s)$ is the shape representing the union of all defects generated by a generator s. The intersection $R(s_i) \cap R(s_j)$ is non-trivial if it at least encloses an entire defect generated by some point $p \in s_i$. An upper bound for K is $O(n^2)$. In practice the number of interacting higher order generators must be small and thus K must also be small. A natural upper bound on the size of $V(G)$ is $O(n + K)$, where n is the complexity of layer A. $V(G)$ can be computed in $O((n + K) \log n)$ time by plane sweep by an algorithm similar to the one presented in [8] for the Hausdorff Voronoi diagram.

4 Critical Area Computation

Once the opens Voronoi diagram is available the entire critical area integral can be easily computed as shown in [10, 9, 7]. In particular, assuming the $1/r^3$ defect distribution, the critical area integral can be discretized as a summation of simple terms derived from Voronoi edges. For any other distribution, the Voronoi diagram still allows for analytical critical area integration within regions reducing the critical area integral to a summation of formulas. See [10, 9, 7] for details.

Critical Area for general missing material defects on layer A can be obtained by combining generators for opens on layer A and generators for via-blocks on the neighboring via/contact layers for the computation of the final opens Voronoi diagram. Generators for via blocks are axis parallel weighted segments (core segments) corresponding to portions of the farthest point Voronoi diagram of via clusters (see [7]). Thus, the Voronoi subdivision for opens on layer A can be easily extended to a Voronoi subdivision for general missing material defects combining opens and via blocks. Critical area computation can then be performed in an identical manner.

[2]The critical radius at a point t is the size of the smallest defect centered at t causing an open.

Acknowledgments

I would like to thank Dr. L. F. Heng of IBM T.J. Watson Research center for helpful discussions on the definition of a net graph. The IBM Voronoi EDA group including R. Allen, S.C. Braasch, J.D. Hibbeler and M.Y. Tan are greatly acknowledged for co-developing, supporting, and expanding the Voronoi Critical Area Analysis Tool.

References

[1] A. Ferris-Prabhu. Defect size variations and their effect on the critical area of VLSI devices. *IEEE J. of Solid State Circuits*, 20(4):878–880, Aug. 1985.

[2] J. Hopcroft and R. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 1973.

[3] J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Comm. ACM*, 16(6):372–378, 1973.

[4] A. B. Kahng, B. Liu, and I. I. Mandoiu. Non-tree routing for reliability and yield improvement. *IEEE Trans. on Comp. Aided Design of Int. Circuits and Systems*, 23(1):148 – 156, 2004.

[5] I. Koren. *Yield Modeling and defect Tolerance in VLSI circuits*, "The effect of scaling on the yield of VLSI circuits", pages 91–99. Adam-Hilger Ltd., 1988.

[6] D. N. Maynard and J. D. Hibbeler. Measurement and reduction of critical area using Voronoi diagrams. In *Advanced Semiconductor Manufacturing IEEE Conference and Workshop*, 2005.

[7] E. Papadopoulou. Critical area computation for missing material defects in VLSI circuits. *IEEE Trans. on Computer-Aided Design*, 20(5):583–597, 2001.

[8] E. Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40:63–82, 2004.

[9] E. Papadopoulou and D. T. Lee. The l_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry and Applications*, 11(5):503–528, 2001.

[10] E. Papadopoulou and D. T. Lee. Critical area computation via Voronoi diagrams. *IEEE Transactions on Computer-Aided Design*, 1999.

[11] C. H. Stapper. Modeling of defects in integrated circuit photolithographic patterns. *IBM J. Res. Develop.*, 28(4):461–475, 1984.

[12] C. H. Stapper and R. J. Rosner. Integrated circuit yield management and yield analysis: Development and implementation. *IEEE Trans. Semiconductor Manufacturing*, 8(2):95–102, May 1995.

[13] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1972.

[14] I. A. Wagner and I. Koren. An interactive VLSI CAD tool for yield estimation. *IEEE Trans. on Semiconductor Manufacturing*, 8(2):130–138, 1995.

Linear Axis Computation for Polygons with Holes

Vadim Trofimov* Kira Vyatkina†

Abstract

We generalize an algorithm that constructs a linear axis of a simple polygon to the case of a polygon with holes. We show that a linear axis ε-equivalent to the medial axis can be computed from the latter in linear time for almost all polygons with holes.

1 Introduction

Skeletons have nowadays a wide range of applications in computer graphics, medical imaging, and shape retrieval.

The most widely used skeleton is the *medial axis* – a subset of a Voronoi diagram $VD(P)$ of a polygon P. More precisely, the medial axis $M(P)$ can be obtained from $VD(P)$ by restricting the latter to the interior of P and discarding its edges incident to the reflex vertices of P (see e.g. [3]).

Another well-known type of skeletons is a so-called *straight skeleton* [1] traced by the polygon's vertices during a shrinking process, when the edges move inside at constant speed. The corresponding process is also referred to as *linear wavefront propagation* (see [5]).

A recently proposed *linear axis* [5] is defined in the following way. Let $\{v_1, v_2, \ldots, v_n\}$ denote the set of reflex vertices of a polygon P, and let $k = (k_1, k_2, \ldots, k_n)$ be a sequence of non-negative integers. Replace each vertex v_i with $k_i + 1$ coinciding vertices connected by k_i zero-lengths edges called *hidden edges*; choose the directions of the hidden edges so that internal edges at all the $k_i + 1$ vertices would be equal. Denote the resulting polygon by P^k.

Definition 1 *The linear axis $L^k(P)$, corresponding to the sequence k of hidden edges, is the trace of the convex vertices of P^k during the linear wavefront propagation.*

In [5], the linear axis was defined only for simple polygons, but the above definition can be applied to polygons with holes as well (Figure 1).

The larger are the values assigned to k_i, $1 \le i \le n$, the better $L^k(P)$ approximates the medial axis $M(P)$. This observation is formalized in [5] by means of

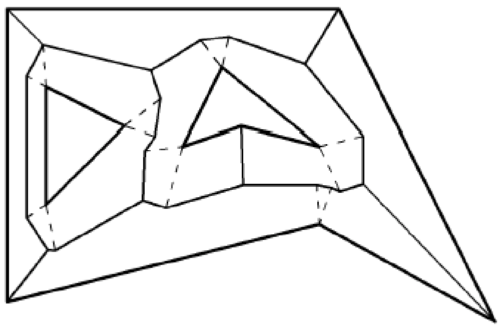


Figure 1: The linear axis of a polygon with two holes, which has exactly one hidden edge at each reflex vertex.

a notion of ε-equivalence between a linear axis and the medial axis. Moreover, for a simple polygon P, an efficient algorithm was proposed, which allowed computation of the values of k_i needed to achieve ε-equivalence for a given $\varepsilon \ge 0$, along with reconstruction of the corresponding linear axis from the medial axis in linear time – under condition that P has a constant number of "nearly co-circular" sites.

Still, criteria developed and used in [5] are inapplicable if polygons with holes are considered.

In this work, we propose a new criterion, which guarantees ε-equivalence of a linear axis and the medial axis for polygons with holes, and show how to adapt the algorithms to this case.

2 Preliminaries

The terminology introduced in this section is borrowed from [4].

Definition 2 *A* ***geometric graph*** *(V, E) is a set in $\mathbb{R}^2$ that consists of a finite set V of points, called vertices, and a finite set E of mutually disjoint, simple curves called arcs. Each arc connects two vertices of V.*

Let P be a polygon with holes. Let us denote by (V_M, E_M) the geometric graph of the medial axis $M(P)$, and by (V_{L^k}, E_{L^k}) – the geometric graph of the linear axis $L^k(P)$. Both V_M and V_{L^k} contain as a subset the set of convex vertices of P, and the vertices of degree at least three of M and L^k, respectively.

A *site* S of polygon P is either an edge or a reflex vertex of P.

*SPE "Air and Marine Electronics", hermit239@mail.ru
†Research Institute for Mathematics and Mechanics, Saint Petersburg State University, kira@meta.math.spbu.ru

Definition 3 *A Voronoi edge between node v_i generated by S_k, S_i and S_l, and node v_j generated by S_k, S_j and S_l is an **ε-edge** if $d(v_i, S_j) < (1+\varepsilon)d(v_i, S_i)$ or $d(v_j, S_i) < (1+\varepsilon)d(v_j, S_j)$.*

A Voronoi edge that is not an ε-edge is called a *non-ε-edge*. A path between two nodes of M is an *ε-path* if it consists only of ε-edges. For any node v of M, a node w is an *ε-neighbour* of v if v and w are connected by an ε-path. Let $N_\varepsilon(v)$ be the set of all ε-neighbours of v. The set $C(v) = \{v\} \cup N_\varepsilon(v)$ is called an *ε-cluster*.

Definition 4 *$M(P)$ and $L^k(P)$ are **ε-equivalent** if there exists a bijection $f : V_M \to V_{L^k}$ such that:*
1. $f(p) = p$, for all convex p of P;
2. $\forall v_i, v_j \in V_M$ with $v_j \notin N_\varepsilon(v_i)$, $\exists$ an arc in E_M connecting v_i and v_j $\Leftrightarrow$ $\exists$ an arc in E_{L^k} connecting $f(v_i')$ and $f(v_j')$, where $v_i' \in C(v_i)$ and $v_j' \in C(v_j)$.

(Note: in [4], function f was required to be surjection, but our results hold for a stronger definition of ε-equivalence – with f being bijection.)

In the following, we will say that the medial axis partitions P into *Voronoi cells*, and will denote by $VC(S)$ the Voronoi cell generated by site S. Similarly, the linear axis partitions P into *linear cells*, where $LC(S)$ denotes a linear cell generated by S.

3 Topological equivalence

In order to rule out degenerated cases, for any graph, let us interpret a node of degree $d \geq 4$ as $(p-2)$ coinciding nodes connected by $(p-3)$ edges of zero length in such a manner that the subgraph induced by these nodes is a tree (Figure 2).

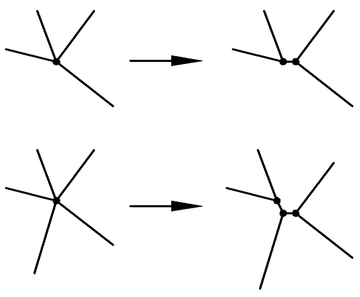


Figure 2: Interpretation of graph nodes having degrees 4 and 5, respectively .

Before we can formulate a criterion of ε-equivalence for polygons with holes, we need to introduce two new concepts.

Definition 5 *Let $(u, v) \in E_M$ be an edge incident to $VC(S_1)$ and $VC(S_2)$. A **barrier** b for (u, v) is formed by two segments connecting a point $c \in (u, v)$ with the two closest points from S_1 and S_2, respectively. The point c is a **center** of the barrier. (Figure 3.)*

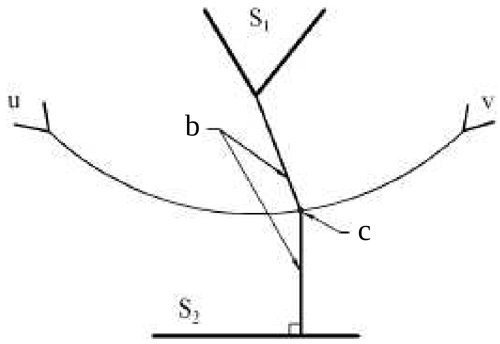


Figure 3: A barrier b for an edge (u, v) centered at point $c \in (u, v)$.

Under *intersection with a barrier* we will assume existence of more than one common point with it.

Definition 6 *An **obstacle** for an ε-cluster C is a set of barriers for all non-ε-edges having one endpoint in C and the other – in $V_M \setminus C$. (Figure 4.)*

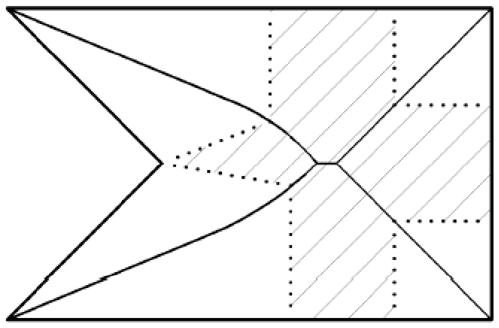


Figure 4: An obstacle for an ε-cluster consisting of two vertices.

Theorem 1 *Let P be a polygon with holes. Let $M(P)$ be the medial axis of P, and $L^k(P)$ – a linear axis of P corresponding to the sequence k of hidden edges. If for any non-ε-edge $e \in E_M$, there exists a barrier b_e: $b_e \subset LC(S_1) \cup LC(S_2)$, where $e \subset \partial(VC(S_1)) \cap \partial(VC(S_2))$, then $L^k(P)$ is ε-equivalent to $M(P)$.*

(Note: the condition for the above Theorem can be weakened. It is sufficient to require existence of barriers only for the non-ε-edges having their endpoints in different ε-clusters.)

4 Conflicting pairs

Theorem 1 implies that in order to assure ε-equivalence of the axes, is is sufficient to have for any non-ε-edge $e \in E_M$ a barrier b_e lying in the union of linear cells of the two sites, Voronoi cells of which are incident to e. We will show that instead, we may ensure existence of barriers, which would separate certain pairs of linear cells – and namely, those generated by so-called *conflicting pairs* of sites.

The notion of a conflicting pair of sites was introduced in [5]. However, we will need to redefine it.

Let us consider an edge $(u, v) \in E_M$. Suppose that its leftmost endpoint is denoted by u. (If u and v have the same x-coordinate, the choice of u does not matter.)

Definition 7 *Let $(u, v) \in E_M$ be a non-ε-edge; denote by S_1 and S_2 the two sites that generated (u, v) (that is, $(u, v) \subset \partial(VC(S_1)) \cap \partial(VC(S_2))$). Any site $S \neq S_1, S_2$, such that at least one vertex of $VC(S)$ belongs to $C(u)$, is a* ***left neighbour*** *of (u, v).*

The definition of a right neighbour of (u, v) is analogous.

Definition 8 *A* ***conflicting pair for a non-ε-edge*** *$(u, v) \in E_M$ is a pair of its left neighbour and its right neighbour, such that at least one of those neighbours is a reflex vertex.*

5 Computation of hidden edges

The framework of the algorithms will remain the same as proposed in [5].

The conflicting pairs of sites for all non-ε-edges are handled in an arbitrary order. For each pair for any such edge e, the maximal speed of the wavefront at its reflex vertex (or vertices) is bounded in order to assure existence of a barrier for e, which would separate the linear cells of the sites being considered.

A detailed analysis of the possible combinations of site types is carried out in [4]. It follows that there always exists a speed, for which one can construct such a barrier lying in the union of the two linear cells generated by the sites, Voronoi cells of which are incident to e.

To bound the speed of a reflex vertex as desired, we must insert at it a sufficient number of hidden edges.

Below we outline the algorithm. For a reflex vertex S_j, we denote by α_j the inner angle of P at S_j, and by s_j – the speed, with which S_j moves inside the polygon. For further details, the reader is referred to [5].

Algorithm ComputeHiddenEdges (P, ε)

Input: A polygon P and a real constant $\varepsilon \geq 0$.
Output: The number of hidden edges for each reflex vertex such that the resulting linear axis is ε-equivalent to the medial axis.

```
1. Compute the medial axis $M$ of $P$
2. For each reflex vertex $S_j$ of $P$:
       if $\alpha_j \geq 3\pi/2$ then $s_j = \frac{1}{\cos((\alpha_j - \pi)/4)}$
       else $s_j = \frac{1}{\cos((\alpha_j - \pi)/2)}$
3. ComputeConflictingSites $(\varepsilon)$
4. For each non-$\varepsilon$-edge $w$,
       for each pair of conflicting sites $S_i$, $S_j$:
           HandleConflictingPair$(w, S_i, S_j)$
5. For each reflex vertex $S_j$ of $P$:
       $k_j = \lceil \frac{\alpha_j - \pi}{2\cos^{-1}(1/s_j)} \rceil$.
```

ComputeConflictingSites(ε) determines all pairs of conflicting sites for all non-ε-edges from E_M. For any such edge (u, v), ε-clusters $C(u)$ and $C(v)$ are computed, and the right and the left neighbours of (u, v) are retrieved. The pairs consisting of a left and a right neighbour, one of those being a reflex vertex, are conflicting.

HandleConflictingPair(w, S_i, S_j) examines positions of S_i and S_j, where (S_i, S_j) is a conflicting pair for w, and of the sites – generators of the Voronoi cells incident to w (let us denote these sites by S_k and S_l). As a result, it bounds the speed s_j of any reflex vertex S_j so that there would exist a barrier on w lying in $LC(S_k) \cup LC(S_l)$ – in a similar manner as it was done in [4].

6 Correctness of the algorithm

First of all, we state that for a *non-conflicting* pair (S_i, S_j) formed by a left and a right neighbor of a non-ε-edge e, there exists a barrier on e separating $LC(S_i)$ and $LC(S_j)$.

Lemma 2 *Let two edges S_i and S_j be a left and a right neighbor of a non-ε-edge e, and let b_e be a barrier centered at an arbitrary point of e. Then both $LC(S_i)$ and $LC(S_j)$ do not intersect b_e.*

Because of presence of holes in a polygon, we need to introduce a more elaborated classification of intersections of a barrier with cells.

Definition 9 *Let us say that cell $C(S)$ intersects a barrier b on the edge (u, v)* ***from the left***, *if the part of $C(S)$, which lies on the same side from b as u, contains S.*

The definition of an intersection from the right is analogous (Figure 5).

Note that an intersection cannot be simultaneously *from the left* and *from the right.*

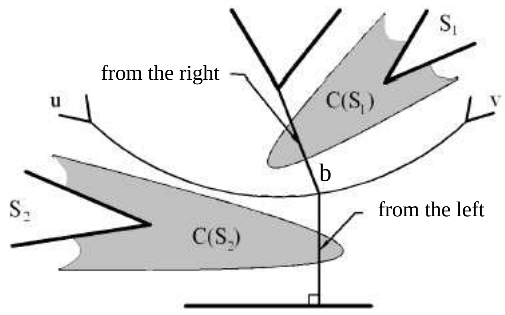


Figure 5: Cell $C(S_1)$ intersects barrier b from the right, and cell $C(S_2)$ intersects b from the left.

Lemma 3 *For any non-ε-edge $e \in E_M$, there exists a barrier b_e on e, which the cells generated by the left neighbours of e do not intersect from the left, and the cells generated by the right neighbours of e do not intersect from the right.*

Lemma 4 *For any non-ε-edge $e \in E_M$, the barrier, which satisfies conditions of Lemma 3, does not intersect the linear cell of any site $S \neq S_1, S_2$, where $e \subset \partial(VC(S_1)) \cap \partial(VC(S_2)$.*

Along with Theorem 1, this proves correctness of the algorithm.

Theorem 5 *Let P be an arbitrary polygon with holes, and $\varepsilon \geq 0$ – a real constant. Let k denote the sequence of hidden edges built by the algorithm ComputeHiddenEdges(P, ε). Then the linear axis $L^k(P)$ is ε-equivalent to the medial axis $M(P)$.*

Thus, given a polygon P with holes, its medial axis $M(P)$ and a real $\varepsilon \geq 0$, one can compute the linear axis ε-equivalent to $M(P)$ in linear time, provided that all ε-clusters have constant size. All the details remain identical to those described in [5] for the case of a simple polygon.

7 Conclusion

We generalized an algorithm for computation of a linear axis ε-equivalent to the medial axis, initially proposed for a simple polygon [5], to the case of polygons with holes. If the polygon has a constant number of "nearly co-circular" sites, the computation of a linear axis from the medial axis will require only linear time. As the medial axis of a polygon with holes can itself be computed in $O(n \log n)$ time [2], this results in $O(n \log n)$ total time.

Acknowledgment

The second author was supported by Human Capital Foundation.

References

[1] O. Aichholzer, F. Aurenhammer, D. Alberts, B. Gärtner. *A novel type of skeleton for polygons.* The Journal of Universal Computer Science, 1:752-761, 1995.

[2] D. G. Kirkpatrick. *Efficient computation of continuous skeletons.* Proc. 20th IEEE Annual Symp. on Foundations of Comput. Sci., pp. 18-27 (Oct.1979).

[3] F. P. Preparata, M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[4] M. Tănase. *Shape Decomposition and Retrieval.* Ph.D. Thesis, Utrecht University, 2005.

[5] M. Tănase, R. C. Veltkamp. *Straight skeleton approximating the medial axis.* Proc. 12th Annual European Symposium on Algorithms, pp. 809-821, 2004.

Data-Powered Geometric Computing

Bernard Chazelle

Professor of Computer Science
Princeton University,
Department of Computer Science
35 Olden Street, Princeton, NJ 08540, USA
chazelle@cs.princeton.edu

Advances in data acquisition technology – together with the imminent demise of Moore's Law – are prompting a rethink of basic algorithm design principles. I will discuss a few examples of this phenomenon in the context of low-entropy computation and dimension reduction.

Exact and efficient computations on circles in CGAL (abstract) *[†]

Pedro M. M. de Castro[‡] Sylvain Pion Monique Teillaud[§]

Abstract

CGAL *(Computational Geometry Algorithms Library)* is a large collection of geometric objects, data structures and algorithms. CGAL currently offers functionalities mostly for linear objects (points, segments, lines, triangles...).

The first version of a kernel for circles and circular arcs in 2D was recently released in CGAL 3.2. We show in this paper a variety of techniques that we tested to improve the efficiency of the 2D circular kernel. These improvements are motivated by applications to VLSI design, and real VLSI industrial data are used to measure the impact of the techniques used to enhance this kernel. The improvements will be integrated in CGAL 3.3.

1 Introduction

The goal of the CGAL Open Source Project is to provide easy access to efficient and reliable geometric algorithms to users in industry and academia. This is achieved in the form of the C++ *Computational Geometry Algorithms Library* [2]. The CGAL packages adopt the generic programming paradigm [12], making extensive use of C++ class templates and function templates, and their design is heavily inspired from the C++ Standard Template Library [5].

For instance, let us consider the case of geometric arrangements: an *arrangement* of a finite set of curves in the plane is the partition of the plane into faces, edges and vertices, that is induced by these curves [15, 3, 11]. A generic implementation of a data structure that handles arrangements is achieved by the `CGAL::Arrangement_2` class:
`template <class Traits> class Arrangement_2`
This class must be instantiated with a class, referred to as a *traits* class [19], that must define a type representing a certain family of curves, and some functions operating on curves of this family.

The CGAL *kernels* provide the users with basic geometric objects and basic functionalities on them. CGAL currently offers kernels for linear objects (points, segments, lines, triangles...), and the first version of a kernel for circles and circular arcs in 2D, called *2D circular kernel* in the sequel, was recently released in CGAL 3.2 [20]. A kernel can be wrapped into a traits class offering the interface for some geometric algorithms; this was done for the CGAL circular kernel and `CGAL::Arrangement_2`. However, a kernel is meant to offer general purpose functionalities, while a traits class offers the minimum set of functionalities required by a specific class.

Robustness, achieved through the exact geometric computation paradigm [23], is probably the first strength of CGAL. The CGAL arrangement package, completely redesigned for CGAL 3.2 [22, 21] offers a robust and efficient implementation. Other libraries like ESOLID [18] may crash on degenerate input data[1]. Real VLSI data sets consist of line segments and circular arcs, containing many degenerate, or close to degenerate, cases (junctions, identical intersection points, tangencies...) requiring highly robust code. Typically, the question is to compute boolean operations on these data, that can be easily performed on top of the computation of the input curves arrangement [13]. Efficiency is also a crucial quality for the use of CGAL on real industrial data. VLSI inputs consist of very large data sets, which leads to the need for improvements in the efficiency of the 2D circular kernel.

We show in this paper a variety of techniques from different nature that we tested to improve the 2D circular kernel, and experimental evidence of their impact is studied by benchmarking on real VLSI industrial data. The techniques resulting in measurable improvements will be integrated in CGAL 3.3.

2 The 2D Circular Kernel

To answer the need for robustness on manipulations of circular objects, a first version of the 2D circular kernel was released in CGAL 3.2.

*The full version of this work is available at `https://hal.inria.fr/inria-00123259/en/` as INRIA research report RR-6091 .

[†]This work is partially supported by the IST Programme of the 6th Framework Programme of the EU as a STREP (FET Open Scheme) Project under Contract No IST-006413 - ACS (Algorithms for Complex Shapes with certified topology and numerics) - `http://acs.cs.rug.nl/`

[‡]current address: Center of Computer Sciences, Universidade Federal de Pernambuco, Brazil. `pmmc@cin.ufpe.br`

[§]INRIA, BP93, 06902 Sophia Antipolis cedex, France, `{Sylvain.Pion,Monique.Teillaud}@sophia.inria.fr`, `http://www-sop.inria.fr/geometrica/team/`

[1]see `http://research.cs.tamu.edu/keyser/geom/esolid/`

The design of the 2D circular kernel [10] uses the extensibility and adaptability properties of the CGAL linear kernel [16]. The circular kernel is parameterized by, and inherits from, a `LinearKernel` parameter, for objects like points, circles and number types. It has a second parameter, `AlgebraicKernel`, providing algebraic operations that are necessary for computations on circles. The geometric level interface provides types already defined by the linear kernel, plus three new types: `Circular_arc_point_2` for points on circles, and `Circular_arc_2` and `Line_arc_2` respectively representing circular arcs and line segments delimited by such points.

Intersections involving circles lead to manipulating algebraic numbers of degree two on this ring. Moreover, most geometric predicates on circular arcs are expressed as comparisons involving such roots. We rely on exact handling using polynomial representation of these roots and algebraic methods (like resultants and Descarte's rule of sign) which reduce the computations to comparisons of polynomial expressions [8, 17, 9]. It was shown that the latter choice, for this specific small degree two, was more efficient than using a general library like CORE or LEDA.A template class `Root_of_2<RT>` is provided for algebraic numbers of degree 2, using the following internal representation: three coefficients of type `RT` specifying the polynomial of degree 2, plus one boolean value specifying whether the smaller of the roots is considered, or the other root.

The CGAL arrangement package comes with a default traits class for line segments and circular arcs, called `Def_traits` in the sequel. We wrapped the circular kernel into a traits class offering the same interface. Since the arrangement package requires the traits to provide a unique type `Curve_2`, the default traits class does not offer separate types for line segments and circular arcs. The circular kernel, meant to be general purpose, offers two different types, `Circular_arc_2` and `Line_arc_2`. The traits class built on the circular kernel uses the `boost::variant` class[2] [1], that allows to wrap the two complex types in one unique type `Curve_2`. Moreover, since arrangements algorithms implemented in this package start by cutting all curves into x-monotone arcs, functionalities like intersection computations are provided only for x-monotone arcs in a traits class for arrangements. The circular kernel implements these functionalities for general arcs.

3 VLSI Data Sets and Conditions of Experiments

We conducted experiments to evaluate the practical influence of several improvement techniques on the CGAL 2D circular kernel. Our experiments consist in computing arrangements of real industrial data of VLSI models[3] representing electronic circuits, with the efficient sweep-line algorithm provided by the new CGAL arrangement package [22].

Input	CGAL 3.2	`Def_traits`	CGAL 3.3
vlsi_1	28.0	8.55	4.61
vlsi_2	48.0	2.59	1.31
vlsi_3	135	26.7	21.8
vlsi_4	569	26.9	25.4
vlsi_5	125	14.3	14.8
vlsi_6	611	137	134
vlsi_7	690	192	169
vlsi_8	3,650	220	136
vlsi_9	2,320	581	492
Very dense	335	77.9	76.2
Sparse	0.91	0.51	0.21

Table 1: Time, in seconds, spent to compute the arrangement with the CGAL 3.2 circular kernel, with `Def_traits`, and after the improvements presented in this paper.

The VLSI data have many cases of junctions, degeneracies and tangencies, causing approximate computation to fail due to rounding errors, and which make them appropriate for exact computation. See Figure 1 for an illustration. The table gives the sizes of the input files we are using for the experiments, together with the sizes of the corresponding arrangements.

We complete our experiments with synthetic input (see also Figure 1): a very dense distribution where circles are centered on a grid and all pairs of circles intersect (a), and a sparse distribution without intersection (b).

The hardware of our experiments was a Pentium 4 at 2.5 GHz with 1GB of memory, running Linux (2.4.20 Kernel). The compiler was g++4.0.2; all configurations were compiled with the -DNDEBUG -O2 flags.

In the sequel, the default traits class `Def_traits` will be used as a reference for measuring the performances of the circular kernel. Both of them are used with the same basic exact ring number type: `CGAL::MP_Float`. Note however that the default traits class is already optimized and uses arithmetic filtering (see Section 4.3.2) which makes it quite efficient. The CGAL 3.2 circular kernel is not yet filtered at all. Note also that the use of `boost::variant` (see Section 2) introduces a slight overhead in the running times obtained by the circular kernel.

These elements partly explain the poor performance of this kernel shown in Table 1. The rest of this work is devoted to show how several techniques can be combined to produce an important improvement in the running times.

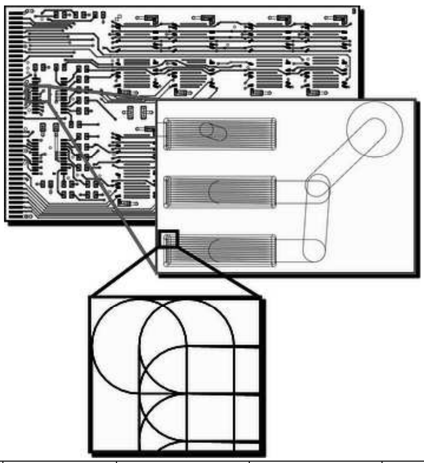
[2]`http://www.boost.org/libs/variant/index.html`

[3]Thanks to MANIA BARCO and GEOMETRYFACTORY.

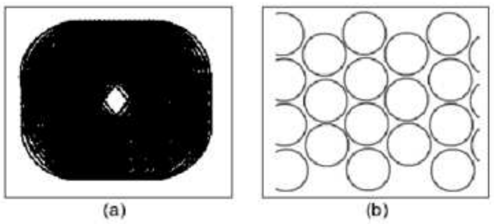
4 Tuning the 2D Circular Kernel

4.1 Caching Using Bit-Field

The results of some costly operations can be stored to avoid recomputing them several times. When those operations return symbolic values, like *boolean*, the memory space used can be very low: the results can be efficiently stored in a *bit-field* which consists in the manipulation of individual bits of an entire block of data [4].



VLSI	N	V	E	F
vlsi_1	11,929	20,649	26,468	6,385
vlsi_2	9,663	8,556	9,439	887
vlsi_3	35,449	101,758	163,316	61,887
vlsi_4	12,937	81,096	143,049	61,986
vlsi_5	4,063	40,636	77,598	36,965
vlsi_6	74,052	547,250	1,016,460	470,480
vlsi_7	89,918	495,209	878,799	383,871
vlsi_8	123,175	370,304	555,412	190,031
vlsi_9	139,908	1,433,248	2,690,530	1,257,684



Distributions	N	V	E	F
very dense (a)	400	160,400	320,000	159,602
sparse (b)	441	882	882	442

Figure 1: An example of VLSI data (*vlsi_7*), composed of 22,406 polygons and 294 circles, for a total number of 89,918 arcs. The zooms show the complexity of the data. The bottom picture shows the two synthetic data sets. The tables give the characteristics of the data sets: N is the number of arcs (line segments or circular arcs), V is the number of vertices of the arrangement, E is the number of edges and F the number of faces.

We introduce a bit-field as an additional data member of `Circular_arc_2`, to store whether an arc is x-monotone, the complement of an x-monotone arc (same for y), and whether its endpoints are on the upper part of the supporting circle (same with left part). The bit-field has 2 bytes (in fact, only 12 from the total 16 bits are used).

The bit-field can be quickly computed when arcs are computed by cutting previous arcs into monotone arcs.

4.2 Caching Intersections of Supporting Circles

We tried to use a `std::map` which takes a pair of circles and returns their (at most two) intersection points. Experiments on VLSI files showed that this was not an interesting contribution.

4.3 Enhancing the Algebraic Number Type

The `Root_of_2` number type was improved by following two directions.

4.3.1 Optimizing Particular Cases

Every time when the rationality of a `Root_of_2` is detectable in constant time, we use a specific constructor that will allow to take advantage of this property. Those cases appear in the intersection of two tangent circles, in the intersection of a line and a circle that are tangent, in the intersection of a vertical/horizontal line with a circle.

4.3.2 Arithmetic Filtering

The general idea of *filtering* comparisons consists in computing an approximate result, together with an error bound. If the error bound is small enough, comparing the approximate values is enough to give the result of the comparison of the exact values, which allows to conclude very quickly. Otherwise, we say that the filter *fails*, and the computation is done using exact arithmetic. Clearly, the errors need to be kept as small as possible to avoid too many filter failures, since expensive exact computation must be used in that case, and the computations of error bound only induce an overhead [6]. The combination of exact computation and filtering techniques allows to get both fast and exact results.

4.4 Reference Counting

After the previous improvements, we profiled the circular kernel using GPROF[4] and VALGRIND[5] and we discovered that around 15% of the whole running time

[4]`http://www.gnu.org/software/binutils/manual/gprof-2.9.1`
[5]`http://valgrind.kde.org/`

was spent on calling the `CGAL::MP_Float` copy constructor. A good option to handle this copy construction bottleneck is to use a *reference counting* technique [7].

4.5 Different Algebraic Number Type Representation

In spite of the overall good performance obtained with the previous improvements, high execution times are obtained on the *vlsi_8* data set, which is a show-stopper for the use of the circular kernel on industrial data.

A number of type `Root_of_2` is a root of a polynomial ax^2+bx+c and is represented by the three coefficients a, b, c of a ring type `RT` and a boolean (see Section 2). The basic computations on such an algebraic number reduce to manipulations of the coefficients. When `RT` is a multi-precision number type (which is necessary to achieve exact computations), the time complexity of these manipulations grows with the length of the numbers. The length increases when computations are cascaded.

Storing α, β, γ as numbers of a field type `FT`, such that $\alpha + \beta\sqrt{\gamma}$ is the root, allows to reduce the lengths of multi-precision numbers.

4.6 Geometric Filtering

We mentioned interval arithmetic filtering techniques in Section 4.3.2. A similar scheme can be applied at the geometric level [14] for filtering predicates: fast approximate computation is done first; most of the time, the error bounds allow to certify the result; in bad cases, the filter fails and the result is computed exactly. Using axis-aligned bounding boxes as enclosing shapes is very appropriate for our application.

References

[1] Boost, C++ libraries. http://www.boost.org.

[2] Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

[3] Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[4] Paul Anderson and Gail Anderson. *Navigating C++ and Object-Oriented Design.* Prentice Hall, 2003.

[5] Matthew H. Austern. *Generic Programming and the STL.* Addison Wesley, 1998.

[6] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109:25–47, 2001.

[7] George E. Collins. A method for overlapping and erasure of lists. *Communications of the ACM*, 3(12):655–657, December 1960.

[8] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. *Comput. Geom. Theory Appl.*, 22:119–142, 2002.

[9] I. Emiris and E. P. Tsigaridas. Computing with real algebraic numbers of small degree. In *Proc. 12th European Symposium on Algorithms, LNCS 3221*, pages 652–663. Springer-Verlag, 2004.

[10] Ioannis Z. Emiris, Athanasios Kakargias, Sylvain Pion, Monique Teillaud, and Elias P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 438–446, 2004.

[11] Efi Fogel, Dan Halperin, Lutz Kettner, Monique Teillaud, Ron Wein, and Nicola Wolpert. Arrangements. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces.* Springer-Verlag, Mathematics and Visualization, 2006.

[12] Efi Fogel and Monique Teillaud. Generic programming and the CGAL library. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces.* Springer-Verlag, Mathematics and Visualization, 2006.

[13] Efi Fogel, Ron Wein, Baruch Zukerman, and Dan Halperin. 2D regularized boolean set-operations. In CGAL Editorial Board, editor, *CGAL-3.2 User and Reference Manual.* 2006.

[14] Stefan Funke and Kurt Mehlhorn. Look: A lazy object-oriented kernel for geometric computation. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 156–165, 2000.

[15] D. Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[16] Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Michael Seel. An adaptable and extensible geometry kernel. *Computational Geometry: Theory and Applications*, To appear. Special issue on CGAL.

[17] Menelaos I. Karavelas and Ioannis Z. Emiris. Root comparison techniques applied to computing the additively weighted Voronoi diagram. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 320–329, 2003.

[18] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. ESOLID - a system for exact boundary evaluation. *Computer-Aided Design*, 26(2):175–193, 2004.

[19] N.C. Myers. Traits: a new and useful template technique. *C++ Report*, June 1995.

[20] Sylvain Pion and Monique Teillaud. 2D circular kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual.* 3.2 edition, 2006.

[21] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. 2D arrangements. In CGAL Editorial Board, editor, *CGAL User and Reference Manual.* 3.2 edition, 2006.

[22] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Computational Geometry: Theory and Applications*, To appear. Special issue on CGAL.

[23] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.

Sweeping and Maintaining Two-Dimensional Arrangements on Surfaces*

Eric Berberich† Efi Fogel‡ Dan Halperin‡ Ron Wein‡

Abstract

We introduce a general framework for processing a set of curves defined on a continuous two-dimensional parametric surface, while sweeping the parameter space. A major goal of our work is to maximize code reuse in implementing algorithms that employ the prevalent sweep-line paradigm, and consequently to minimize the effort needed to extend the implementation of the paradigm to various surfaces and families of curves embedded on them. We show how the sweep-line paradigm is used to construct an arrangement of curves embedded on an orientable parametric surface, and explain how the arrangement package of CGAL, which previously handled only arrangements of bounded planar curves, is extended to handle curves embedded on a general surface. To the best of our knowledge, this is the first software implementation of generic algorithms that can handle arrangements on general parametric surfaces.

1 Introduction

We are given a surface S in $\mathbb{R}^3$ and a set $\mathcal{C}$ of curves that all lie on this surface. The *arrangement* of the curves of $\mathcal{C}$, denoted $\mathcal{A}(\mathcal{C})$ is the subdivision these curves induce on the surface S into cells of dimension 0 (*vertices*), 1 (*edges*) and 2 (*faces*).

CGAL, the Computational Geometry Algorithms Library,[1] is the product of a collaborative effort of several sites in Europe and Israel, aiming to provide a generic and robust, yet efficient, implementation of widely used geometric data structures and algorithms. The arrangement package [9] included in the latest public release of CGAL (Version 3.2) is capable of constructing and maintaining planar arrangements of bounded curves. That is, the surface S is the xy-plane, and all curves in $\mathcal{C}$ are bounded. When working with unbounded curves, users are required to properly clip them as a preprocessing step, so that no essential information about the arrangements (e.g., a finite intersection point) is lost. However, this solution is insufficient for some applications. For example, it is possible to represent the minimization diagram of a set of surfaces in $\mathbb{R}^3$ as a planar arrangement, where each face is labeled with the surface that induces the lower envelope over that face [8]. As an arrangement of bounded curves has only a single unbounded face, it is impossible to represent the minimization diagram of a set of unbounded surfaces, where several unbounded faces might be required.

We have recently enhanced the arrangement package to support planar arrangements of unbounded curves. This extension will be included in the forthcoming release of CGAL (Version 3.3). The same principles we used for handling unbounded curves, or more precisely curve-ends that lie at infinity, can be nicely generalized for the case of a set of curves embedded on a parametric surface. An orientable *parametric surface* S is a surface defined by parametric equations involving two parameters u and v, namely: $f_S(u,v) = (x(u,v),\, y(u,v),\, z(u,v))$. Thus, $f_S : \mathbb{P} \longrightarrow \mathbb{R}^3$ and $S = f_S(\mathbb{P})$, where $\mathbb{P}$ is a continuous and simply connected two-dimensional parameter space. The general case is currently implemented as a prototypical package in CGAL.

Related work: Effective algorithms for manipulating arrangements of curves have been a topic of considerable interest in recent years, with an emphasis on exactness and efficiency of implementation [5]. Mehlhorn and Seel [7] propose a general framework for extending the sweep-line algorithm to handle unbounded curves. Note that they do not address the case of surfaces other than the plane. Andrade and Stolfi [1] develop exact algorithms for manipulating circular arcs on a sphere. Cazals and Loriot [4] compute exact arrangements of circles on a sphere. Halperin and Shelton [6] incrementally construct arrangements of circles on a sphere, using floating-point arithmetic and assuming general position. The latter two works are motivated by molecular modeling.

2 Sweeping on Surfaces

Recall that the main idea behind the Bentley-Ottmann sweep-line algorithm [2] is to sweep a vertical line starting from $x = -\infty$ onward and maintain the set of x-monotone curves that intersect it. These curves are ordered according to the y-coordinate of their intersection with the vertical line and stored in a balanced search tree named the *status structure*. The

*This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes), by the Israel Science Foundation (grant no. 236/06), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

†Max-Planck-Institut für Informatik, Saarbrücken, Germany, `eric@mpi-inf.mpg.de` .

‡School of Computer Science, Tel-Aviv University, Israel, `{efif,danha,wein}@post.tau.ac.il` .

[1] `http://www.cgal.org/`.

contents of the status line change only at a finite number of *event points*, where an event point may correspond to a curve endpoint or to an intersection of two curves. The event points are sorted in ascending xy-lexicographic order and stored in an *event queue*. This event queue is initialized with all curve endpoints, and is updated dynamically during the sweep process as new intersection points are discovered.

2.1 Augmenting the Parameter Space

Our goal is to study the subdivision induced on a parametric surface by sweeping over its parameter space. However, to conveniently do so, we must consider a subspace of $\mathbb{P}$. Sweeping over the entire parameter space raises, in general, several difficulties either when the parameter space is unbounded, or when there is no inverse mapping from the surface to the parameter space. We eliminate these difficulties by cutting out portions of the parameter space and symbolically keeping track of these modifications.

We next formally define three aspects that require special attention when generalizing the sweep procedure. In all cases, S is a parametric surface defined over $\mathbb{P}$ in the uv-plane. We give the definitions using the u-parameter; the definitions with respect to the v-parameter are similar.

Definition 1 (Infinite boundary:) *Let $\hat{u}$ be one of the values defining the u-range of $\mathbb{P}$ ($\hat{u}$ may be finite or $\hat{u} = \pm\infty$). We say that the surface has an* infinite boundary *in u if: $\forall v \ \lim_{u \to \hat{u}} f_S(u, v) = \pm\infty$.*

Definition 2 (Curve of discontinuity:) *If u is defined over a bounded parameter range $[u_{\min}, u_{\max})$ such that: $\forall v \ \lim_{u \to u_{\max}} f_S(u, v) = f_S(u_{\min}, v)$, then the curve defined by $f_S(u_{\min}, v)$ forms a* curve of discontinuity *in u on the surface S.*

Definition 3 (Singularity point:) *A point $p_0 = f_S(u_0, v_0) \in S$ is a* singularity point *in u if u_0 is either $u_{\min}$ or $u_{\max}$, and for each $\delta > 0$ we have: $\forall v \ \exists u \ \|f_S(u, v) - p_0\| < \delta$.*

The xy-plane (see Fig. 1(a)), for example, has an infinite boundary in the minimal and the maximal values of u *and* in the minimal and maximal values of v. A canonical 3D cylinder of radius r (see Fig. 1(b)), parameterized for $\mathbb{P} = [-\pi, \pi) \times \mathbb{R}$ such that $f_S(u, v) = (r \cos u, r \sin u, v)$, contains a line of discontinuity that is parallel to the z-axis and passes through $(-r, 0, 0)$. The unit sphere (see Fig. 1(c)), parameterized over $\mathbb{P} = [-\pi, \pi) \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ using $f_S(u, v) = (\cos u \sin v, \sin u \sin v, \cos v)$, contains a semicircle of discontinuity that connects the two poles $(0, 0, -1)$ and $(0, 0, 1)$ through $(-1, 0, 0)$. In addition, the two poles are singularity points in v.

Given a surface containing curves of discontinuity and singularity points we modify the parameter space as follows: In case of discontinuity in u, we consider the open u-range $(u_{\min} + \varepsilon, u_{\max} - \varepsilon)$ for an infinitesimally small $\varepsilon > 0$. In case of a singularity point in $u_{\min}$ we augment the u-parameter range to be lower bounded by $u_{\min} + \varepsilon$ (or upper bounded by $u_{\max} - \varepsilon$ in case of a singularity point in $u_{\max}$), for an infinitesimally small $\varepsilon > 0$. We handle singularities in v in a similar fashion. As a result, we obtain an augmented parameter space $\tilde{\mathbb{P}}$, for which it is possible to define the inverse mapping $f_S^{-1} : \mathbb{R}^3 \longrightarrow \tilde{\mathbb{P}}$.

It is now possible to apply an augmented sweep-line algorithm to our parametric surface, where we actually perform a plane sweep over $\tilde{\mathbb{P}}$. Let $\tilde{S}$ denote the image of the augmented parameter space, namely $f_S(\tilde{\mathbb{P}})$. Given a set $\mathcal{C}$ of curves defined on S, we start by computing $C' = C \cap \tilde{S}$ for each $C \in \mathcal{C}$, and by subdividing C' into u-monotone subcurves. We refer to the resulting subcurves as *sweepable curves*. Note that in particular, the interior of a sweepable curve cannot intersect a curve of discontinuity or contain a singularity point. However, the curve-ends may be incident to the modified surface boundaries.

We start the sweep with the curve $f_S(u_0, v)$, for some initial fixed u-value u_0 (e.g., $u_0 = u_{\min} + \varepsilon$ in the example of the cylinder). We now sweep the curve over the surface $\tilde{S}$. For each u-value u', a subset of the sweepable curves induced by $\mathcal{C}$ intersect the sweep-curve $f_S(u', v)$, at the points $p_1, \ldots, p_k \in S$. The status structure stores these curves ordered in ascending v of $f_S^{-1}(p_1), \ldots, f_S^{-1}(p_k)$. Similarly, when we detect an intersection point p, we insert it into the event queue, considering the lexicographic uv-order of $f_S^{-1}(p)$. The event queue must contain events associated with curve-ends incident to the surface boundaries for its proper maintenance.

2.2 Sweeping Unbounded Curves

The main difficulty in adapting the Bentley-Ottmann sweep algorithm [2] to the unbounded case, lies in handling its initialization step, and symmetrically in completing the sweep after all the finite event points were encountered. Let us revise the terminology used so far. Instead of considering the endpoints of a curve, we refer to the two *curve-ends*. A curve-end may be unbounded or bounded, and only in the latter case we have a valid endpoint. In order to perform the sweep-line procedure, we require the two following comparisons involving unbounded curve-ends (in addition to the operations listed in [5] for bounded curves): (i) determine the relative vertical position of two curve-ends defined at $x = \pm\infty$,[2] and (ii) determine the relative horizontal positions of two curve-ends with finite x-coordinates that lie at $y = \pm\infty$.

[2]For two lines this amounts to comparing their slopes, and in case of equality we can compare their vertical position at $x = 0$. Other curves may require more careful analysis.

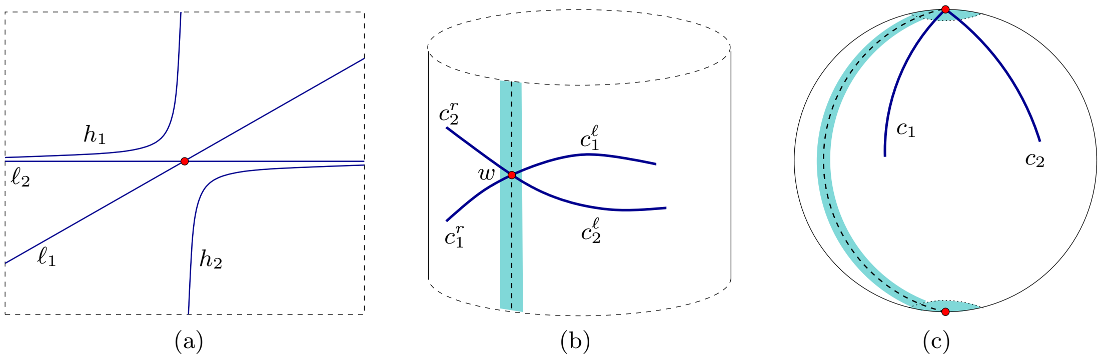


Figure 1: Comparing curve-ends with boundary conditions: (a) Comparing at infinity. (b) Comparing near the line of discontinuity. (c) Comparing near a singularity point.

Having defined the geometric primitives, we are ready to modify the sweep-line algorithm to handle infinite curves. First, we store extra information with the events: an event may be associated with a (finite) point, or it may be associated with an unbounded curve-end at $x = \pm\infty$ or at $y = \pm\infty$. We begin the sweep process by constructing events that represent all unbounded and bounded curve-ends. To sort these events we use a simple procedure based on the two primitive comparison operations listed above: if one event lies at $x = -\infty$ and the other is a (finite) point, then the first event is obviously smaller; if both events lie at $x = -\infty$ we compare their associated curve-ends there, etc. For instance, in Fig. 1(a) we have $\ell_1 < \ell_2 < h_1$ when the sweep is initialized. We omit further details of the process in this abstract, and remark that whenever infinity in x or in y is involved, barely any geometric operations are required.

2.3 Sweeping on General Surfaces

We can now generalize the sweep-line procedure for sweeping over curves embedded on a surface in $\mathbb{R}^3$. So far we swept over the parameter space $\mathbb{P} = \mathbb{R}^2$, and treated curve-ends that coincide with the infinite boundaries symbolically. We can use the same set of geometric primitives for sweeping over a set of curves on a surface. However, we have to re-interpret the geometric predicates as if they are given on the uv-plane. For instance, instead of comparing two points p_1 and p_2 by their xy-lexicographic order, we compare $f_S^{-1}(p_1)$ and $f_S^{-1}(p_2)$ according to their uv-lexicographic order in $\tilde{\mathbb{P}}$.

A sweepable curve-end may have *boundary conditions*. In the previous subsection we have already encountered curves with unbounded ends, and we say that the boundary condition of such an end in x (or in y) is of type *minus infinity* or *plus infinity*. In the general case, we may also encounter curve-ends whose boundary condition is *leaving discontinuity* (or *approaching discontinuity*), or *leaving singularity* (or *approaching singularity*). For instance, in the example depicted in Fig. 1(b), all sweepable curve-ends may start right after the line of discontinuity or may end right before this line, as we have removed the line of discontinuity from $\tilde{S}$. The two curves C_1 and C_2 are split at the line of discontinuity, forming the sweepable curves c_1^ℓ, c_1^r and c_2^ℓ, c_2^r, respectively. Yet when we compare the curve-ends we consider an ε-neighborhood around the line of discontinuity (shaded). Thus, c_1^ℓ is above c_2^ℓ after the line of discontinuity, when the sweep starts. Fig. 1(c) shows how we symbolically handle curve-ends that are incident to a singularity point (the north pole of a sphere in this case): c_1 lies to the left of c_2, as we compare the ends of sweepable curves in an ε-neighborhood below the north pole (shaded). Note that this means that we have a different event for every curve-end that coincides with a pole.

3 Constructing Arrangements on Surfaces

Constructing an arrangement of curves on a parametric surface boils down to properly handling the subcurves the sweep-line procedure detects and inserting them into the doubly-connected edge-list (DCEL for short) that represents the arrangement; see, e.g., [3, Chap. 2]. As the only modification of the sweep-line algorithm involves curve-ends with boundary conditions, we have to augment the curve-insertion procedures to properly handle such curve-ends.

Already when moving to unbounded curves we should consider a representation of the arrangement that caters for more than one unbounded face. Fig. 2(a) demonstrates one possibility, where we use an implicit bounding rectangle embedded in the DCEL structure using *fictitious* edges that are not associated with any concrete planar curve. It is also possible to choose a different representation of a planar arrangement of bounded curves that uses a single vertex at infinity v_{inf}, such that all unbounded curve-ends are incident to this vertex; see illustration in Fig. 2(b).

Aiming for modularity, we wish to decouple the implementation of the basic arrangement operations (e.g., inserting a new edge associated with a subcurve, removing an edge, etc.)

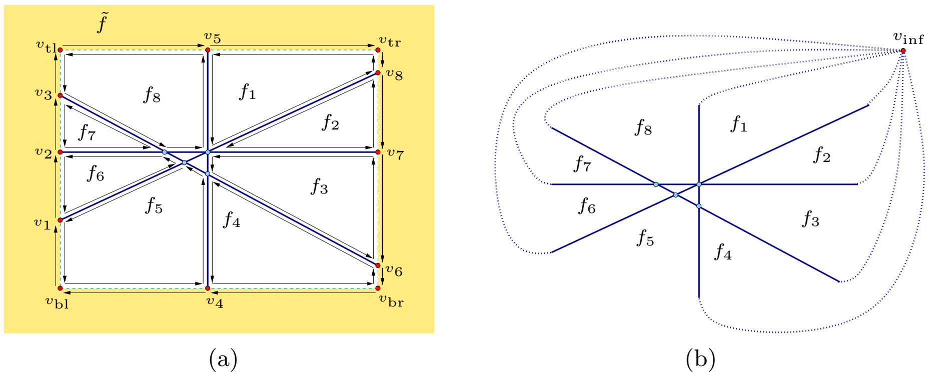


Figure 2: Possible DCEL representations of an arrangement of four lines in the plane.

from the actual representation of the arrangement. We do this by introducing the class-template `Arrangement_on_surface_2<GeomTraits, TopTraits>`, which should be instantiated by two types. The first is the *geometry-traits class*, which defines the family of curves that induce the arrangement, and encapsulates all primitive geometric predicates and constructions (e.g, comparing two points by their *uv*-lexicographic order, computing intersection points, etc.) on curves of this family. The second type is a *topology-traits class*, which encapsulates the topology of the surface on which the arrangement is embedded, and determines the underlying DCEL representation of the arrangement. It does so by supplying predicates and operations related to curve-ends with boundary conditions. For example, it is responsible for initializing a DCEL structure that represents an empty arrangement, and for locating the DCEL feature that represents a given curve-end (this feature may be a fictitious edge as in Fig. 2(a), a vertex at infinity as in Fig. 2(b), etc.). Using the topology-traits primitives, we can use the sweep-line procedure to construct the arrangement of a set of curves on a surface: When we detect a subcurve with boundary conditions, we query the topology-traits class to obtain the DCEL feature containing the curve-end, then insert the subcurve accordingly. For example, if we sweep over the cylinder depicted in Fig. 1(b), a vertex w is created on the line of discontinuity when we insert c_1^ℓ into the arrangement. The topology-traits class keeps track of this vertex, so it will associate w as the minimal end of c_2^ℓ and as the maximal end of c_1^r and c_2^r. Similarly, in the example shown in Fig. 1(c), the north pole will eventually be represented as a single DCEL vertex, with c_1 and c_2 incident to it.

We have already implemented a topology-traits class for handling unbounded curves on the plane, along with geometry-traits classes for handling lines and rays, and with EXACUS[3] based geometry-traits classes for algebraic curves. We have also designed and implemented two other topology-traits classes along with corresponding geometry-traits classes, that define curves on surfaces: the first maintains arrangements of arcs of great circles embedded on a sphere, and the other constructs arrangements of intersection curves between quadric surfaces embedded on a quadric surface. For lack of space, we omit the implementation details.

[3]`http://www.mpi-sb.mpg.de/projects/EXACUS/`.

References

[1] M. V. A. Andrade and J. Stolfi. Exact algorithms for circles on the sphere. *Internat. J. Comp. Geom. Appl.*, 11(3):267–290, 2001.

[2] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, 28(9):643–647, 1979.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, Berlin, Germany, 2nd edition, 2000.

[4] F. Cazals and S. Loriot. Computing the exact arrangement of circles on a sphere, with applications in structural biology. Technical Report 6049, INRIA Sophia-Antipolis, 2006.

[5] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, chap. 1, pages 1–66. Spinger, 2006.

[6] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.

[7] K. Mehlhorn and M. Seel. Infimaximal frames: A technique for making lines look like segments. *Internat. J. Comp. Geom. Appl.*, 13(3):241–255, 2003.

[8] M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. In *Proc. 14th Europ. Sympos. Alg.*, pages 792–803, 2006.

[9] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Comp. Geom. Theory Appl.* To appear.

Efficient Spatial Indexes for Approximate Range Searching

Micha Streppel* Ke Yi†

Abstract

Building efficient spatial indexes for range searching is a central problem in spatial databases. The R-tree has been a popular spatial index thanks to its simplicity, ability to answer various queries, and the flexibility to store spatial objects of different shapes. However, the R-tree is known to be a heuristic-based structure and no guarantees can be made on its query performance. In this paper, we present two disk-based indexes for approximate range searching that are as versatile as the R-tree, and at the same time provide good guarantees on the query performance, albeit in the approximate sense.

1 Introduction

The very basic operation in a spatial database is to retrieve all objects intersected by a query range. For this purpose many access methods, or *spatial indexes*, have been proposed in the past decades, to organize the spatial objects stored in the database so that such a range searching query can be answered efficiently. See the survey by Gaede and Günther [12] and the book by Samet [16].

The R-tree [13] and its many variants [14] have been used heavily in practice. Unfortunately, the R-trees are known to be heuristic-based structures and do not have good guarantees on the query performance. In fact it was shown [2] that in the worst case, a query has to visit $\Omega((N/B)^{1-1/d} + k/B)$ blocks using *any* variant of R-tree built on N points in $\mathbb{R}^d$, where B is the disk block size, and k is the output size. This lower bound is reached by a recently developed R-tree variant, called the PR-tree [4], but the result holds only if both the queries and objects are axis-parallel hypercubes.

Ideally one would like a logarithmic query bound similar to that of the B-tree answering one-dimensional range searching queries. Given the negative result on R-trees, many other types of (theoretical) spatial indexes have been proposed. However, they often require non-linear space or super-logarithmic query costs, and/or can only answer a specific type of queries.

Approximate range searching. Exact range searching either uses non-linear storage or incurs super-logarithmic query time [5], it is therefore natural to seek for approximate solutions. The concept ϵ-*approximate range searching* was first introduced by Arya and Mount [6]. Here one considers, for a parameter $\epsilon > 0$ and a query range Q of constant complexity, the ϵ-extended query range Q_ϵ, which is the locus of points lying at distance at most $\epsilon \cdot \mathrm{diam}(Q)$ from Q, where $\mathrm{diam}(Q)$ is the diameter of Q. For a point set P of N points in $\mathbb{R}^d$, the approximate range searching problem is to return a set P^* such that $P \cap Q \subseteq P^* \subseteq P \cap Q_\epsilon$.

The first structure for approximate range searching, the BBD-tree, was proposed by Arya and Mount [6]. Later, a similar, but simpler structure, called the BAR-tree, was proposed by Duncan et al. [11]. In this paper, we extend these results to external memory, and design new disk-based indexes for answering these queries. Furthermore, we generalize the structure to accommodate arbitrary shapes rather than just points, so that our index can serve as an alternative of R-trees, but with provable and potentially much better bounds.

The I/O-model and previous work. For the analysis of external memory data structures, the standard *I/O model* by Aggarwal and Vitter [3] is often used. In this model, the memory has a limited size M but any computation in memory is free. Only the number of I/Os is considered when analyzing the cost of an algorithm. In one I/O a disk block consisting of B items are read from or written to the external memory.

There has been some work on an efficient disk layout of the BAR-tree. In his thesis [10] Duncan gave an I/O-efficient variant of the BAR-tree, which uses a breadth-first blocking scheme. The number of I/Os for the construction is $O((N/B)\log N)$ and the number of I/Os to answer an approximate range searching query is claimed to be $O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$. However Duncan made the implicit assumption that all blocks contain $\Theta(B)$ nodes, which is not necessarily the case. Some leaves may contain a small number of points and the query bound is in fact $O(\log_B N + \epsilon^\gamma + k_\epsilon)$ in the worst case.

Agarwal et al. [1] gave a general framework for externalizing and dynamizing *weight-balanced partitioning trees* such as the BAR-tree. They describe how a weight-balanced partition tree can be constructed

*Department of Computer Science, TU Eindhoven, P.O.Box 513, 5600 MB Eindhoven, the Netherlands. Supported by the Netherlands Organisation for Scientific Research (N.W.O.) under project no. 612.065.203.

†AT&T Labs - Research, Florham Park, NJ 07932, USA.

in the optimal $O(N/B \log_{M/B} N/B)$ I/Os. Like Duncan [10], they use a breadth-first blocking scheme for storing the resulted tree on disk. To remove the assumption made by Duncan they group blocks together which contain too few nodes. As a result there is at most one block containing too few nodes. This improvement ensures that the resulted layout only uses $O(N/B)$ disk blocks, but the approximate range searching cost is still $O(\log_B N + \epsilon^\gamma + k_\epsilon)$, since there can be $\Theta(B)$ blocks containing one (or a few) subtrees of constant size whose stored points have to be reported.

Our results. We obtain two main results in this paper. We first give a new blocking scheme for the BAR-tree that yields the first disk-based index structure, the *BAR-B-tree*. It answers an approximate range searching query in the desired $O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$ I/Os, i.e., achieving an $O(\log_B N)$ search term and an $O(k_\epsilon/B)$ output term simultaneously. Such terms are optimal when disk-based indexes are concerned. Unfortunately it seems difficult to reduce the error term $O(\epsilon^\gamma)$. We can also bulk load the BAR-B-tree efficiently.

Next, we generalize the BAR-B-tree to the *object-BAR-B-tree*, which indexes not just points, but arbitrary spatial objects of constant complexity. Our idea is based on range searching data structures for *low-density scenes* [9, 8]. It is believed that many realistic inputs are low-density scenes.

2 The BAR-B-tree

In this section we describe the BAR-B-tree, an efficient layout for the BAR-tree on disk. We introduce our two-stage blocking scheme and analyze its query cost when answering an approximate range searching query in Section 2.1. We give an efficient bulk loading algorithm in Section 2.2. For the remainder of the paper we assume that $\mathcal{T}$ has at least $B/2$ nodes.

2.1 The blocking scheme

For any node $u \in \mathcal{T}$, let $\mathcal{T}_u$ be the subtree rooted at u, and we define $|\mathcal{T}_u|$, the size of $\mathcal{T}_u$, to be the number of nodes in $\mathcal{T}_u$ (including u). Our blocking scheme consists of two stages. In the first stage the tree is blocked such that for any $u \in \mathcal{T}$, $\mathcal{T}_u$ is stored in $O(\lceil |\mathcal{T}_u|/B \rceil)$ blocks. In the second stage we make sure that any root-to-leaf path can be traversed by accessing $O(\log_B N)$ blocks.

Tree-blocks. In the first stage we block the tree $\mathcal{T}$ into *tree-blocks* such that $\mathcal{T}$ is stored in $O(\lceil |\mathcal{T}|/B \rceil)$ blocks. The blocking procedure is detailed in Algorithm 3. We traverse the tree $\mathcal{T}$ in a top-down fashion, and keep in a set $\mathcal{S}$ all nodes u for which a block will be allocated such that u is the topmost node in the block. Initially $\mathcal{S}$ only contains the root of $\mathcal{T}$. For any node $u \in \mathcal{S}$, we find a connected subtree rooted at u to fit in one block using an adapted breadth-first strategy with a queue $\mathcal{Q}$. Throughout the blocking algorithm we maintain the invariant that $|\mathcal{T}_u| \geq B/2$ for any u that is ever added to $\mathcal{S}$ or $\mathcal{Q}$. The invariant is certainly true when the algorithm initializes (line 1).

Algorithm 3 Algorithm to construct tree-blocks

Input: a binary tree $\mathcal{T}$
Output: a set of tree-blocks stored on disk

```
1: initialize $\mathcal{S} := \{$root of $\mathcal{T}\}$, and a block $\mathcal{B} := \emptyset$
2: while $\mathcal{S} \neq \emptyset$ do
3:   remove any node $u$ from $\mathcal{S}$
4:   initialize a queue $\mathcal{Q} := \{u\}$
5:   while $\mathcal{Q} \neq \emptyset$ do
6:     remove the first node $v$ from $\mathcal{Q}$, let $v_1, v_2$ be $v$'s children
7:     if $|\mathcal{T}_v| \leq B$ then
8:       put $\mathcal{T}_v$ in a new block $\mathcal{B}'$
9:       write $\mathcal{B}'$ to disk
10:    else if $|\mathcal{T}_{v_1}| \geq B/2$ and $|\mathcal{T}_{v_2}| \geq B/2$ then
11:      add $v$ to $\mathcal{B}$
12:      add $v_1, v_2$ to $\mathcal{Q}$
13:    else
14:      suppose $|\mathcal{T}_{v_1}| < B/2$
15:      if $|\mathcal{B}| + |\mathcal{T}_{v_1}| + 1 \leq B$ then
16:        add $v$ and $\mathcal{T}_{v_1}$ to $\mathcal{B}$
17:        add $v_2$ to $\mathcal{Q}$
18:      else
19:        add $v$ to $\mathcal{S}$
20:    if $|\mathcal{B}| = B$ then
21:      write $\mathcal{B}$ to disk and reset $\mathcal{B} := \emptyset$
22:      add all nodes of $\mathcal{Q}$ to $\mathcal{S}$
23:      set $\mathcal{Q} := \emptyset$
24:  if $|\mathcal{B}| \neq \emptyset$ then
25:    write $\mathcal{B}$ to disk and reset $\mathcal{B} := \emptyset$
```

For a node $u \in \mathcal{S}$, we fill a block with a top portion of $\mathcal{T}_u$ by an adapted breadth-first search (line 4–23). The BFS starts with $\mathcal{Q} = \{u\}$ (line 4), which is consistent with the invariant since u is a node from $\mathcal{S}$. For each node v encountered in the BFS search, we distinguish among the following three cases. (a) If $|\mathcal{T}_v| \leq B$, then we allocate a new block to store the entire $\mathcal{T}_v$ (line 7–9). Note that this block contains at least $B/2$ nodes by the invariant. (b) Let v_1, v_2 be the two children of v. If both $\mathcal{T}_{v_1}$ and $\mathcal{T}_{v_2}$ have more than $B/2$ nodes, then we add v to the block and continue the BFS process (line 10–12). It is safe to add v_1, v_2 to $\mathcal{Q}$ as we have ensured the invariant. (c) Otherwise, it must be the case that one of the subtrees is smaller than $B/2$ nodes while the other one has more than $B/2$ nodes. Without loss of generality we assume $|\mathcal{T}_{v_1}| < B/2$, and then check if $\mathcal{T}_{v_1}$ plus v itself still fits in the current block. If so we add v and the entire $\mathcal{T}_{v_1}$ to the current block, add v_2 to $\mathcal{Q}$ and continue the BFS; else we put v into $\mathcal{S}$, and will allocate a new block for v (line 14–19). Notice that the blocking process for v will not go back to line 19 again since with a new empty block, $\mathcal{T}_{v_1}$ and v must be able to fit. Please refer to Figure 1 for an illustration of this blocking algorithm.

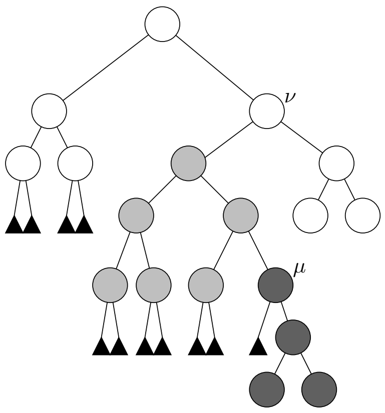


Figure 1: Three tree-blocks (white, light gray and dark gray) obtained using the blocking scheme for $B = 8$. The black triangles denote the existence of a subtree of size at least $B/2$. The right subtree of ν is placed completely in the white block. The node μ and its right subtree do not fit in the light gray block so a new block must be started at μ.

Lemma 1 *For any $u \in \mathcal{T}$, the nodes in $\mathcal{T}_u$ are stored in $O(\lceil|\mathcal{T}_u|/B\rceil)$ blocks.*

The blocked BAR-tree resulting after the first stage might have depth as bad as $\Theta(\log N)$. In the second stage we introduce *path-blocks* which ensure that $O(\log_B N)$ blocks have to be accessed in order to visit all nodes on any root-to-leaf path.

Path-blocks. To identify the places where a path-block has to be introduced we visit $\mathcal{T}$ in a top-down fashion. For a node u, if $|\mathcal{T}_u| \leq B$ we stop. From Lemma 1 we know that $\mathcal{T}_u$ is already stored in $O(1)$ tree-blocks. Otherwise we consider the top subtree of B nodes of $\mathcal{T}_u$ obtained by a BFS starting from u. We denote this subtree by $\widehat{\mathcal{T}}_u$. We check all root-to-leaf paths in $\widehat{\mathcal{T}}_u$. If there is at least one such path that is divided among more than c tree-blocks for some integer constant $c \geq 2$, then we introduce a path-block that stores $\widehat{\mathcal{T}}_u$. We also remove all nodes of $\widehat{\mathcal{T}}_u$ from the tree-blocks where they are stored. Finally we continue this process recursively with each subtree below $\widehat{\mathcal{T}}_u$.

This completes our two-stage blocking scheme. With the introduction of path-blocks, now we have the following.

Theorem 2 *A BAR-B-tree on N points in $\mathbb{R}^d$ takes $O(N/B)$ disk blocks and any root-to-leaf path in $\mathcal{T}$ can be traversed by accessing $O(\log_B N)$ blocks.*

Since no node is stored in multiple blocks we can use the standard query algorithm for BSPs. The traversal can be performed in either a BFS or DFS manner, with the use of an I/O-efficient stack or queue such that the extra overhead is $O(1)$ I/Os per B nodes.

Theorem 3 *A range searching query Q in a BAR-B-tree can be answered by accessing $O(\log_B N + \epsilon^{\gamma} + k_\epsilon/B)$ blocks.*

2.2 An efficient construction algorithm

We are left with giving an I/O-efficient bulk loading algorithm to build a BAR-B-tree with a set of N points in $\mathbb{R}^d$. We use the "grid" technique introduced by Agarwal et al. [1], which we briefly review here. The grid technique can be used to construct a class of space partitioning structures I/O-efficiently, including the BAR-tree. The idea is to first build a grid of size $\Theta((M/B)^c)$ in memory for some constant $0 < c \leq 1/2$, which is then used to build the top $\Theta(\log(M/B))$ levels of the BSP. Next the data set is partitioned into subsets that correspond to the subtrees below. This process can be completed with a constant number of scans of the data set. Finally we recurse to build the subtrees. The recursion stops when we have less than M points to deal with, for which we just build the entire subtree in memory. The overall cost is then $O(N/B \log_{M/B} N/B)$ I/Os. Observing that each recursive call to the grid method must still have at least B points to deal with, since $M/(M/B)^c \geq B$, our top-down algorithm for the construction of the tree-blocks can easily be coupled with the also top-down grid technique. After we have built $\widehat{\mathcal{T}}$, the top $\Theta(\log(M/B))$ levels of $\mathcal{T}$ using the grid the technique, since the grid also gives us all the subtree sizes of $\widehat{\mathcal{T}}$ [1], we can run Algorithm 3 on $\widehat{\mathcal{T}}$. However, some tree-blocks at the bottom of $\widehat{\mathcal{T}}$ are not complete, i.e., they include some nodes below $\widehat{\mathcal{T}}$ that have not been built yet. Then we simply push these incomplete tree-blocks into the corresponding subtrees for which the grid method will recurse. Later when the grid method recurses on a subtree $\mathcal{T}_u$, we will be able to resume Algorithm 3 from the incomplete tree-block that contains u. This modification to the grid method incurs at most $O(N/B)$ additional I/Os. After constructing the tree-blocks, we build the path-blocks as described above. It takes $O(N/B)$ I/Os to build all the path-blocks. This completes the analysis of our bulk loading algorithm.

Theorem 4 *It takes $O(N/B \log_{M/B} N/B)$ I/Os to bulk load a BAR-B-tree on a set of N points in $\mathbb{R}^d$.*

3 Extension to objects: the object-BAR-B-tree

In this section we show how to externalize the object-BAR-tree [9] for a λ-low-density set S of objects of constant complexity. The object-BAR-tree is based on the idea of *guarding sets* [7, 15]. For a subset $X \subseteq S$, a set of points G_X is called a *guarding set* of X if the region associated with any leaf in the BAR-tree constructed on G_X intersects at most $O(\lambda)$ objects of X.

We first build all the guards with a scan over S. For $\mathbb{R}^2$ we can use the simple construction of De Berg et al. [9]. For $\mathbb{R}^d$, $d \geq 3$ there also exists a guarding set but due to lack of space we will not mention it here[1].

Next we build the BAR-B-tree on the set of all guards G using Algorithm 3. The adaptation of removing guards during the construction as described above can be easily accommodated in the algorithm, and we can build and lay out the tree $\mathcal{T}$ on disk in $O(\lambda N/B \log_{M/B} \lambda N/B)$ I/Os. During the process we can also compute for each leaf v of $\mathcal{T}$, the set of at most $O(\lambda)$ objects that intersect the region R_v. We omit the technical details.

Finally, for each leaf block $\mathcal{B}$ of $\mathcal{T}$, we store all the intersecting objects consecutively on disk. More precisely, consider a block $\mathcal{B}$ and let L be the set of leaves stored in $\mathcal{B}$. The objects intersecting the regions of the nodes in L are stored together in one list as follows. Let $v_1, \cdots, v_{|L|}$ be the leaves in L ordered according to an in-order traversal of $\mathcal{T}$. We first store the objects intersecting R_{v_1}, then the objects intersecting R_{v_2}, etc. Note that an object might be stored more than once in the list. At every leaf v_i we store a pointer to the first and last object in the list intersecting R_{v_i}.

Theorem 5 *Let S be a set of N objects in $\mathbb{R}^d$ with density λ. An object-BAR-B-tree on S takes $O(\lambda N/B)$ blocks and can be constructed using $O(\lambda N/B \log_{M/B} \lambda N/B)$ I/Os. An object-BAR-B-tree for S answers an approximate range searching query Q using $O(\log_B N + \lceil \lambda/B \rceil \epsilon^{\gamma} + \lambda k_\epsilon/B)$ I/Os, where k_ϵ is the number of objects intersecting Q_ϵ.*

[1]The guarding set construction for $\mathbb{R}^d$, $d \geq 3$ mentioned in [9] is incorrect since there is no direct relation between the guards and the objects in any subset X of S.

References

[1] P. K. Agarwal, L. Arge, O. Procopiuc, and J. S. Vitter. A framework for index bulk loading and dynamization. In *Proc. International Colloquium on Automata, Languages, and Programming*, pages 115–127, 2001.

[2] P. K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H. J. Haverkort. Box-trees and R-trees with near-optimal query time. *Discrete and Computational Geometry*, 28(3):291–312, 2002.

[3] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[4] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. In *Proc. SIGMOD International Conference on Management of Data*, pages 347–358, 2004.

[5] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. ACM Symposium on Principles of Database Systems*, pages 346–357, 1999.

[6] S. Arya and D. M. Mount. Approximate range searching. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 172–181, 1995.

[7] M. de Berg, H. David, M. J. Katz, M. Overmars, A. F. van der Stappen, and J. Vleugels. Guarding scenes againt invasive hypercubes. *Computational Geometry: Theory and Applications*, 26:99–117, 2003.

[8] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 294–303, 1997.

[9] M. de Berg and M. Streppel. Approximate range searching using binary space partitions. *Computational Geometry Theory and Applications*, 33(3):139–151, 2006.

[10] C. Duncan. *Balanced Aspect Ratio Trees.* PhD thesis, John Hopkins University, 1999.

[11] C. Duncan, M. Goodrich, and S. Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. *Journal of Algorithms*, 38:303–333, 2001.

[12] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.

[13] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[14] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. *R-trees: Theory and Applications.* Springer, 2005.

[15] J. Nievergelt and P. Widmayer. Guard files: Stabbing and intersection queries on fat spatial objects. *Comput. J.*, 36:107–116, 1993.

[16] H. Samet. Spatial data structures. In W. Kim, editor, *Modern Database Systems, The Object Model, Interoperability and Beyond*, pages 361–385. ACM Press and Addison-Wesley, 1995.

Exact Computation of Arrangements of Rotated Conics*

Eric Berberich† Manuel Caroli† Nicola Wolpert‡

Abstract

Transformations of geometric objects, like translation and rotation, are fundamental operations in CAD-systems. Rotations trigger the need to deal with trigonometric functions, which is hard to achieve when aiming for exact and robust implementation.

We show how we efficiently compute the planar arrangement of conics rotated by angles that can be constructed with straightedge and compass. Well-known examples are multiples of $45°$, $30°$, and $15°$. The main problem one has to solve is root-isolation of univariate polynomials $p(x) \in \mathbb{Q}(\sqrt{c_1}) \dots (\sqrt{c_d})[x]$, for which we use a modified version of the Descartes method. For $d = 1$, we additionally present a new method that isolates the real roots of p by using root isolation for polynomials $q(x) \in \mathbb{Q}[x]$ only. We show results of our benchmark experiences comparing both methods.

1 Introduction

We construct a set of transformed conics C' by applying an individual sequence of translations and rotations to each conic of a given set C. We aim to compute the subdivision of the plane induced by C' into cells of dimension 0 (*vertices*), of dimension 1 (*edges*), and of dimension 2 (*faces*) — commonly referred to as the *arrangement* of C'. Arrangements are well-studied in the field of computational geometry and serve as a basis for different applications [12]. In the past years, research concentrated on finding exact, robust, and efficient solutions to compute the arrangement of non-linear objects. Emiris et al. [11] concentrated on circles, while arrangements of conics have been considered in CGAL[1] by Wein [16] and in EXACUS[2] by Berberich et al. [3], all using a modified version [14] of the sweep-line algorithm [1].

Performing arbitrary rotations requires evaluation of trigonometric functions. Canny et al. [5] introduced a rotation scheme to approximate rotations by α with angles $\bar{\alpha}$ whose sine and cosine are rational. Such angles $\bar{\alpha}$ are dense in $[0, 2\pi)$, which can be be easily seen by the parameterization $(\frac{2t}{1+t^2}, \frac{1-t^2}{1+t^2})$ of the unit circle. Each desired angle can be arbitrarily approximated by existing implementations. But this method leads to increased bitlengths of the involved coefficients and still does not compute the exact solutions as expected by the *exact geometric computation* paradigm [18]. Each exact solution recalls the question whether trigonometric computations can be made "geometrically exact" – first tackled by [7]. We give a positive answer for rotations by angles that can be constructed with straightedge and compass. Such rotations are discussed in Section 2. We explain in Section 3 how to compute the intersection points of such rotated conics and present details of the new implementation within EXACUS in Section 4. This work concludes in Section 5 with a selection of experiments.

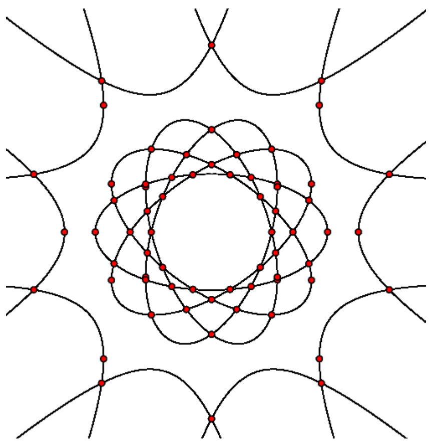


Figure 1: The arrangement of a hyperbola and an ellipse, both rotated by angles of $0°, 36°, 72°, 108°$, and $144°$.

2 Transformations of Conics

Definition 1 (Conic) *Let $\mathbb{K}$ be a field and $f(x, y) \in \mathbb{K}[x, y]$ a bivariate polynomial. If $\deg(f) \leq 2$ we call the zero set $\mathcal{V}(f) := \{(x, y) \mid f(x, y) = 0\}$ a* conic *and denote it by c.*[3]

A usual choice is $\mathbb{K} = \mathbb{Q}$. Since computations with integers are much faster than using rational arith-

*This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes)

†Max-Planck-Institut für Informatik, Saarbrücken, Germany, {eric, mcaroli}@mpi-inf.mpg.de

‡University of Applied Sciences, Stuttgart, Germany nicola.wolpert@hft-stuttgart.de

[1]http://www.cgal.org/

[2]http://www.mpi-inf.mpg.de/projects/EXACUS/

[3]For convenience, we also use c to denote the polynomial f.

metic, we always multiply rational coefficients by their common denominator keeping the curve unchanged.

Definition 2 (Euclidean Transformations) *Let $\boldsymbol{x}, b \in \mathbb{K}^n$, and let $A \in \mathbb{K}^{n\times n}$ an orthogonal $n \times n$-matrix. Transformations of the form $\mathcal{E}(\boldsymbol{x}) := A\boldsymbol{x} + b$ are called* Euclidean Transformations.

We aim for the coefficients of the new conic $\bar{c}$ when applying $\mathcal{E}$ on c, i. e., for each $(x, y) \in \mathcal{V}(c)$ we want that $\mathcal{E}(x, y) \in \mathcal{V}(\bar{c})$. This condition is fulfilled if we choose $\bar{c} := c \circ \mathcal{E}^{-1}$. Since A is orthogonal we get $\mathcal{E}^{-1}(x, y) = A^T\binom{x}{y} - A^T b$. Let us consider rotations by an angle α. Then

$$A = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}, \qquad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

If the entries of A are rational we can use the existing implementations. Otherwise the entries of A are non-rational and in most cases they are even transcendental. For some angles α we have $\sin(\alpha)$ and $\cos(\alpha) \in \mathbb{Q}(\sqrt{c_1})\dots(\sqrt{c_d})$, for a constant d and all $0 < c_i \in \mathbb{Q}$. This holds for all angles that can be constructed by straightedge and compass, e. g., multiples of $\alpha = \frac{2\pi}{k}$, with $k = 2^n F_1 \dots F_l$, $l, n \in \mathbb{N}$, where the F_i are Fermat primes. Fermat primes are of the form $F_i = 2^{2^{e_i}} + 1$ for some natural $e_i \neq e_j$. Since the Fermat primes grow very fast, it is clear that we can get very small angles, e. g., $1.5° = \frac{360°}{2^4 F_0 F_1}$. Note that angles can be halved by straightedge and compass (one additional square root). In the following sections we only consider angles for rotations constructible by straightedge and compass. Observe that the angle $\alpha = 1°$ is not constructible this way [15].

3 Intersection Points of Rotated Conics

Newer implementations of the sweep-line algorithm require a set of basic geometric predicates on the curves to be swept [17]. It even can be reduced to the topological analysis of single curves and pairs of curves [2]. A basic step is to find all x-coordinates of intersection points of two curves. These are usually computed by real root isolation of a univariate polynomial p that is obtained by a resultant computation [19]. In case of conics $\deg(p) \leq 4$. Real root isolation means to determine for every real root of p a (rational) interval $[l, r]$, that contains exactly one root of p. A well-known technique for real root isolation is the Descartes method [8, 10] that we adapted for our purpose. Although intended for integral polynomials, it is also applicable to polynomials with non-rational coefficients, as in our case $p \in \mathbb{Q}(\sqrt{c_1})\dots(\sqrt{c_d})[x]$. If only one root is adjoined, we additionally explore another technique to isolate the real roots:

Theorem 1 *The roots of a polynomial $p \in \mathbb{Q}(\sqrt{c})[x]$ with $\deg(p) \leq 4$ can be isolated by only using real root isolation on polynomials $\bar{p} \in \mathbb{Q}[x]$.*

We sketch a constructive proof by giving the main ideas of the algorithm. Note that for $p \in \mathbb{Q}(\sqrt{c})[x]$ we have

$$\begin{aligned} p(x) &= \sum_{i=0}^{4} (a_i + \sqrt{c}\cdot b_i)x^i \\ &= \sum_{i=0}^{4} a_i x^i + \sqrt{c}\cdot\sum_{i=0}^{4} b_i x^i \\ &= \alpha(x) + \sqrt{c}\cdot\beta(x) \end{aligned}$$

with $\alpha, \beta \in \mathbb{Q}[x]$. We consider the bivariate polynomial $q(x, u) := \alpha(x) + u\beta(x)$ and our goal is to isolate the real roots of $q(x, \sqrt{c}) = p(x)$. Note that q is linear in u with coefficients in $\mathbb{Q}[x]$. If $\mathrm{con}(q) := \gcd(\alpha(x), \beta(x))$ is constant, $\mathcal{V}(q)$ defines the graph of a function in x. Otherwise, for each root x_i of $\mathrm{con}(q)$ we have: $\forall u\ q(x_i, u) = 0$. More intuitively: q defines an algebraic curve of degree 4 in the xu-plane that either comprises of the graph of a function or the graph of a function multiplied with some vertical lines.

To isolate the roots of $q(x, \sqrt{c})$ we approximate $\sqrt{c}$ by an interval $[s, t]$, $s, t \in \mathbb{Q}$, such that at most $\sqrt{c}$ is a root of $\rho(u) := \mathrm{res}_x(q, \frac{\partial}{\partial x}q)$ in $[s, t]$. We infer the desired isolating intervals from the isolating intervals of $p_s(x) := q(x, s)$ and $p_t(x) := q(x, t)$. Observe that $p_s, p_t \in \mathbb{Q}[x]$. Depending on whether $\rho(\sqrt{c}) = 0$ we have two distinguish two cases:

***p* square-free:** It is easy to see that $\sqrt{c}$ is not a root of $\rho(u)$. The number of roots of p_s and p_t is equal and bounded by 4. Since q is continuous and locally either the graph of a function or a vertical line, and due to the choice of s and t, we have, for each root x_i, $i \leq 4$, $\lim_{s\nearrow\sqrt{c}} q(x_i, s) = q(x_i, \sqrt{c}) = \lim_{t\searrow\sqrt{c}} q(x_i, t)$. The convex hull of properly refined isolating intervals for $q(x_i, s)$ and $q(x_i, t)$ forms the isolating interval of $q(x_i, \sqrt{c})$.

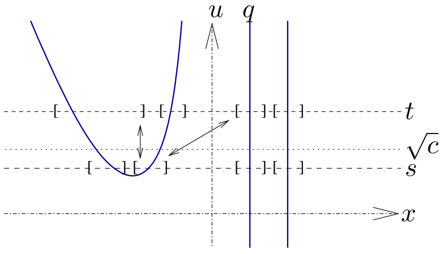


Figure 2: The isolating intervals for $q(x_i, s)$ and $q(x_i, t)$ define the isolating interval for $q(x_i, \sqrt{c})$, after the ones for $q(x_i, s)$ are refined (examples indicated by $\leftrightarrow$) with respect to $q(x_{i-1}, t)$ and $q(x_{i+1}, t)$. Similar for $q(x_i, t)$.

***p* not square-free:** We have $\rho(\sqrt{c}) = 0$ which disables us to infer the desired isolating intervals directly from the isolating intervals of roots of p_s and p_t, since the number of roots may differ. Let $\sigma(u) := \text{res}_x(\frac{\partial}{\partial x}q, \frac{\partial^2}{\partial x^2}q)$. Both, $\rho(u)'$ and $\sigma(u)$ help to distinguish different cases for the number of roots of p and their multiplicities listed in the given table.

	$\rho'(\sqrt{c})$	$\sigma(\sqrt{c})$	# of multiplicities			
			1	2	3	4
A	$\neq 0$		≤ 2	1	0	0
B	$= 0$	$\neq 0$	0	2	0	0
C	$= 0$	$= 0$	≤ 1	0	1	0
D			0	0	0	1

Note that in case A and C we still need to assign the correct multiplicity to each root. We omit the full case distinction conducted in [6], e. g., additional criteria that help to distinguish between case C and D, and exemplarily, we present case A.

The double root, denoted by $\bar{\bar{x}}$, may or may not induce roots in p_s and p_t. Consider the case of a vertical line at $\bar{\bar{x}}$. Then the numbers of roots are positive and equal. Let $q_x(x, u) := \frac{\partial}{\partial x}q(x, u)$. Then $(x - \bar{\bar{x}})$ is a double factor of q and a simple factor of q_x that can be computed by means of the greatest common divisor.

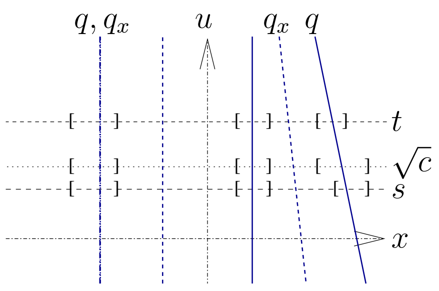


Figure 3: p_s and p_t show 3 roots that do not determine the double root of $p(x, \sqrt{c})$, but $\gcd(q, q_x)$ gives it.

Otherwise, $\bar{\bar{x}}$ induces no root in p_s and two in p_t, or vice versa. It suffices to isolate all roots of p_s with respect to the roots of p_t until we locate $\bar{\bar{x}}$.

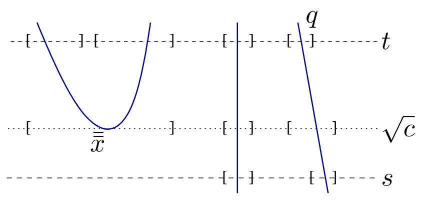


Figure 4: The isolating and cross-refined intervals for roots of p_s and p_t determine the simple roots and the double root $\bar{\bar{x}}$ of $p(x)$.

When analyzing the topology of a pair of conics, we also require to compare y-coordinates of points on conics (x, y) whose x-coordinates are equal. This has to be done only in cases where the coordinates can be represented and exactly as well as efficiently compared by number types like `leda::real` [4] or `CORE::Expr` [13]. This also holds for the case of rotated conics.

4 Implementation

To compute arrangements of rotated conics we extend the EXACUS libraries by the two real root isolators mentioned in Section 3: A modified version of the Descartes method that works on polynomials $p \in \mathbb{Q}(\sqrt{c_1})\dots(\sqrt{c_d})[x]$, and the isolator that infers the isolating intervals from roots of integral polynomials. We also introduce a new representation class template for transformed conics (`CnX::Rotated_conic_2`), that provides access to a *transformation history*, i. e., a sequence of rotations and translations applied to the original integral polynomial. Its actual behavior is determined by a model of the *RotatedConicTraits* concept. It especially defines the allowed rotation angles, the number type used for the coefficients of the bivariate polynomial, and the method to isolate the real roots of its univariate counterparts. For the number type of the coefficients we rely on `NiX::Sqrt_extension`. It represents (nested) square root extensions, i. e., numbers of the form $a + b \cdot \sqrt{c}$, where a, b, and c are of type `Integer` or, if nested, even another instance of `NiX::Sqrt_extension` again. A such equipped version of `Rotated_conic_2` inherits all the functionality from the generic `Conic_2` class which is required for arrangement computations (e. g., with CGAL's `Arrangement_2` package) or even to compute boolean set operations on polygons bounded by arcs of rotated conics. EXACUS 1.0 contains traits classes to rotate conics by multiples of 45°, 30°, and 15°, dealing with all degenerate cases. Rotations by other angles constructible with straightedge and compass can be implemented straightforward, as we recently did for 36°.

5 Benchmark Results

We provide a selected excerpt of the benchmark results from [6]. We compute the arrangement of n (varying from 10 to 200) randomly chosen conics with the generic implementation of the sweep-line algorithm in EXACUS. Three approaches have been tested:

`CnX`: non-rotated conics using the Descartes method for real root isolation

`s-t`: rotated conics using the Descartes method twice for integral p_s and p_t to isolate the real roots of $p \in \mathbb{Q}(\sqrt{c})[x]$

`Des`: rotated conics using the Descartes method directly on $p \in \mathbb{Q}(\sqrt{c})[x]$

All conics are rotated by 45°. The running times are taken from runs on an Intel Pentium 4 CPU, clocked at 2.8 GHz with 512 kB cache. We list and illustrate the results in Figure 5 and Table 1. As one expects, the computation of arrangements of non-rotated conics is fastest. Running times roughly double when we switch to rotated conics. The comparison of Des versus s-t shows that Des usually is measurably faster, such that we recommend to use the extended Descartes method when computing arrangements of rotated conics. The running times in general increase remarkable if more sophisticated rotation angles are involved, like rotations by 36°.

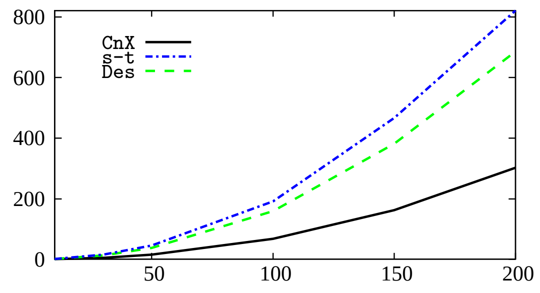


Figure 5: Comparison of running times for arrangement computation of non-rotated and rotated conics

No.	CnX	s-t	Des
10	0.6	1.5	1.3
100	67.8	190.9	158.8
200	302.5	821.2	686.3

Table 1: Running times in seconds.

We next plan to apply the Descartes method for bit-stream coefficients by Eigenwillig et al. [9] to compute intersection points of rotated conics.

Acknowledgments

We thank all EXACUS developers, especially Arno Eigenwillig, Michael Hemmer, and Tobias Reithmann, and we thank Michael Sagraloff for fruitful discussions.

References

[1] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, **28**(9):643–647, 1979.

[2] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, and N. Wolpert. EXACUS: Efficient and exact algorithms for curves and surfaces. In *Proc. 13th Europ. Sympos. Alg. (ESA)*, volume **3669** of *LNCS*, pages 155–166, 2005.

[3] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and Boolean operations on conic polygons. In *Proc. 10th Europ. Sympos. Alg. (ESA)*, volume **2461** of *LNCS*, pages 174–186, 2002.

[4] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Efficient exact geometric computation made easy. In *15th ACM Symposium on Computational Geometry (SCG'99)*, pages 341–350, New York, NY, USA, 1999. ACM Press.

[5] J. Canny, B. Donald, and E. K. Ressler. A rational rotation method for robust geometric algorithms. In *SCG '92: Proceedings of the eighth annual symposium on Computational geometry*, pages 251–260, New York, NY, USA, 1992. ACM Press.

[6] M. Caroli. Exakte Arrangement-Berechnung gedrehter quadratischer Kurven. Bachelor's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2005.

[7] E.-C. Chang, S. W. Choi, D. Kwon, H. Park, and C. K. Yap. Shortest path amidst disc obstacles is computable. In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, pages 116–125, New York, NY, USA, 2005. ACM Press.

[8] G. E. Collins and A. G. Akritas. Polynomial real root isolation using descarte's rule of signs. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 272–275, New York, NY, USA, 1976. ACM Press.

[9] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A descartes algorithm for polynomials with bit-stream coefficients. In *CASC*, pages 138–149, 2005.

[10] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the descartes method. In *ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 71–78, New York, NY, USA, 2006. ACM Press.

[11] I. Z. Emiris, A. Kakargias, S. Pion, M. Teillaud, and E. P. Tsigaridas. Towards an open curved kernel. In *Proc. 20th Annu. ACM Sympos. Comput. Geom. (SCG)*, pages 438–446, 2004.

[12] E. Fogel, D. Halperin, L. Kettner, M. Teillaud, R. Wein, and N. Wolpert. Arrangements. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, chapter **1**, pages 1–66. Springer, 2006.

[13] V. Karamcheti, C. Li, I. Pechtchanski, and C. K. Yap. A core library for robust numeric and geometric computation. In *15th ACM Symposium on Computational Geometry (SCG'99)*, pages 351–359, 1999.

[14] J. Snoeyink and J. Hershberger. Sweeping arrangements of curves. In *Proc. 5th Annu. ACM Sympos. Comput. Geom. (SCG)*, pages 354–363, 1989.

[15] I. Stewart. *Galois theory*. Chapman and Hall, 2nd ed. edition, 1992.

[16] R. Wein. High-level filtering for arrangements of conic arcs. In *Proc. 10th Europ. Sympos. Alg. (ESA)*, volume **2461** of *LNCS*, pages 884–895, 2002.

[17] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. In *1st Wrkshp. Library-Centric Software Design. (LCSD'06)*, 2005.

[18] C. K. Yap. Towards exact geometric computation. *CGTA: Computational Geometry: Theory and Applications*, 7, 1997.

[19] C. K. Yap. *Fundamental problems of algorithmic algebra.* Oxford University Press, Inc., New York, NY, USA, 2000.

Computing Envelopes of Quadrics[*]

Eric Berberich[†] Michal Meyerovitch[‡]

Abstract

We present the computation of envelopes of a set of quadratic surfaces defined in $\mathbb{R}^3$. Our solution is based on the new CGAL `Envelope_3` package that provides a generic and robust implementation of a divide-and-conquer algorithm. This work concentrates on the theory of algebraic and combinatorial tasks that occur for quadratic surfaces and their implementation. The implementation is exact and efficient.

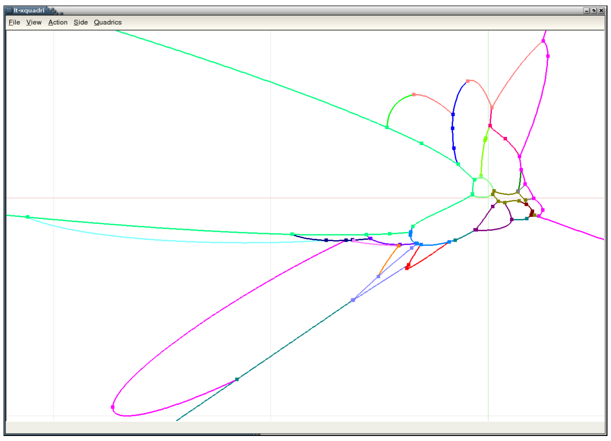


Figure 1: Cutout of the lower envelope of 400 quadrics, hyperboloids and ellipsoids. It consists of 30 faces, 4 of which are unbounded, 101 edges, and 76 vertices.

1 Envelopes

Lower envelopes are fundamental structures in computational geometry, which have many applications like computing general Voronoi diagrams, or performing hidden surface removal. Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a set of n (hyper)surface patches in $\mathbb{R}^d$. Let $x_1, \ldots, x_d$ denote the axes of $\mathbb{R}^d$, and assume that each s_i is monotone in $(x_1, \ldots, x_{d-1})$, namely every line parallel to the x_d-axis intersects s_i in at most one point (without multiplicities). Regard each patch s_i as a partially defined $(d-1)$-variate function $s_i(x_1, \ldots, x_{d-1})$.

Definition 1 (Envelope) *The* lower envelope $\mathcal{E}_{\mathcal{S}}$ *of $\mathcal{S}$ is the pointwise minimum of these functions: $\mathcal{E}_{\mathcal{S}}(x_1, \ldots, x_{d-1}) := \min s_i(x_1, \ldots, x_{d-1})$, where the minimum is taken over all functions defined at $(x_1, \ldots, x_{d-1})$.*

Similarly, the *upper envelope* is defined as the pointwise maximum of these functions. Throughout the remaining parts we refer for the sake of simplicity to lower envelopes only.

Definition 2 (Minimization Diagram) *The* minimization diagram $\mathcal{M}_{\mathcal{S}}$ *of $\mathcal{S}$ is the subdivision of $\mathbb{R}^{d-1}$ into maximal connected cells such that $\mathcal{E}_{\mathcal{S}}$ is attained by a fixed (possibly empty) subset of functions over the interior of each cell.*

The complexity of an envelope [10, 19, 20] is given by the complexity of its minimization diagram. The computation of an envelope is a non-trivial task. Note that the minimization diagram can be easily extracted from the *cylindrical algebraic decomposition* (cad) scheme [7]. Several more efficient algorithms have been developed for $d = 3$. Agarwal et at. [1] gave a divide-and-conquer method, Boissonnat and Dobrindt [5] showed a randomized incremental algorithm. Both algorithms run in time $\mathcal{O}(n^{2+\epsilon})$, with $\epsilon > 0$. For special cases there also exist output-sensitive algorithms [8, 12, 17]. Meyerovitch recently presented the generic and exact implementation of a divide-and-conquer algorithm that decouples the combinatorial part from the geometric predicates [15]. In order to support a new class of surface patches it suffices to implement a set of geometric types and operations on them, as we do in Section 3 for quadrics. A full description of the implementation is given in [14]. We only repeat that all the families of surfaces that are supported benefit from combinatorial deductions carried out by the algorithm. These deductions significantly reduce the amount of geometric constructions and comparisons. Such operations are usually expensive, especially when using exact geometric computation [23]. The implementation will be available as the `Envelope_3` package of the upcoming release (3.3) of CGAL.[1] It is based on the mature `Arrangement_2`

[*]This work has been supported in part by the IST Programme of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS - Algorithms for Complex Shapes), by the Israel Science Foundation (grant no. 236/06), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

[†]Max-Planck-Institut für Informatik, Saarbrücken, Germany, eric@mpi-inf.mpg.de

[‡]School of Computer Science, Tel-Aviv University, Israel, gorgymic@post.tau.ac.il

[1]`http://www.cgal.org/`

package, which is a well-taken choice, since the problem actually is two-and-a-half-dimensional: The input $\mathcal{S}$ consists of objects in $\mathbb{R}^3$, while their minimization diagram is represented by an augmented planar arrangement in $\mathbb{R}^2$.

2 Quadrics

Definition 3 (Quadratic Algebraic Surface) *Let $\mathbb{K}$ be a field and $f(x,y) \in \mathbb{K}[x,y,z]$ a trivariate polynomial. If* $\deg(f) \leq 2$ *we call the zero set* $\mathcal{V}(f) := \{(x,y,z) \mid f(x,y,z) = 0\}$ *a* quadric.

Usually one chooses $\mathbb{K} = \mathbb{Q}$. Quadrics are at the front of research of three-dimensional geometric modelling. For recent results in the computational study of quadrics see [4, 13, 16]. The main difficulty in computing arrangements of quadrics exactly is that high-degree algebraic numbers are involved, even when the quadrics are defined by rational coefficients. An important task is to robustly deal with the three-dimensional intersection curves of quadrics, either by a parameterization or by a projection step. We notice that especially the projection method turns out to be a fundamental basis when computing envelopes. Let us briefly review the results of [4].

Let $\mathcal{Q} := \{q_1, \ldots, q_n\}$ be a set of n quadrics,[2] among which we select one *base quadric*, w.l.o.g. q_1. The intersection curves $q_1 \cap q_i, 2 \leq i \leq n$, induce a two-dimensional arrangement on the surface of q_1. The computation of such an arrangement is motivated by the cylindrical algebraic decomposition method [7]. By resultant computations [24] the intersection curves are projected onto the xy-plane resulting in plane algebraic curves of degree at most 4, and we call them *cut-curves*. The silhouette of q_1, defined by $\gcd(q_1, \frac{\partial}{\partial_z} q_1)$, partitions q_1 into a *lower* and an *upper* part. We also project the silhouette onto the xy-plane. The corresponding curve has degree at most 2 and is called the *silhouette-curve*. See Figure 2 for an illustration. Cut-curves can be decomposed into maximal subcurves that are x-monotone and whose interior can be assigned to the lower or upper part of q_1 uniquely. To do so we also have to split them at their intersection points with the silhouette-curve of q_1. Note that this decomposition is conservative in the sense that we may split at projected points of $q_1 \cap q_i$ where this intersection curve only touches the silhouette of q_1, but does not cross it. The implementation uses the topology of pairs of projected curves to efficiently obtain such intersections, but also to provide the predicates needed to compute (*sweep*) the projected arrangement [3]. For CGAL's `Arrangement_2` package, a valid model of the `ArrangementTraits_2` concept [21] is also provided.

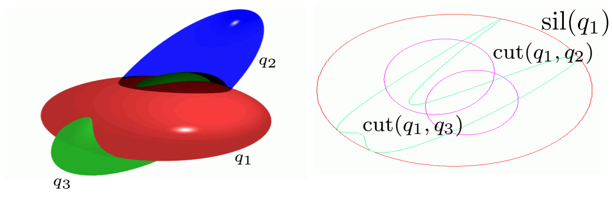


Figure 2: q_1 is the base quadric. To the right, we see the silhouette-curve of q_1 and the corresponding cut-curves.

The analysis of curves and pairs of curves is demanding. Some conditions on the coordinate system are required which can be attained by applying a shear transformation [2]. Note that in such a case we still compute an envelope, but with respect to a slightly different direction. Shearing is planned to become transparent to the user. The implementation of [4] is contained in the QUADRIX library of EXACUS.[3]

3 The Traits

The combinatorial divide-and-conquer algorithm-template is instantiated with a *traits* class [18] that encapsulates the geometric objects and operations on them. It must be a model of the `EnvelopeTraits_3` concept [15]. We provide such a class for quadrics that extends the model of the `ArrangementTraits_2` mentioned in Section 2 in the following way.

We map both basic surface types `Surface_3` and `Xy_monotone_surface_3` to the quadric type taken from QUADRIX. This may be surprising at first, since a quadric in general is not xy-monotone. However, it is only an implementation detail to simplify matters. All the operations that work on an xy-monotone surface s_i consider only the lower part of the appropriate quadric.

- Extracting xy-monotone surfaces from a general surface. The implementation is trivial according to our types choice.

Two operations benefit from silhouette- and cut-curves that we already introduced in Section 2.

- Constructing all subcurves and possibly isolated points that form the projection of the boundary of a given xy-monotone surface s_i onto the xy-plane. The decomposition of the silhouette-curve into maximal x-monotone subcurves is the natural implementation for quadrics.

- Constructing the projection of the intersection of two xy-monotone surfaces s_1, s_2 onto the xy-plane, which consists of x-monotone curves and possibly isolated points. The cut-curve already gives the projection. Again, the intersection points of the silhouette-curves with the cut-curve help to obtain a decomposition such that we can select the correct subcurves, namely the ones that can be assigned to the

[2]For convenience, we use q_i to denote both the quadratic surface as well as the underlying polynomial.

[3]`http://www.mpi-inf.mpg.de/projects/EXACUS/`

lower parts of both quadrics. Ray-shooting in the z-direction is used to compute these assignments [2].

The `EnvelopeTraits_3` concept also expects to determine the relative z-order of two xy-monotone surface patches s_1, s_2 with respect to some projected geometric objects. It distinguishes five methods for such comparisons, all of which use ray-shooting in the z-direction as a basic technique. Consider a point $(x_0, y_0) \in \mathbb{R}^2$ with $\mathcal{R}_i(x_0, y_0) := \{z \,|\, 0 = q_i(x_0, y_0, z) \in \mathbb{R}[z]\} \neq \emptyset$. Observe that $\deg(q_i(x_0, y_0, z)) \leq 2$, which means that the line parallel to the z-axis running through (x_0, y_0) intersects q_i either once or twice. Then the relative z-order of s_1 and s_2 is given by the order of $\min \mathcal{R}_1(x_0, y_0)$ and $\min \mathcal{R}_2(x_0, y_0)$. The following methods are expected by the traits concept and we mainly explain for each of them how to find a suitable point $p(x_0, y_0)$ from which to apply ray-shooting.

• Comparing the relative z-order of two surface patches s_1, s_2 over the planar region immediately above[4] a given subcurve of the projected intersection curve of s_1 and s_2. A planar point p is *above* a projected x-monotone curve c if it is in the x-range of c and lies to the left when the c is traversed from its xy-lexicographical smaller endpoint to its larger endpoint. The envelope algorithm only invokes this function if R_1 and R_2 are non-empty for all points of a ϵ-strip above c. Therefore we choose *rational* x_0, y_0, e.g., by real root isolation, such that p lies above c and the vertical planar segment starting at p and ending on c does not intersect the silhouette-curves nor any other part of the cut-curve.

• Comparing the relative z-order of two unbounded, non-intersecting surface patches s_1 and s_2 that are defined over the entire plane. We choose $p = (0, 0)$.

• Comparing the relative z-order of s_1 and s_2 over a projected x-monotone curve c where s_1 and s_2 do not intersect. If c forms a cut-curve (not of s_1 and s_2) we choose as before a rational point $p(x_0, y_0)$ in the valid ϵ-neighborhood of c. More precisely, p is chosen such that the interior of the vertical planar segment from p to c neither intersects the cut-curve of s_1 and s_2, nor the silhouette-curves, nor the curve defining c. c can also be a part of a projected boundary. In such case, its supporting curve has degree at most 2, and we are able to find y_0 of the form $y_0 := y_1 + y_2 \cdot \sqrt{y_3}$, with $y_1, y_2, y_3 \in \mathbb{Q}$ for a rational x_0 in the x-range of c. The correct relative z-order of s_1 and s_2 can then be derived using repeated squaring on rationals or with the help of number types like `leda::real` [6] or `CORE::Expr` [11].

• Comparing the relative z-order over a projected point p. It turns out that this operation is the most expensive one, because the coordinates of such points are usually defined by high-degree algebraic numbers and there is no guarantee to find a "nice" point nearby where s_1 and s_2 have the same order. It helps, that the envelope algorithm invokes this function only in some degenerate cases. Either p is isolated. Then we know that we can find x_0, y_0 that can be represented by (nested) square-root expressions [22] and we again use the number types provided by LEDA or CORE to represent the coordinates of p. If, otherwise, p lies on a silhouette-curve, then its x-coordinate has degree at most 8. Both LEDA and CORE can currently cope with such numbers. To compute the y-coordinate we have to solve a quadratic equation. The ray-shooting requires an additional square-root. Both steps still can be handled using the mentioned number types. As expected, this comparison is the most expensive one, especially when the z-coordinates are equal. We plan to adapt the bit-stream Descartes method [9] on this task and check its performance. Note that, in general, the comparison over an arbitrary point is possible using the same techniques, but not expected during the execution of the divide-and-conquer algorithm, because of the special care taken in designing the algorithm [14].

[4]The analog *below* version is also needed.

4 Results

The traits class has been implemented as part of the QUADRIX library. Since EXACUS currently merge to CGAL, all presented components will become available as packages of CGAL in one of its future releases.

We measured the running time of the divide-and-conquer algorithm when computing lower envelopes of quadrics.

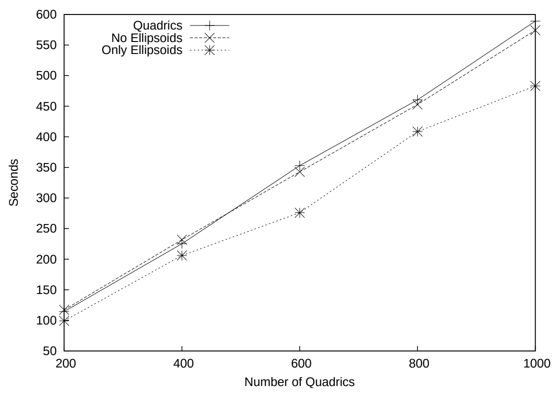


Figure 3: The running time of computing the lower envelope of sets of quadrics as a function of the number of input quadrics.

For increasing n we generated five sets of n random quadrics whose coefficients are ten-bit integers. We further distinguish between mixed quadrics, ellipsoids only, and sets that do not contain ellipsoids. Figure 3 shows the resulting running times on a 3 GHz Pentium IV machine with 2 MB of cache, averaged over all the sets of the same size. Note that computing the lower

Set	Number of comparisons over Point	Curve	Curve Side
quadrics	0 (18315)	2804 (31373)	1273 (4638)
non-ellipsoids	0 (18087)	2386 (30777)	1273 (4640)
ellipsoids	0 (22747)	1292 (38172)	1282 (3798)

Table 1: The number of times each of the three main comparison operations is invoked when computing the lower envelope of 1000 quadrics of different input sets. The *huge* number of operations invoked by a less sophisticated algorithm is shown in parenthesis.

envelope of 1000 quadrics takes less than 10 minutes, using *exact* arithmetic of LEDA. Computing the lower envelope of ellipsoids is faster.

Table 1 shows the number of times each of the three main comparison operations is invoked by the algorithm of [15]. The number of times the same operations are invoked when not using the combinatorial deductions of the algorithm is given in parenthesis. As one can see, computing lower envelopes of quadrics benefits significantly from propagation of information about the relative z order of surface patches.

References

[1] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.

[2] E. Berberich. Exact arrangements of quadric intersection curves. M.Sc. thesis, Universität des Saarlandes, Saarbrücken, Germany, January 2004.

[3] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, and N. Wolpert. EXACUS: Efficient and exact algorithms for curves and surfaces. In *Proc. 13th Annual European Symposium on Algorithms (ESA)*, volume 3669 of *LNCS*, pages 155–166, 2005.

[4] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *21st ACM Symposium on Computational Geometry (SCG'05)*, pages 99–106, June 2005.

[5] J.-D. Boissonnat and K. T. G. Dobrindt. On-line construction of the upper envelope of triangles and surface patches in three dimensions. *Comput. Geom. Theory Appl.*, 5(6):303–320, 1996.

[6] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Efficient exact geometric computation made easy. In *15th ACM Symposium on Computational Geometry (SCG'99)*, pages 341–350, 1999.

[7] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183, 1975.

[8] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12:30–53, 1994.

[9] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A descartes algorithm for polynomials with bit-stream coefficients. In *CASC*, pages 138–149, 2005.

[10] D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. *Discrete Comput. Geom.*, 12:313–326, 1994.

[11] V. Karamcheti, C. Li, I. Pechtchanski, and C. K. Yap. A core library for robust numeric and geometric computation. In *15th ACM Symposium on Computational Geometry (SCG'99)*, pages 351–359, 1999.

[12] 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.

[13] S. Lazard, L. M. Peñaranda, and S. Petitjean. Intersecting quadrics: an efficient and exact implementation. In *20th ACM Symposium on Computational Geometry (SCG'04)*, pages 419–428, 2004.

[14] M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. M.Sc. thesis, School of Computer Science, Tel Aviv University, Tel Aviv, Israel, July 2006.

[15] M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional spaces. In *Proc. 14th Annual European Symposium on Algorithms (ESA)*, volume 4168 of *LNCS*, pages 792–803, 2006.

[16] B. Mourrain, J.-P. Técourt, and M. Teillaud. On the computation of an arrangement of quadrics in 3d. *Comput. Geom. Theory Appl.*, 30(2):145–164, 2005.

[17] K. Mulmuley. An efficient algorithm for hidden surface removal, II. *J. Comput. Syst. Sci.*, 49(3):427–453, 1994.

[18] N. Myers. "Traits": A new and useful template technique. pages 451–457, 1996.

[19] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete Comput. Geom.*, 12:327–345, 1994.

[20] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, Cambridge-New York-Melbourne, 1995.

[21] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. 2006. To appear in Comput. Geom. Theory Appl.

[22] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics.* PhD thesis, Universität des Saarlandes, October 2002.

[23] C. K. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7, 1997.

[24] C. K. Yap. *Fundamental problems of algorithmic algebra.* Oxford University Press, Inc., New York, NY, USA, 2000.

An Efficient Algorithm for the InCircle Predicate among Smooth Closed Curves

Ioannis Z. Emiris* George M. Tzoumas†

Abstract

This paper concentrates on the InCircle predicate which is used for the computation of the Voronoi diagram of smooth closed curves. The predicate decides the position of a query object relative to the Voronoi circle of three given ones. We focus on (non-intersecting) ellipses but our method extends to arbitrary closed smooth curves, given in parametric representation. We describe an efficient algorithm for InCircle based on a certified numeric algorithm. The algorithm relies on a geometric preprocessing that guarantees a unique solution in a box of parametric space, where a customized subdivision-based method approximates the Voronoi circle tracing the bisectors. Our subdivision method achieves quadratic convergence by exploiting the geometric characteristics of the problem. The paper concludes with experiments showing that most instances run in less than 0.1 sec using floating-point arithmetic, on a 2.6GHz Pentium-4.

1 Introduction

The InCircle predicate is used in the incremental computation of the Voronoi diagram of curved objects [5]. Exact computation of this predicate is hard, since there may exist up to 184 (complex) tritangent circles to 3 conic curves; see [3] for a proof in the case of ellipses. In that work, the authors sketched a subdivision scheme for InCircle that worked better than generic solvers, although it had linear convergence. Notice that solving the algebraic system that defines the Voronoi circle is not enough to decide InCircle; one must take into account the query ellipse as well, which is equivalent to solving another similar system. In this approach, the computation of the Voronoi circle is the most computationally demanding task of InCircle.

The work coming closest to ours towards a complete Voronoi diagram is [4]. 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 and their software[1] works well in practice, with runtimes ranging from a few seconds to a few minutes.

In this paper, we focus on the computation of InCircle which is clearly the most challenging predicate and the only one that is not satisfactorily answered yet. Our main contribution is a new numeric algorithm that exhibits *quadratic* convergence in order to approximate the Voronoi circle, by exploiting several geometric properties of the problem. Not only is our method faster than generic solvers and other existing implementations, including the one in [3], but also allows us to decide InCircle before full precision has been achieved. An advantage of this method is that it can be generalized to arbitrary parametric curves and arcs. The algorithm proved to be very efficient, approximating the root with precision up to 10^{-15} in less than 0.1 sec, when using standard floating-point arithmetic. This work has been presented in a more complete context in [2].

We assume that an ellipse is given in rational parametric form, i.e. constructively, in terms of its rational axes, center and rotation:

$$\begin{aligned} x(t) &= x_c + \frac{-\alpha(1-w^2)t^2 - 4\beta wt + \alpha(1-w^2)}{(1+w^2)(1+t^2)}, \\ y(t) &= y_c + 2\frac{-\alpha wt^2 + \beta(1-w^2)t + \alpha w}{(1+w^2)(1+t^2)}, \end{aligned} \tag{1}$$

where $2\alpha, 2\beta$ are the lengths of the major and minor axes, respectively, $t = \tan(\theta/2) \in (-\infty, \infty)$, θ is the angle that traces the ellipse, $w = \tan(\omega/2)$, ω is the rotation angle between the major and horizontal axes and (x_c, y_c) is its center. We assume that all ellipses are parameterized in the same direction, i.e. CCW.

The *bisector* of two ellipses is the locus of points at equal distance from the two ellipses. It can be shown that the bisector of two ellipses in the parametric space, is a bivariate polynomial $B_1(t, r)$ of total degree 12, six in each variable. Each point on the bisector corresponds to the center of a circle which is bitangent to the two ellipses. For InCircle we are interested in that part of the bisector which is the locus of centers of externally bitangent circles.

Lemma 1 *Given two smooth closed curves and a point on the first, there is a bounded number of real bitangent circles, tangent at the specific point. This number is 6 for conics and is tight for ellipses.*

*Department of Informatics and Telecoms, National University of Athens

†Department of Informatics and Telecoms, National University of Athens, geotz@di.uoa.gr

[1]www.cs.technion.ac.il/~irit/

The proof of the above lemma (for ellipses) as well as more details on bisectors of ellipses can be found in [3]. For parametric bisectors, the reader may also refer to [1]. The smoothness property implies that a unique normal line is defined at every point of the curve.

Note that from the circles in the previous lemma, only one is *external* to both ellipses. We call this unique external bitangent circle the *Apollonius* circle of the two ellipses. We denote the Apollonius circle of E_t and E_r tangent at points $\hat{t}$ and $\hat{r}$ respectively by $\mathcal{A}_{tr}(\hat{t},\hat{r})$. Since $\hat{r}$ depends on $\hat{t}$, we may omit the latter and write only $\mathcal{A}_{tr}(\hat{t})$.

In the parametric space, the intersection of two bisectors involves three variables. In order to express the Voronoi circle, we consider the intersection of three bisectors by solving the system:

$$B_1(t,r) = B_2(s,t) = B_3(r,s) = 0. \qquad (2)$$

2 Voronoi circle using subdivision

We have tried the ALIAS[2] [6] built-in subdivision solvers. The naive subdivision algorithm failed to converge, while gradient based methods converged within a few seconds depending on the initial interval.

Here we present a subdivision scheme to approximate the solution of system (2) which is simpler, exhibits quadratic convergence, and exploits the system's geometric symmetry.

Lemma 2 *Consider $\mathcal{A}_{tr}(t,r)$, which means that $B_1(t,r) = 0$. If we slide this circle along the boundary of the ellipses for a sufficiently small amount, while varying its radius and keeping it tangent to t' and r' respectively, then $t' > t \Longrightarrow r' < r$.*

Proof. This lemma can be proved by contradiction. See figure 2 for a sketch of the proof. □

The basic idea of our algorithm is the following: Let $(\hat{t},\hat{r},\hat{s})$ be the solution of (2) we are looking for. Now consider the following system:

$$\begin{aligned} B_1(t_1,r_2) = B_3(r_2,s_1) = B_2(s_1,t_2) &= 0 \quad (3)\\ B_1(t_2,r_1) = B_3(r_1,s_2) = B_2(s_2,t_1) &= 0 \quad (4)\end{aligned}$$

These two systems look like (2). The difference is that we have considered $t_1 \neq t_2$ in the general case and thus we can start solving the above equations in the given order. Doing so and keeping solutions that correspond to Apollonius circles (using the geometric arguments described in [3]), leads to a construction as in fig. 1. All bitangent circles coincide with the Voronoi circle when $t_1 = \hat{t} = t_2$. Otherwise, we have found an interval $[t_1, t_2]$ that contains

[2]www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html

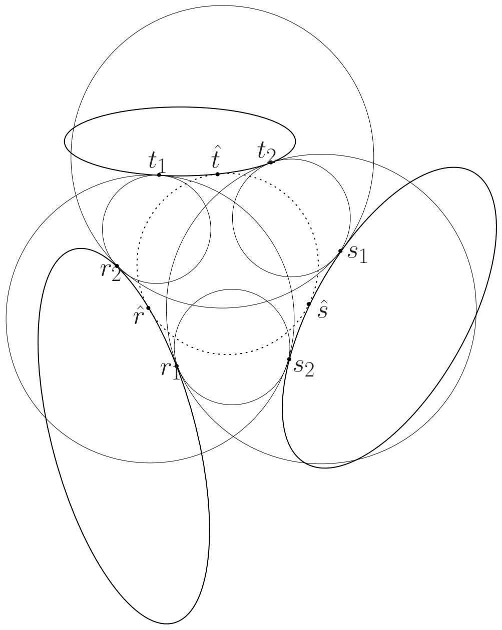


Figure 1: All-pair bitangent circles

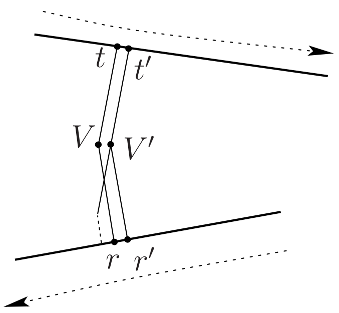


Figure 2: $t' > t \Longrightarrow r' < r$

$\hat{t}$. We can refine it by choosing a new point t'_1 inside this interval and computing $[t'_1, t'_2]$ (suppose without any harm that $t'_1 < t'_2$). As a consequence of lemma 2, t_1 approaches $\hat{t}$ from the left $\Rightarrow$ r_2 approaches $\hat{r}$ from the right $\Rightarrow$ s_1 approaches $\hat{s}$ from the left $\Rightarrow$ t_2 approaches $\hat{t}$ from the right (see fig. 1). Therefore $t'_1 \in [t_1,t_2] \Longrightarrow \hat{t} \in [t'_1,t'_2] \subset [t_1,t_2]$. Note that computing a smaller interval on dimension t allows us to compute smaller intervals on dimensions r and s as well. Therefore we maintain exactly one box that contains our solution, contrary to generic interval-arithmetic techniques that may need to maintain a large number of boxes.

2.1 Starting interval

First, we show that it is possible to find a box that contains a unique solution. Consider the half-planes l_1, l_2 which do not contain the ellipses and are bounded by the two external bitangents of E_1 and E_2.

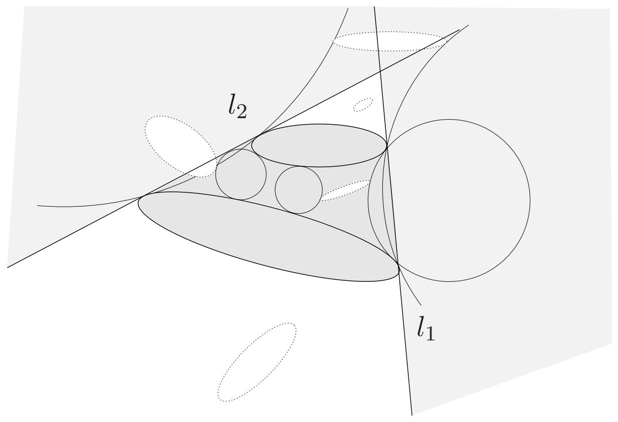


Figure 3: The cases on the number of Voronoi circles, depending on the position of the third dotted ellipse

Consider also a query ellipse E that does not intersect the other two. (cf. fig. 3) Let $|l_i| = 0$ or 1 depending on whether $E \cap l_i = \emptyset$ or not. Let C be the interior of the convex hull of E_1, E_2. Then, the number of Voronoi circles is 0, 1 or 2, namely $|l_1| + |l_2|$, if $E \cap C = \emptyset$ and $2 - |l_1| - |l_2|$ otherwise.

We can find a starting interval that contains the tangency point of the Voronoi circle by computing the external bitangents of each pair of ellipses and taking the intersection of the interior of their convex hulls. After computing the initial interval, we can pick any random point t_1 to start our algorithm.

These intervals' endpoints may not correspond to the same bitangent circle. Then, it is necessary to "normalize" them so that they contain some solution of (3) and (4), cf. fig. 1. We start from value t_1. At any given step, we have computed a value for parameter t, say $\bar{t}$ and then compute the Apollonius circle at point $\bar{r}$ of the next ellipse, that is we compute $\mathcal{A}_{tr}(\bar{t}, \bar{r})$ (by solving $B_1(\bar{t}, r)$ with respect to r). If $\bar{r} \in [r_1, r_2]$, then we update one appropriate endpoint to $\bar{r}$. If $\bar{r} \notin [r_1, r_2]$, then we update $\bar{t}$ by computing the Apollonius circle tangent to the corresponding endpoint of $[r_1, r_2]$. This process is analogous for the other parameter pairs (r, s) and (s, t).

There is also the case where two Voronoi circles exist. In this case the overall algorithm indicates which one is needed and therefore we pick a proper subinterval.

2.2 Subdivision

The subdivision algorithm in pseudo-code is as follows:

Subdivision Algorithm

```
* INPUT: Initial intervals $[t_1, t_2], [r_1, r_2], [s_1, s_2]$ that
  contain unique $(\hat{t}, \hat{r}, \hat{s})$ and $\sigma \in \mathbb{R}$, $\sigma > 0$.
* OUTPUT: Subintervals $[t_1, t_2], [r_1, r_2], [s_1, s_2]$ of the
  given ones, which contain $(\hat{t}, \hat{r}, \hat{s})$ and $t_2 - t_1 < \sigma$.
1. Start from point $t_1$ on the first ellipse and solve system (3), with the
   additional constraint that $r_2$, $s_1$, $t_2$ correspond to Apollonius circles.
   After computing $t_2$, also solve (4) in order to obtain intervals for
   $[r_1, r_2]$ and $[s_1, s_2]$ respectively. For each interval, set the left
   endpoint to be smaller than the right one, by swapping them if necessary.
2. If $t_2 - t_1 < \sigma$ then stop.
3. Set $t_1 := \frac{t_1+t_2}{2}$. Note that the midpoint of $[t_1, t_2]$ could be
   on the left or on the right of $\hat{t}$. Go to step 1.
```

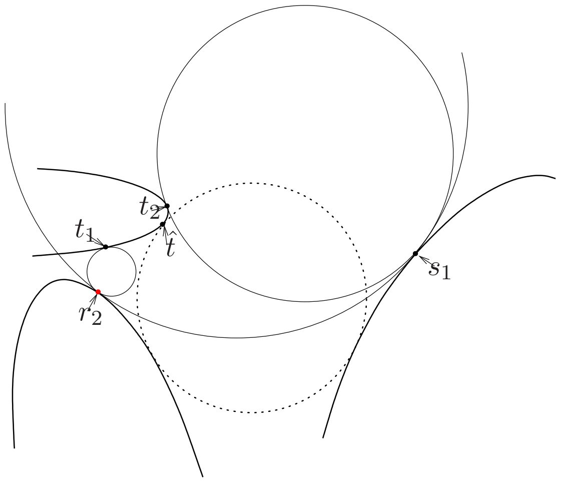


Figure 4: Computing t_2 from t_1 encloses $\hat{t}$

Eliminating r_2, s_1 from (3) yields resultant $R(t_1, t_2)$ as a bivariate polynomial equation. Similarly, eliminating r_1, s_2 from (4) yields $R(t_2, t_1)$ as a bivariate polynomial equation. Now we observe that $R(t_1, t_2) = R(t_2, t_1)$, since they have been derived from the same equations with the same coefficients. When $t_1 \neq t_2$, systems (3) and (4) express a family of circles bitangent to each pair of ellipses, as in fig. 1.

Looking at $R(t_1, t_2)$ we see that t_2 is an implicit function of t_1, say f, that is $f(t_1) = t_2$. Given value t_1, $f(t_1)$ is the value of t_2 after solving (3), shown in fig. 4. Obviously $f(\hat{t}) = \hat{t}$. From lemma 2 it follows that $t_1 < t' < \hat{t} \implies f(t_1) > f(t') > f(\hat{t})$. That is, as t_1 approaches $\hat{t}$ from the left, t_2 approaches $\hat{t}$ from the right and $\hat{t} \in [t_1, t_2]$.

Lemma 3 *In the above notation $f'(\hat{t}) = -1$.*

Proof. From the Implicit Function Theorem we have that $\frac{df}{dt_1}$ exists and (by chain rule): $\frac{df}{dt_1} = f'(t_1) = -\frac{\partial R/\partial t_1}{\partial R/\partial t_2}$ where R is the resultant polynomial. Since $R(t_1, t_2) = R(t_2, t_1)$, at point $t_1 = t_2 = \hat{t}$ we have $\left.\frac{\partial R(t_1,t_2)}{\partial t_1}\right|_{t_1=t_2=\hat{t}} = \left.\frac{\partial R(t_1,t_2)}{\partial t_2}\right|_{t_1=t_2=\hat{t}}$, therefore $f'(\hat{t}) = -\left.\frac{\partial R/\partial t_1}{\partial R/\partial t_2}\right|_{t_1=t_2=\hat{t}} = -1$. □

Theorem 4 (Convergence) *The above subdivision algorithm converges quadratically.*

Proof. For the proof, t_1, t_2 are not generic variables, but have specific values, as is the case during the execution of the algorithm. Let

$$\epsilon = |f(t_1) - t_1|$$

be the error at one iteration of the method. At the next iteration, the new error is $\epsilon' = |f(\frac{t_1+f(t_1)}{2}) - \frac{t_1+f(t_1)}{2}|$ or equivalently: $\epsilon' = |f(t_1 + \frac{f(t_1)-t_1}{2}) - t_1 - \frac{f(t_1)-t_1}{2}|$. Applying Taylor expansion around point t_1 for $f(t_1 + \frac{f(t_1)-t_1}{2})$ yields $\epsilon' = |f(t_1) + \frac{f(t_1)-t_1}{2}f'(t_1) + \frac{(f(t_1)-t_1)^2}{8}f''(\xi) - t_1 - \frac{f(t_1)-t_1}{2}|$, with ξ between t_1 and $\frac{t_1+t_2}{2}$. Notice that, from the theory of Taylor expansion, the use of ξ allows us to omit less significant terms. Now ϵ' becomes: $\epsilon' = \left|\frac{f(t_1)-t_1}{2}(1+f'(t_1)) + \frac{\epsilon^2}{8}f''(\xi)\right|$. This time we combine the Taylor expansion around point $\hat{t}$ for $f'(t_1)$ with lemma 3: $f'(t_1) = f'(\hat{t}) + (t_1 - \hat{t})f''(\xi') = -1 + (t_1 - \hat{t})f''(\xi')$, with ξ' between t_1 and $\hat{t}$, such that it allows us to omit less significant terms. Now remember that $\hat{t} \in [t_1, t_2]$ which means that $|t_1 - \hat{t}| \le |t_2 - t_1| = |f(t_1) - t_1|$. Therefore: $\epsilon' = \left|\frac{f(t_1)-t_1}{2}(t_1 - \hat{t})f''(\xi') + \frac{\epsilon^2}{8}f''(\xi)\right| \Longrightarrow$

$$\epsilon' \le \frac{\epsilon^2}{2}|f''(\xi')| + \frac{\epsilon^2}{8}|f''(\xi)|.$$

Given that $f''(t)$ is a continuous function it takes a minimum and maximum value inside $[t_1, t_2]$, therefore $|f''(\xi)|$ and $|f''(\xi')|$ are bounded by a positive constant C and eventually $\epsilon' \le \frac{5C}{8}\epsilon^2$. □

3 Conclusion

We have implemented the subdivision algorithm in C++ using the interval-arithmetic library Alias, therefore the results are certified up to floating-point precision. At each iteration we have to solve the bisector polynomial $B_1(t, r)$ for a fixed t. Fortunately Alias provides a fast univariate polynomial solver for this task. Computing the diameter of the approximating intervals during each iteration of the algorithm verified its quadratic convergence. For instance, a diameter sequence was: [3.01679, 0.978522, 0.303665, 0.0381727, 0.000628676, 1.70776e-07, 1.17684e-14]. Our experiments show that about 8 iterations are enough to approximate the roots of the system with a precision of 10^{-15} in about 80 msec on a Pentium-4 2.6GHz.

The presented algorithm can be readily generalized in order to compute the Voronoi circle of arbitrary parametric curves, as shown in fig. 5. A subdivision algorithm alone does not suffice to decide the

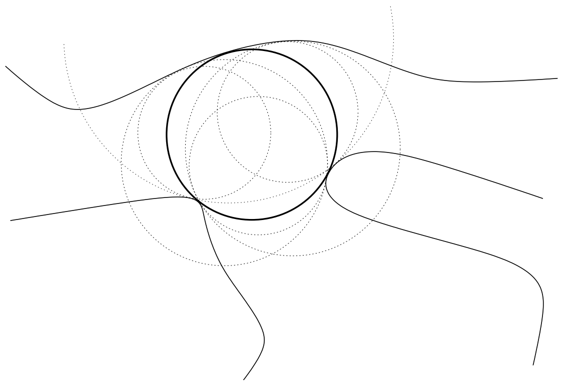


Figure 5: Voronoi circle of three arbitrary curves

predicate in degenerate cases. To handle those cases, we apply resultants and real solving [3]. Along these lines, we plan to develop C++ code that will lead to a complete CGAL[3] package.

Acknowledgments

George Tzoumas is partially supported by State Scholarship Foundation of Greece, Grant No. 4631. Both authors acknowledge 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).

References

[1] G. Elbert and M.-S. Kim. Bisector curves of planar rational curves. *Computer-Aided Design*, 30:1089–1096, 1998.

[2] I. Emiris and G. Tzoumas. A real-time and exact implementation of the predicates for the Voronoi diagram of parametric ellipses. Submitted. Manuscript available from http://www.di.uoa.gr/∼geotz/.

[3] I. Z. Emiris, E. P. Tsigaridas, and G. M. Tzoumas. The predicates for the voronoi diagram of ellipses. In *Proc. ACM 22th Annual Symposium on Computational Ceometry (SoCG)*, pages 227–236, Sedona, Arizona, USA, 2006. ACM Press.

[4] 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).

[5] M. Karavelas and M. Yvinec. Voronoi diagram of convex objects in the plane. In *Proc. ESA*, pages 337–348, 2003.

[6] J.-P. Merlet. ALIAS: an interval analysis based library for solving and analyzing system of equations. In *Systèmes d'Équations Algébriques*, Toulouse, France, 2000.

[3]www.cgal.org

Author Index