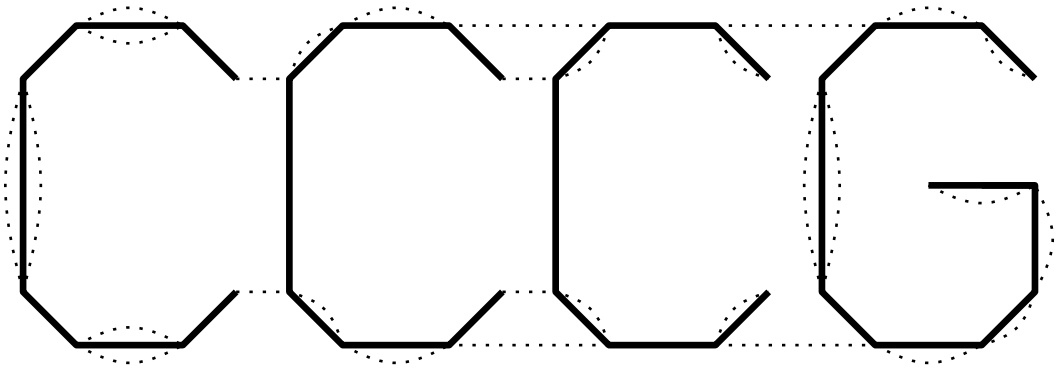
Proceedings of the

20th Annual Canadian Conference on Computational Geometry

August 13–15, 2008

McGill University, Montréal, Québec



Conference Organization

Organizing Committee

David Avis
Luc Devroye
Bruce Reed
Godfried Toussaint
Sue Whitesides

Programme Chair

Pat Morin

Programme Committee

Therese Biedl
Prosenjit Bose
Jean Cardinal
Paz Carmi
Timothy M. Chan
Siu-Wing Cheng
Sebastien Collette
Vida Dujmovic
Hazel Everett
Joachim Gudmundsson
Ferran Hurtado
John Iacono
Stefan Langerman
Henk Meijer
Jason Morrison
Asish Mukhopadhyay
Diane Souvaine
Csaba Toth
Godfried Toussaint
Norbert Zeh
Marc van Kreveld

External Reviewers

Oswin Aichholzer
Greg Aloupis
Gill Barequet
Peter Brass
David Bremner
Maike Buchin
Kevin Buchin
Luca Castelli Aleardi
Karen Daniels
J. Miguel Diaz-Banez
Karim Douieb
Laurent Dupont
Xavier Goaoc
Andreas Holmsen
Dana Jansens
Gwenaël Joret
Sylvain Lazard
Eythan Levy
Jun Luo
Maarten Löffler
Anil Maheshwari
Sylvain Petitjean
Marc Pouget
Rodrigo Silveira
Bettina Speckmann
Elias Tsigaridas

Table of Contents

Paul Erdös Memorial Lecture: Iterated Partitions of Triangles 1
Ron Graham

How Did It Start? 3
Michael I. Shamos

The Geometry of Music 5
Dmitri Tymoczko

Data Structures for Range-Aggregate Extent Queries 7
Ravi Janardan, Prosenjit Gupta, Yokesh Kumar, Michiel Smid

Searching for Frequent Colors in Rectangles 11
Marek Karpinski, Yakov Nekrich

Data Structures for Restricted Triangular Range Searching 15
Mashhood Ishaque, Diane Souvaine, Nadia Benbernou

A Generalization of Apollonian Packing of Circles 19
Gerhard Guettler, Colin Mallows

A Note on α-Drawable k-Trees 23
Svetlana Stolpner, Jonathan Lenchner, Giuseppe Liotta, David Bremner, Christophe Paul, Marc Pouget, Stephen Wismath

VC-Dimension of Visibility on Terrains 27
James King

Polygons Folding to Plural Incongruent Orthogonal Boxes 31
Ryuhei Uehara

A Class of Convex Polyhedra with Few Edge Unfoldings 35
Alex Benton, Joseph O'Rourke

Inverting Linkages with Stretch 39
Youichi Fujimoto, Mitsuo Motoki, Ryuhei Uehara

Polynomial Irreducibility Testing Through Minkowski Summand Computation 43
Deepanjan Kesh, Shashank Mehta

Convex Hull of the Union of Convex Objects in the Plane: an Adaptive Analysis 47
Jérémy Barbay, Eric Y, Chen

Polar Diagram of Moving Objects 51
Mojtaba Nouri Bygi, Mohammad Ghodsi

Isometric Morphing of Triangular Meshes 55
Prosenjit Bose, Joseph O'Rourke, Chang Shu, Stefanie Wuhrer

Computing the Stretch Factor of Paths, Trees, and Cycles in Weighted Fixed Orientation Metrics 59
Christian Wulff-Nilsen

The Focus of Attention Problem Revisited 63
Manjish Pal

On Distinct Distances Among Points in General Position and Other Related Problems 67
Adrian Dumitrescu

Monochromatic Simplices of any Volume 71
Adrian Dumitrescu, Minghui Jiang

Empty Monochromatic Triangles 75
Oswin Aichholzer, Ruy Fabila-Monroy, David Flores-Peñaloza, Thomas Hackl, Clemens Huemer, Jorge Urrutia

Draining a Polygon–or–Rolling a Ball out of a Polygon 79
Greg Aloupis, Jean Cardinal, Sébastien Collette, Ferran Hurtado, Stefan Langerman, Joseph O'Rourke

Partial Matching of Planar Polygons Under Translation and Rotation 83
Eric McCreath

Recognition of Largest Empty Orthoconvex Polygon in a Point Set 87
Subhas Nandy, Krishnendu Mukhopadhyaya, Bhargab B. Bhattacharya

Minimum Blocking Sets of Circles for a Set of Lines in the Plane 91
Natasa Jovanovic, Jan Korst, Augustus J.E.M. Janssen

Fault-Tolerant Conflict-Free Coloring 95
Mohammad Ali Abam, Mark de Berg, Sheung-Hung Poon

A Pumping Lemma for Homometric Rhythms 99
Joseph O'Rourke, Perouz Taslakian, Godfried Toussaint

Maximal Covering by Two Isothetic Unit Squares 103
Priya Ranjan Sinha Mahapatra, Partha P. Goswami, Sandip Das

Triangulating and Guarding Realistic Polygons 107
Greg Aloupis, Prosenjit Bose, Vida Dujmović, Chris Gray, Stefan Langerman, Bettina Speckmann

Computing Dehn Twists and Geometric Intersection Numbers in Polynomial Time 111
Marcus Schaefer, Eric Sedgwick, Daniel Štefankovič

An Efficient Query Structure for Mesh Refinement 115
Benoît Hudson, Duru Türkoğlu

Application of Computational Geometry to Network p-center Location Problems 119
Qiaosheng Shi, Binay Bhattacharya

Generalized Ham-Sandwich Cuts for Well Separated Point Sets 123
Jihui Zhao, William Steiger

Direct Planar Tree Transformation and Counterexample 127
Selim Akl, Kamrul Islam, Henk Meijer

Partitioning a Polygon into Two Mirror Congruent Pieces 131
Dania El-Khechen, John Iacono, Thomas Fevens, Günter Rote

The Embroidery Problem 135
Esther Arkin, George Hart, Joondong Kim, Irina Kostitsyna, Joseph Mitchell, Girishkumar Sabhnani, Steven Skiena

Computational Balloon Twisting: The Theory of Balloon Polyhedra 139
Erik D. Demaine, Martin L. Demaine, Vi Hart

On the Complexity of Point Recolouring in Geometric Graphs 143
Henk Meijer, Yurai Núñez Rodríguez, David Rappaport

Improved Bounds on the Average Distance to the Fermat-Weber Center of a Convex Object 147
Karim Abu Affash, Matthew J. Katz

On the Nonexistence of Dimension Reduction for ℓ_2^2 Metrics 151
Mohammad Moharrami, Avner Magen

The Steiner Ratio for Obstacle-Avoiding Rectilinear Steiner Trees 155
Mina Razaghpour, Anna Lubiw

Core-Preserving Algorithms 159
Hamid Zarrabi-Zadeh

Achieving Spatial Adaptivity while Finding Approximate Nearest Neighbors 163
Jonathan Derryberry, Don Sheehy, Maverick Woo, Danny Sleator

Smallest Enclosing Circle Centered on a Query Line Segment 167
Prosenjit Bose, Stefan Langerman, Sasanka Roy

On a Cone Covering Problem 171
Khaled Elbassioni, Hans Raj Tiwary

Linear-Size Meshes 175
Don Sheehy, Gary Miller, Todd Phillips

Exact Pareto-Optimal Coordination of Two Translating Polygonal Robots on a Cyclic Roadmap 179
Hamid Reza Chitsaz, Steven M. LaValle, Jason O'Kane

Open Problems from CCCG 2007 183
Erik Demaine, Joseph O'Rourke

Polygonal Chain Simplification with Small Angle Constraints 191
Ovidiu Daescu, Anastasia Kurdia

Memory Requirements for Local Geometric Routing and Traversal in Digraphs 195
Maia Fraser, Evangelos Kranakis, Jorge Urrutia

A Distributed Algorithm for Computing Voronoi Diagram in the Unit Disk Graph Model 199
Yurai Núñez Rodríguez, Henry Xiao, Kamrul Islam, Waleed Alsalih

A Framework for Multi-Core Implementations of Divide and Conquer Algorithms and its Application to the Convex Hull Problem 203
Stefan Näher, Daniel Schmitt

Guaranteed Voronoi Diagrams of Uncertain Sites 207
Jeff Sember, William Evans

The Solution Path of the Slab Support Vector Machine 211
Joachim Giesen, Madhusudan Manjunath, Michael Eigensatz

Adaptive Searching in One and Two Dimensions 215
Reza Dorrigiv, Alejandro López-Ortiz

Competitive Search for Longest Empty Intervals 219
Peter Damaschke

Erratum for "Disjoint Segments have Convex Partitions with 2-Edge Connected Dual Graphs" 223
Nadia Benbernou, Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, Mashhood Ishaque, Diane Souvaine, Csaba Toth

Paul Erdös Memorial Lecture: Iterated Partitions of Triangles

Ron Graham*

Abstract

There are many ways in which one can subdivide a triangle into smaller triangles. However, the limiting behavior when various methods of partitioning are iterated can be quite different. In this talk, we will describe some recent results for this problem, which include a few facts that we can prove, and a large set of conjectures arising from computational experiments that we cannot (yet) prove. This is joint work with Steve Butler (UCSD).

*University of California at San Diego and California Institute for Telecommunications and Information Technology

How Did It Start?

Michael I. Shamos*

Abstract

The field of computational geometry coalesced during the period 1972-78 when then-recent algorithm design techniques were applied to geometric problems. The speaker was deeply involved in the subject during those years, culminating in his Ph.D. thesis "Computational Geometry" in 1978. This talk traces the development of the discipline, starting with a straight-line graph embedding problem that arose at the National Institutes of Health in 1971. This gave rise to flood of other problems and resulted in the discovery of important unifying principles. We will cover the contributions of other researchers and look at some now-forgotten problems and results.

*Carnegie Mellon University

The Geometry of Music

Dmitri Tymoczko*

Abstract

In my talk, I will describe five properties that help make music sound tonal – or "good," to most listeners. I will then show that combining these properties is mathematically non-trivial, with the consequence that space of possible tonal musics is severely constrained. This leads me to construct higher-dimensional geometrical representations of musical structure, in which it is clear how the various properties can be combined. Finally, I will show that Western music combines these five properties at two different temporal levels: the immediate level of the chord, and the long-term level of the scale. The resulting music is hierarchically self-similar, exploiting the same basic procedures on two different time scales. In fact, one and the same twisted cubic lattice describes the musical relationships among common chords and scales.

*Princeton University

Data Structures for Range-Aggregate Extent Queries

Prosenjit Gupta* Ravi Janardan† Yokesh Kumar† Michiel Smid‡

Abstract

We consider a generalization of geometric range searching, with the goal of generating an informative "summary" of the objects contained in a query range via the application of a suitable *aggregation function* on these objects. We provide some of the first results for functions such as *closest pair*, *diameter*, and *width* that measure the extent (or "spread") of the retrieved set. We discuss a subset of our results, including closest pair queries on point-sets in the plane and on random point-sets in $\mathbb{R}^d$ ($d \geq 2$) and guaranteed-quality approximations for diameter and width queries in the plane, all for axes-parallel query rectangles.

1 Introduction

In a traditional instance of range searching, we are given a set, S, of geometric objects and wish to retrieve the subset S' contained in some query object Q (see [1] for a survey). Often, however, we desire a more informative "summary" of S', such as an order-statistic. (For instance, the average or the median price of homes (the "objects") in different neighborhoods (the "queries") of a city.) This can be done by applying, on S', an *aggregation function* such as *count*, *sum*, *min*, *max*, *mean*, *median*, *mode*, and *top-k* that is computed on a set of suitable weights defined on the objects (e.g., house prices). Prior work on such *range aggregate* query problems includes [3, 6, 8, 11, 15, 16, 17, 19, 18].

We present results for a new class of aggregation functions, including *closest pair*, *diameter*, and *width*, that measure the extent or "spread" of the objects in S'. Extent measures have applications in collision detection, shape-fitting, clustering etc. [2] and, instead of computing the measure on the entire set, it is often both sufficient and more efficient to "zoom in" on a query region and compute the measure only for this region (e.g., the closest pair of aircraft in a prescribed region of airspace).

A major challenge with extent functions is that (unlike, say, *count*) they are not decomposable efficiently, i.e., the answer for S' cannot be inferred quickly from answers for subsets that partition S'. (For instance, the closest pair in S' cannot be inferred in sublinear time from the closest pairs for subsets S'_1 and S'_2 that partition S.) Despite this, we obtain space- and query-time-efficient solutions (exact or guaranteed-quality approximations) to several range-aggregate extent queries, as summarized in Table 1. Our results are based on multilevel range trees, Voronoi Diagrams, Euclidean Minimum Spanning Trees, and generating sparse representations of candidate output sets and proving (expected) upper bounds on their size.

In prior related work, Shan *et al.* [12] gave empirical results for range-aggregate closest pair with axes-parallel query rectangles, based on R-trees. Gupta [7] gave a solution in $\mathbb{R}^1$ (resp., $\mathbb{R}^2$) with query time $O(1)$ using $O(n)$ space (resp., $O(\log^3 n)$ query time using $O(n^2 \log^3 n)$ space). Sharathkumar and Gupta [14] improved the 2D result to $O(\log^3 n)$ query time using $O(n \log^3 n)$ space and in [13, 14], showed how to decide in $O(\log^2 n)$ time and $O(n \log^{2+\epsilon} n)$ space if the closest pair in a query rectangle was within a user-specified tolerance. To our knowledge, there is no prior work on the range-aggregate diameter or width problems.

Due to space limitations, we present only results # 1, 2, 4, and 5 in Table 1 and omit proofs and most details.

2 Computing the closest pair in a query rectangle

We wish to preprocess a planar point-set S so that for a query rectangle Q, the closest pair in $S \cap Q$ can be reported. We develop our solution by successively generalizing solutions for simpler queries.

Computing the closest pair in a quadrant or vertical strip: First, let Q be a (north-east) quadrant. Let G be the graph with vertex set S where points p and q are connected by an edge iff (p, q) is the closest pair in $S \cap Q$ for some Q. G can be shown to be a plane graph and so has $O(n)$ edges, even though the number of "distinct" north-east quadrants (w.r.t. S) is $\Theta(n^2)$.

*Mentor Graphics, Hyderabad, and International Institute of Information Technology, Gachibowli, Hyderabad 500032, India. Supported, in part, by grants SR/S3/EECE/22/2004 and DST/INT/US/NSF-RPO-0155/04, Dept. of Science and Technology, Govt. of India. prosenjit_gupta@acm.org

†Dept. of Computer Science & Engg., Univ. of Minnesota, Minneapolis, MN 55455, U.S.A. {janardan,kumaryo}@cs.umn.edu. Supported, in part, by NSF grants INT–0422775 and CCF-0514950.

‡School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. michiel@scs.carleton.ca. Research supported by NSERC.

#	Problem	Query	Space
1	Closest-pair in rect. (points in $\mathbb{R}^2$)	$\log^2 n$	$n \log^5 n$
2	Closest-pair in d-rect. (random points in $\mathbb{R}^d$)	$\log^{2d} n$	$n \log^{3d-2} n$ (expected)
3	Closest-pair in disk (random points in $\mathbb{R}^2$)	$n^{2/3+\epsilon}$ (expected)	$n^{1+\epsilon}$ (expected)
4	Diameter in rect. (points in $\mathbb{R}^2$)	$k \log^5 n$ $(1 \le k \le n)$	$(n + (n/k)^2) \log^2 n$
5	Approx. diameter in rect. (points in $\mathbb{R}^2$)	$\frac{1}{\sqrt{\delta}} \log^2 n$	$\frac{1}{\sqrt{\delta}} n \log n$
		$\frac{1}{\sqrt{\delta}} \log n + \log^3 n$ (variable δ)	$n \log^2 n$
6	Approx. width in rect. (points in $\mathbb{R}^2$)	$\frac{1}{\sqrt{\delta}} \log^3 n$	$\frac{1}{\sqrt{\delta}} n \log^2 n$
7	Closest-pair "partly in" hyper-rect. (points in $\mathbb{R}^d$)	$\log^d n$	$n \log^d n$
8	Closest-pair "partly in" halfspace (points in $\mathbb{R}^d$)	$n^{1-1/d+\epsilon}$	$n^{1+\epsilon}$
9	Closest-pair "partly in" ball (points in $\mathbb{R}^d$)	$n^{1-1/(d+1)+\epsilon}$	$n^{1+\epsilon}$

Table 1: Summary of results. Query rectangles are axes-parallel. "Random points" means that the points are chosen independently and uniformly at random in the unit-square. "Partly in" means that one point in the closest pair is in the query and the other is outside. Here k is a tunable parameter, $1 \le k \le n$, δ is an error tolerance parameter $0 < \delta < 1$, $\epsilon > 0$ is a constant, and $d \ge 2$ is a constant. "Variable δ" means that δ is part of the query; otherwise, it is fixed.

For each edge $e = (p, q)$ of G, we define the planar point $r_e = (\min(p_x, q_x), \min(p_y, q_y))$, with weight $d(p, q)$, where $d(\cdot, \cdot)$ is the Euclidean distance function. Let $R = \{r_e : e \text{ is an edge in } G\}$. Our problem is equivalent to reporting the point of minimum weight in $R \cap Q$, which can be done with a 2D range tree and fractional cascading.

Lemma 1 *A planar point-set S can be stored in a structure of size $O(n \log n)$ such that the closest pair in any north-east query quadrant Q can be reported in $O(\log n)$ time.*

For a query vertical strip, we will use the following:

Lemma 2 ([14]) *S can be stored in a structure of size $O(n \log^2 n)$ such that the closest pair in any vertical query strip can be reported in $O(\log n)$ time.*

The opposite-quadrant lemma: We can localize the closest-pair between points in two opposite quadrants as follows. Let A (resp. B) be the points of S strictly in the first (resp. third) quadrant Let $A_5 \subseteq A$ (resp. $B_5 \subseteq B$) be the $\min(5, |A|)$ (resp. $\min(5, |B|)$) points that are L_∞-closest to the origin.

Lemma 3 *Let (p, q) be the closest pair in S and let $p \in A$ and $q \in B$. Then, $p \in A_5$ and $q \in B_5$.*

Computing L_∞-neighbors in a quadrant: Assume S lies strictly in the first quadrant. Given the south-west quadrant Q_q of a query point q in the first quadrant, we wish to report the $\min(5, |S \cap Q_q|)$ points in $S \cap Q_q$ that are L_∞-closest to the origin.

Let A be the set of points of S on or below the diagonal $y = x$, and let $B := S \setminus A$. Then, for each point p in A (resp. B), the L_∞-distance between p and the origin is equal to the x-coordinate (resp. y-coordinate) of p. This leads to the following structure to answer queries when q is, wlog, on or above the diagonal $y = x$: We maintain (i) an array storing the points of A sorted by x-coordinates, and (ii) an array storing the points of B sorted by x-coordinates; with each entry p in this array, we store the $\min(5, |B_p|)$ lowest points in B_p, where $B_p := \{b \in B : b_x \le p_x\}$.

Lemma 4 *A set S of points strictly in the first quadrant can be stored in a structure of size $O(n)$ such that for any query point q strictly in the first quadrant, the $\min(5, |S \cap Q_q|)$ points in $S \cap Q_q$ that are L_∞-closest to the origin can be reported in $O(\log n)$ time.*

Computing the closest pair in an anchored 3-sided rectangle: Let ℓ be a fixed vertical line. An *anchored 3-sided rectangle* Q is a rectangle of the form $Q = [a, b] \times [c, \infty)$ that intersects ℓ. Given Q, we wish to report the closest pair in $S \cap Q$.

Let T be a balanced binary search tree storing S at its leaves, by y-order. Let u be the highest node on the right spine of T such that the horizontal line ℓ'_u that separates the left and right subtrees of u intersects Q. Point $X_u = \ell \cap \ell'_u$ partitions the plane into four quadrants. Let u_1 and u_2 be the left and right children of u, respectively. Let $S^l_{u_1}$ (resp. $S^r_{u_2}$) be the points of S_{u_1} to the left (resp. right) of ℓ. (Throughout, S_v denotes the subset of S stored at the leaves of v's subtree.) Define $S^l_{u_2}$ and $S^r_{u_2}$ similarly w.r.t S_{u_2}.

Six cases exist for the closest pair (p, q) in $S \cap Q$. (1) p and q are both to the left of ℓ: Then (p, q) is the closest pair of $S^l_{u_1} \cup S^l_{u_2}$ which is in the north-east quadrant of the point (a, c). We find (p, q) by storing at u the structure of Lemma 1 for $S^l_{u_1} \cup S^l_{u_2}$. (2) p and q are both to the right of ℓ: This is is symmetric to (1). (3) p and q are both above ℓ'_u: Then (p, q) is the closest pair of $S^l_{u_2} \cup S^r_{u_2}$ which is in the strip bounded by the vertical lines through (a, c) and (b, c). We find (p, q) by storing at u the structure of Lemma 2 for $S^l_{u_2} \cup S^r_{u_2}$. (4) p (resp. q) is in the south-west (resp. north-east) quadrant of X_u: By storing at u appropriate variants of the structure of Lemma 4, and using Lemma 3, we can compute 25 point-pairs, such that (p, q) is among them. (5) p (resp. q) is in the north-west (resp. south-east) quadrant of X_u: This is symmetric to (4). (6) p and q are both below ℓ'_u: Then both points are in the subtree of u_1. We can find (p, q) by recursively querying this subtree.

Lemma 5 *S can be stored in a structure of size $O(n \log^3 n)$ the closest pair in any anchored 3-sided query rectangle Q can be reported in $O(\log^2 n)$ time.*

Computing the closest pair in an anchored rectangle: Let ℓ be a fixed horizontal line. An *anchored rectangle* Q is a rectangle of the form $[a, b] \times [c, d]$ that intersects ℓ. Given Q, we wish to report the closest pair in $S \cap Q$.

Let T be a balanced binary search tree storing S at its leaves, by x-order. Using T, we can reduce our query to four queries for anchored 3-sided rectangles and two closest pair queries for opposite quadrants.

Lemma 6 *S can be stored in a structure of size $O(n \log^4 n)$ such that the closest pair in any anchored query rectangle Q can be reported in $O(\log^2 n)$ time.*

General closest pair rectangle queries: Given a general query rectangle Q, we wish to report the closest pair in $S \cap Q$.

Let T be a balanced binary search tree storing S at its leaves, by y-order. For each internal node u of T, define the horizontal line ℓ'_u as before. We store at u the structure of Lemma 6 for S_u, to answer closest pair queries for rectangles anchored w.r.t. ℓ'_u.

Given Q, we search down T to the first node u such that ℓ'_u intersects Q. Q is anchored w.r.t. ℓ'_u, so we use the structure for S_u to find the closest pair in $S \cap Q$.

Theorem 7 *A set S of n points in the plane can be stored in a structure of size $O(n \log^5 n)$ such that for any axes-parallel query rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.*

3 Closest pair rectangle queries on randomly distributed points

Let S be a set of n points in the plane, chosen independently and uniformly at random in the unit-square. We obtain a data structure of expected size $O(n \log^4 n)$ and query time $O(\log^4 n)$ time. Though not as efficient, asymptotically, as the one in [14], our solution is simple and practical, and, moreover, extends naturally to any fixed dimension $d > 2$.

Our approach is to precompute each point-pair (p, q), with $p, q \in S$, that is the closest pair for at least one query rectangle. We then store each such pair as a weighted point in a four-dimensional range tree. The four dimensions are the x- and y-coordinates of p and of q; the weight is the Euclidean distance $d(p, q)$. Given a query rectangle Q, we find the closest pair in $S \cap Q$ by doing a range-minimum query [5] on the tree with the hyper-rectangle $Q \times Q$.

Formally, let $\mathcal{Q}$ be the (infinite) set of all axes-parallel query rectangles. Let Λ be the number of pairs (p, q), with $p, q \in S$ and p to the left of q, such that there is a rectangle $Q \in \mathcal{Q}$ for which (p, q) is a closest pair in $S \cap Q$. Then our structure uses $O(\Lambda \log^3 n)$ space and has a query time of $O(\log^4 n)$. Moreover, it can be built in time equal to that needed to compute the Λ pairs plus $O(\Lambda \log^3 n)$ time. Thus, if Λ is "small", then this will be an efficient and practical solution.

Unfortunately, Λ can be $\Theta(n^2)$ in the worst case. (Take two sets of $n/2$ points on the boundary of the unit-circle, in opposite quadrants. Every pair of points, one from each set, contributes 1 to Λ.) However, if the points of S are chosen at random then the expected value of Λ is $O(n \log n)$, as seen below.

Lemma 8 *Let (p, q) be an ordered point-pair, with $p, q \in S$ and p to the left of q. This pair contributes 1 to Λ iff the rectangle, $R(p, q)$, that has $\overline{pq}$ as a diagonal is empty, i.e., contains no point of $S \setminus \{p, q\}$.*

Thus, Λ is the number of empty rectangles $R(p, q)$ in Lemma 8. If the points of S are chosen at random, then they are "well-distributed" and there will not be many empty rectangles $R(p, q)$, as formalized by Lemma 9. (This result also appears, without proof, in [4].)

Lemma 9 *For a set S of n points that are chosen independently and uniformly at random in the unit-square, the expected value, $E(\Lambda)$, of Λ is $O(n \log n)$.*

Thus, the closest pair in $S \cap Q$ can be computed in $O(\log^4 n)$ time using a structure of expected size $O(n \log^4 n)$.

This approach generalizes to $\mathbb{R}^d$, for any fixed $d \geq 3$, based on a result from [10] that there are $O(n \log^{d-1} n/(d-1)!)$ so-called direct domination pairs (p, q) among n points drawn independently at random from the unit-hypercube in $\mathbb{R}^d$, since each such pair defines the diagonal of an empty hyper-rectangle. Thus Λ is $O(n \log^{d-1} n)$. Our problem reduces to storing each of the Λ pairs (p, q) as a weighted point in a $2d$-dimensional range tree. We conclude:

Theorem 10 *A set S of n points chosen independently and uniformly at random in the unit-hypercube in $\mathbb{R}^d$, $d \geq 2$, can be stored in a structure of expected size $O(n \log^{3d-2} n)$ so that the closest pair in any axes-parallel query rectangle can be reported in $O(\log^{2d} n)$ time.*

4 Approximating the diameter in a query rectangle

Let S be a set of n points in the plane and let δ be a fixed real, $0 < \delta < 1$. Given a query rectangle Q, we wish to report a pair of points in $S \cap Q$ whose distance is at least $(1 - \delta)$ times the diameter of $S \cap Q$.

Let $\beta(\delta) = \left\lceil \frac{2 \arcsin(1-\delta)}{\pi - 2 \arcsin(1-\delta)} \right\rceil$; $\beta(\delta) = O(1/\sqrt{\delta})$ if δ converges to zero. Our approach uses the following:

Lemma 11 ([9]) *Choose $2(\beta(\delta)+1)$ equally-spaced vectors around the unit-circle. For each vector d_i, let p_i (resp. q_i) be the point of S that is extreme in direction d_i (resp. $-d_i$). Let D be the diameter of S and let $\Delta = \max_i d(p_i, q_i)$. Then $1-\delta \leq \Delta/D \leq 1$.*

Our structure is a 2D range tree on S. With each node v in each secondary tree, we store the $O(\beta(\delta)) = O(1/\sqrt{\delta})$ point-pairs of Lemma 11 for S_v. The space used is $O((1/\sqrt{\delta})n\log n)$.

Given Q, we compute a set C of $O(\log^2 n)$ canonical nodes v in the secondary structures of the range tree such that $S \cap Q = \cup_{v\in C} S_v$. Consider any of the $O(\beta(\delta))$ direction pairs d_i and $-d_i$. We compute the extreme points of $S\cap Q$ in directions d_i and $-d_i$ by computing the extreme points among those stored with the canonical nodes for this direction pair. By Lemma 11, the farthest pair so computed over all direction pairs is an approximation to the diameter of $S \cap Q$.

Theorem 12 *A set S of n points in the plane can be stored in a structure of size $O((1/\sqrt{\delta})n\log n)$, so that for any axes-parallel query rectangle Q, a $(1-\delta)$-approximation to the diameter of $S\cap Q$ can be reported in $O((1/\sqrt{\delta})\log^2 n)$ time, where $0<\delta<1$.*

This solution extends to queries where δ comes as an input parameter along with Q. The idea is to use the range tree on S to also compute the convex hull of the $S\cap Q$ (by repeated merging of convex hulls stored at the canonical nodes) and then finding the extremal points for each of the $O((1/\sqrt{\delta})$ directions—in logarithmic time per merge and per direction. This yields the bounds shown in Table 1.

5 Approximating the width in a query rectangle

The *width* of a planar point-set S is the width of a narrowest enclosing strip. For a query rectangle Q, we wish to report a strip enclosing $S\cap Q$ of width at most $(1+\delta)$ times the width of $S\cap Q$, for a fixed real δ, $0<\delta<1$.

Let $\gamma(\delta) = \left\lceil \frac{\pi}{2\arccos(1/(1+\delta))} \right\rceil$; $\gamma(\delta) = O(1/\sqrt{\delta})$ if δ converges to zero. Our approach uses the following:

Lemma 13 ([9]) *Let $S_0 = S$ and S_i be a copy of S rotated clockwise around the origin from S_{i-1} by an angle $\pi/\gamma(\delta)$, $1 \leq i \leq \gamma(\delta)$. For $0 \leq i \leq \gamma(\delta)$, let L_i (resp. R_i) be the downward (resp. upward) convex chain dual to the upper (resp. lower) hull of the convex hull of S_i. Let ω_i^L (resp. ω_i^R) be the minimum distance between any vertex of L_i (resp. R_i) and any point vertically below (resp. above) it on R_i (resp. L_i). Let $\Omega = \min_i\{\omega_i^L, \omega_i^R\}$ and let W be the width of S. Then $1 \leq \Omega/W \leq 1+\delta$.*

Both ω_i^L and ω_i^R can be computed in $O(\log^2 n)$ time if L_i and R_i are stored in balanced binary search trees [9].

We store S in a range tree. Each node v in each secondary tree stores $1+\gamma(\delta)$ instances of the structure of Lemma 13 for S_v, where the i-th instance is a pair of balanced binary search trees built on the dual chains $L_i(v)$ and $R_i(v)$ associated with the i-th rotated copy of S_v. The space is $O((1/\sqrt{\delta})n\log^2 n)$.

Given Q, we compute, in $O(\log^2 n)$ time, a set C of $O(\log^2 n)$ canonical nodes v in the secondary structures such that $S\cap Q = \cup_{v\in C} S_v$. For each i, we merge the $L_i(v)$'s for all $v\in C$ into a single chain in $O((1/\sqrt{\delta})\log^3 n)$ time. Similarly, for the $R_i(v)$'s. From these $O(1/\sqrt{\delta})$ pairs of chains, we compute the minimum vertical distance between each pair and take the smallest as Ω.

Theorem 14 *A set S of n points in the plane can be stored in a structure of size $O((1/\sqrt{\delta})n\log^2 n)$ such that for any query rectangle Q, a $(1+\delta)$-approximation to the width of $S\cap Q$ can be reported in $O((1/\sqrt{\delta})\log^3 n)$ time, where $0<\delta<1$.*

References

[1] P. 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, pages 1–56. AMS, 1999.

[2] P. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51:606–635, 2004.

[3] P. Bose, E. Kranakis, P. Morin, and Y. Tang. Approximate range mode and range median queries. In *Proc. 22nd Symp. on Theoretical Aspects of Computer Science*, volume 3404 of *LNCS*, pages 377–388, Berlin, 2005. Springer-Verlag.

[4] S. Felsner. Empty rectangles and graph dimension. arXiv:math/0601767v1, 2006. http://arxiv.org/abs/math/0601767v1.

[5] H. Gabow, J.Bentley, and R. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 135–142, 1984.

[6] S. Govindarajan, P. Agarwal, and L. Arge. CRB-tree: An efficient indexing scheme for range aggregate queries. In *Proc. 9th Intl. Conf. on Database Theory*, pages 143–157, 2003.

[7] P. Gupta. Algorithms for range-aggregate query problems involving geometric aggregation operations. In *Proc. 16th Intl. Symp. on Algorithms and Computation*, volume 3827 of *LNCS*, pages 892–901, Berlin, 2005. Springer-Verlag.

[8] S. Hong, B. Song, and S. Lee. Efficient execution of range-aggregate queries in data warehouse environments. In *Proc. 20th Intl. Conf. on Conceptual Modeling*, volume 2224 of *LNCS*, pages 299–310, Berlin, 2001. Springer-Verlag.

[9] R. Janardan. On maintaining the width and diameter of a planar point-set online. *Intl. Journal of Computational Geometry & Applications*, 3:331–344, 1993.

[10] R. Klein. Direct dominance of points. *Intl. Journal of Computer Mathematics*, 19:225–244, 1987.

[11] D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12:1–17, 2005.

[12] J. Shan, D. Zhang, and B. Salzberg. On spatial-range closest-pair query. In *Proc. 8th Intl. Symp. on Spatial and Temporal Databases*, volume 2750 of *LNCS*, pages 252–269, Berlin, 2003. Springer-Verlag.

[13] R. Sharathkumar and P. Gupta. Range-aggregate proximity detection for design rule checking in VLSI layouts. In *Proc. 18th Canadian Conf. on Computational Geometry*, pages 151–154, 2006.

[14] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. IIIT/TR/2007/80, Intl. Inst. of Info. Tech., Hyderabad, www.iiit.net/techreports/2007_80.pdf, 2007.

[15] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2002.

[16] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *IEEE Trans. on Knowledge and Data Engineering*, 16:1555–1570, 2004.

[17] D. Zhang, A. Markowetz, V. Tsotras, D. Gunopulos, and B. Seeger. Efficient computation of temporal aggregates with range predicates. In *Proc. 20th Symp. on Principles of Database Systems*, pages 237–245, 2001.

[18] D. Zhang, V. Tsotras, and D. Gunopulos. Efficient aggregation over objects with extent. In *Proc. 21st Symp. on Principles of Database Systems*, pages 121–132, 2002.

[19] D. Zhang and V. J. Tsotras. Improving min/max aggregation over spatial objects. *VLDB Journal*, 14:170–181, 2005.

Searching for Frequent Colors in Rectangles

Marek Karpinski* Yakov Nekrich†

Abstract

We study a new variant of colored orthogonal range searching problem: given a query rectangle Q, all colors c, such that at least a fraction τ of all points in Q are of color c, must be reported. We describe several data structures for that problem that use pseudo-linear space and answer queries in poly-logarithmic time.

1 Introduction

The colored range reporting problem is a variant of the range searching problem in which every point $p \in P$ is assigned a color $c \in C$. The set of points P is preprocessed in the data structure so that for any given rectangle Q all distinct *colors* of points in Q can be reported efficiently. In this paper we consider a variant of this extensively studied problem in which only frequently occurring colors must be reported.

We say that a color $c \in C$ τ-*dominates* rectangle Q if at least a τ-fraction of points in Q are of that color: $|\{\, p \in P \cap Q \mid \mathrm{col}(p) = c \,\}| \geq \tau |P \cap Q|$, where $\mathrm{col}(p)$ denotes the color of point p. We consider several data structures that allow us to report colors that dominate Q [1].

Motivation Standard colored range reporting problem arises in many applications. Consider a database in which every object is characterized by several numerical values (point coordinates) and some attribute (color). For instance the company database may contain information about age and salary of each employee. The attribute associated with each employee is his or her position. The query consists in reporting all different job types for all employees with salary between 40.000 and 60.000 who are older than 40 and younger than 60 years old. Colored range reporting also occurs naturally in computational biology applications: each amino acid is associated with certain attributes (hydrophobic, charged, etc.). We may want to report different attributes associated with amino acids in certain range [9].

However, in certain applications we are not interested in all attributes that occur in the query range. Instead, we may be interested in reporting the *typical* attributes. For instance, in the first example above we may wish to know all job types, such that at least a fraction τ of all employees with a given salary and age range have a job of this type. In this paper we describe data structures that support such and similar queries.

Related Work. Traditional colored range reported queries can be efficiently answered in one, two, and three dimensions. There are data structures that use pseudo-linear (i.e. $n \log^{O(1)} n$) space and answer one- and two-dimensional colored range reporting queries in $O(\log n + k)$ time [7], [8] and three-dimensional colored queries in $O(\log^2 n + k)$ time [7], where k is the number of colors. A semi-dynamic data structure of Gupta *et al.* [7] supports two-dimensional queries in $O(\log^2 n + k)$ time and insertions in $O(\log^3 n)$ amortized time. Colored orthogonal range reporting queries in d dimensions can be answered in $O(\log n + k)$ time with a data structure that uses $O((n^{1+\varepsilon}))$ space [1], but no efficient pseudo-linear space data structure is known for $d > 3$.

De Berg and Haverkort [4] consider a variant of the colored range searching in which only *significant* colors must be reported. A color c is *significant* in rectangle Q if at least a fraction τ of points of that color belong to Q, $|\{\, p \in Q \cap P \mid \mathrm{col}(p) = c \,\}| \geq \tau |\{p \in P \mid \mathrm{col}(p) = c \,\}|$. For $d = 1$, de Berg and Haverkort [4] describe a linear space data structure that answers queries in $O(\log n + k)$ time, where k is the number of significant colors. For $d \geq 2$ significant queries can be answered approximately: in $O(\log n + k)$ time we can report a set of colors such that each color in a set is $(1-\varepsilon)\tau$-significant for a fixed constant ε and all τ-significant colors are reported. The only known data structure that efficiently answers exact significance queries uses cubic space [4].

Our Results In this paper we show that we can find domination colors in an arbitrary d-dimensional rectangle in poly-logarithmic time using a pseudo-linear space data structure.

- We describe a static $O((1/\tau)n)$ space data structure that supports one-dimensional queries in $O((1/\tau) \log n \log\log n)$ time. A static $O((1/\tau)n \log\log n)$ space data structure supports one-dimensional domination queries in $O((1/\tau) \log n)$ time.
- In the case when all coordinates are integers bounded by U, there is a $O((1/\tau)n)$ space static data structure that supports one-dimensional dom-

*Dept. of Computer Science, University of Bonn. Email marek@cs.uni-bonn.de.

†Dept. of Computer Science, University of Bonn. Email yasha@cs.uni-bonn.de.

[1] Further we will assume that parameter τ is fixed and simply say that a color c dominates rectangle Q .

ination queries in $O((1/\tau)\log\log n \log\log U)$ time
- There is a dynamic $O((1/\tau)n)$ space data structure that supports one-dimensional domination queries and insertions in $O((1/\tau)\log n)$ time and deletions in $O((1/\tau)\log n)$ amortized time. We can reduce the update time to (amortized) $O(\log n)$ by increasing the space usage to $O((1/\tau)n\log n)$
- There is a data structure that supports domination queries in d dimensions in $O((1/\tau)\log^d n)$ time and uses $O((1/\tau)n\log^{d-1} n)$ space
- There is a dynamic data structure that answers domination queries in d dimensions in $O((1/\tau)\log^{d+1} n)$ time, uses $O((1/\tau)n\log^{d-1} n)$ space, and supports insertions in $O((1/\tau)\log^{d+1} n)$ time and deletions in $O((1/\tau)\log^{d+1} n)$ amortized time

We describe static and dynamic data structures for one-dimensional domination queries in sections 2 and 3. Data structures for multi-dimensional domination queries are described in section 4.

2 Static Domination Queries in One Dimension

The following simple property plays an important role in all data structures for domination queries.

Observation 1 *If $Q = Q_1 \cup Q_2$, $Q_1 \cap Q_2 = \emptyset$, and color c is dominant in Q, then either c is dominant in Q_1 or c is dominant in Q_2.*

Due to this property a query on a set Q can be *decomposed* into queries on some disjoint sets $Q_1, \ldots, Q_p$ such that $\cup Q_i = Q$ and p is a constant: we find the dominating colors for each Q_i and for each color c that dominates some Q_i we determine whether c dominates Q by a range counting query.

Our data structure is based on the same approach as exponential search trees [2]. Let P be the set of all points. In one-dimensional case we do not distinguish between a point and its coordinate. P is divided into $\beta(n)$ intervals $I_1, \ldots, I_{\beta(n)}$ so that each $P_i = P \cap I_i$ contains between $n^{2/3}/2$ and $2n^{2/3}$ points and $\beta(n) = \Theta(n^{1/3})$. Let l_i and r_i denote the left and right bounds of interval I_i. For each $1 \le i \le j \le \beta(n)$, the list L_{ij} contains the set of colors that dominate $[l_i, r_j]$. We denote by n_{ij} the total number of points in $[l_i, r_j]$.

Each interval I_i is recursively subdivided in the same manner: an interval that contains m points is divided into $\beta(m)$ subintervals and each subinterval contains between $m^{2/3}/2$ and $2m^{2/3}$ points. If some interval I_j is divided into $I_{j,1}, \ldots, I_{j,\beta(m)}$, then we say that I_j is a parent of $I_{j,i}$ ($I_{j,i}$ is a child of I_j). The tree $\mathcal{T}$ reflects the division of intervals into sub-intervals: each tree node u corresponds to an interval I_u and a node u is a child of v if and only if I_u is a child of I_v. The root of $\mathcal{T}$ corresponds to P and leaves of $\mathcal{T}$ correspond to points of P. The height of $\mathcal{T}$ is $O(\log\log n)$. For every color c, we also store all points of color c in a data structure that supports range counting queries.

Consider a query $Q = [a, b]$. Let l_a and l_b be the leaves of $\mathcal{T}$ in which a and b are stored, and let q be the lowest common ancestor of l_a and l_b. The search procedure visits all nodes on the path from l_a to q (l_b to q); for each visited node u we construct the set of colors S_u, such that every $c \in S_u$ dominates $I_u \cap [a, b]$. We also compute the total number of points in $I_u \cap [a, b]$. Let u be the currently visited node of $\mathcal{T}$ situated between l_b and q, and suppose that the node v visited immediately before u is the $(i+1)$-st child of u. Due to Observation 1 only colors stored in L_{1i} and S_v may dominate $I_u \cap Q$. For each color c in $L_{1i} \cup S_v$ we count how many times it occurs in $I_u \cap Q$ using the range counting data structure for that color. Thus we can construct S_u by answering at most $2/\tau$ counting queries. Nodes between l_a and q are processed in the same way. Finally, we examine all colors in sets S_p and S_r and list L_{ij} of the node q, where p and r are nodes on the paths from q to l_a and l_b respectively, p is the i-th child of q, and r is the j-th child of q. The search procedure visits $O(\log\log n)$ nodes and answers $O((1/\tau)\log\log n)$ counting queries. Hence, queries can be answered in $O(\log n \log\log n)$ time.

If an interval I contains m points, then all lists L_{ij} contain $O(m^{2/3})$ elements. Data structures for range counting queries use $O(n)$ space. Therefore the space usage of our data structure is $O(n)$.

We can reduce the query time to $O(\log n)$ by storing range counting data structures for each interval: for every interval I_u and every color c, such that $\{\, p \in P \cap I_u \mid \mathrm{col}(p) = c \,\} \neq \emptyset$, we store a data structure that supports range counting queries in time $O(\log |I_u|)$. The total number of colors in all intervals I_u for all nodes u situated on the same level of tree $\mathcal{T}$ does not exceed the number of points in P. Therefore the total number of elements in all range counting data structures is $O(n\log\log n)$. The query is processed in the same way as described above. We must answer $O((1/\tau))$ counting queries on I_q, $O((1/\tau))$ range counting queries on children of I_q, $O((1/\tau))$ range counting queries on children of children of I_q, etc. Therefore the query time is $O((1/\tau)(\log(|I_q|) + \log(|I_q|^{2/3}) + \log(|I_q|^{4/9}) + \ldots)) = O((1/\tau)\sum(2/3)^i \log n) = O((1/\tau)\log n)$.

We obtain the following result

Theorem 1 *There exists a $O((1/\tau)n\log\log n)$ space data structure that supports one-dimensional domination queries in $O((1/\tau)\log n)$ time. There exists a $O((1/\tau)n)$ space data structure that supports one-dimensional domination queries in $O((1/\tau)\log n\log\log n)$ time.*

In the case when all point coordinates are integers bounded by a parameter U we can easily answer one-

dimensional counting queries in $O(\log\log U)$ time using the van Emde Boas data structure [6]. As shown above, a domination query can be answered by answering $O((1/\tau)\log\log n)$ counting queries; hence, the query time is $O((1/\tau)\log\log n\log\log U)$. Since it is not necessary to store range counting data structures for each interval, all range counting data structures use $O(n)$ space.

Theorem 2 *There exists a $O((1/\tau)n)$ space data structure that supports one-dimensional domination queries in $O((1/\tau)\log\log U\log\log n)$ time.*

3 Dynamic Domination Queries in One Dimension

Let T be a binary tree on the set of all $p \in P$. With every internal node v we associate a range $rng(v) = [l_v, r_v)$, where l_v is the leftmost leaf descendant of v and r_v is the leaf that follows the rightmost leaf descendant of v. T is implemented as a balanced binary tree, so that insertions and deletions are supported in $O(\log n)$ time and the tree height is $O(\log n)$. In each node v we store the number of its leaf descendants, and the list L_v; L_v contains all colors that dominate $rng(v)$. For every color c in L_v we also maintain the number of points of color c that belong to $rng(v)$. For each color c there is also a data structure that stores all points of color c and supports one-dimensional range counting queries.

A query $Q = [a, b]$ is answered by traversing the paths from l_a to q and from l_b to q, where l_a and l_b are the leaves that contain a and b respectively, and q is the lowest common ancestor of a and b. As in the previous section, in every visited node u the search procedure constructs the set of colors S_u, such that every $c \in S_u$ dominates $rng(v) \cap [a, b]$. Suppose that a node v on the path from l_b to q is visited and let u be the child of v that is also on the path from l_b to q. If u is the left child of v, then $rng(v) \cap [a, b] = rng(u) \cap [a, b]$ and $S_v = S_u$. If u is the right child of v, then $rng(v)\cap[a, b] = rng(w)\cup(rng(u)\cap[a, b])$ where w is a sibling of u. Colors that dominate $rng(w)$ are stored in L_w; we know colors that dominate $(rng(u) \cap [a, b])$ because u was visited before v and S_u is already constructed. Hence, we can construct S_v by examining each color $c \in L_w \cup S_u$ and answering the counting query for each color. Since one-dimensional dynamic range counting can be answered in $O(\log n)$ time, we spend $O((1/\tau)\log n)$ time in each tree node. Nodes on the path from l_a to q are processed in a symmetric way. Finally we examine the colors stored in S_{q_1} and S_{q_2}, where q_1 and q_2 are the children of q, and find the colors that dominate $rng(q) \cap [a, b] = [a, b]$.

When a new element is inserted(deleted), we insert a new leaf l into T (remove l from T). For every ancestor v of l, the list L_v is updated.

After a new point of the color c_p is inserted, the color c_p may dominate $rng(v)$ and colors in L_v may cease to dominate $rng(v)$. We may check whether c_p must be inserted into L_v and whether some colors $c \in L_v$ must be removed from L_v by performing at most $(1/\tau) + 1$ range counting queries. Since a new point has $O(\log n)$ ancestors, insertions are supported in $O((1/\tau)\log^2 n)$ time.

When a point of color c_p is deleted, we may have to delete the color c_p from L_v. We can test this by performing one counting query. However, we may also have to insert some new color c into L_v because the number of points stored in descendants of the node v decreased by one. To implement this, we store the set of candidate colors L'_v; L'_v contains all colors that $(\tau/2)$-dominate $rng(v)$. For each color $c \in L'_v$ we test whether c became a τ-dominating color after deletion. When the number of leaf descendants of the node v decreased by a factor 2, we re-build the list L'_v. If P_v is the set of leaf descendants of v (that is, points that belong to $rng(v)$), then we can construct the set of distinct colors that occur in P_v in $O(|P_v|\log(|P_v|))$ time. We can also find the sets of colors that τ-dominate and $(\tau/2)$-dominate $rng(v)$ in $O(|P_v|\log(|P_v|))$ time. Since we re-build L'_v after a sequence of at least $|P_v/2|$ deletions, re-build of some L'_v incurs an amortized cost $O(\log n)$. Every deletions may affect $O(\log n)$ ancestors; hence, deletions are supported in $O(\log^2 n)$ amortized time.

We can speed-up the update operations by storing in each tree node u the set of distinct colors in P_u, denoted by C_u. For each color $c \in C_u$, we store how many times points with color c occur in P_u. When a new point p is inserted/deleted, we can update C_v for each ancestor v of p in $O(1)$ time. Using C_v, we can decide whether a given new color must be inserted into L_v in $O(1)$ time. Using C_v we can also re-build L'_v in $O(|C_v|) = O(|P_v|)$ time. Hence, we can support insertions in $O((1/\tau)\log n)$ time and deletions in $O(\log n)$ time with help of lists C_v. The total number of elements in all C_v is $O((1/\tau)n\log n)$.

Thus we obtain the following

Theorem 3 *There exists a $O((1/\tau)n)$ space data structure that supports one-dimensional domination queries and insertions in $O((1/\tau)\log^2 n)$ time and deletions in $O((1/\tau)\log^2 n)$ amortized time. There exists a $O((1/\tau)n\log n)$ space data structure that supports one-dimensional domination queries and insertions in $O((1/\tau)\log n)$ time and deletions in $O((1/\tau)\log n)$ amortized time.*

4 Multi-Dimensional Domination Queries

We can extend our data structures to support d-dimensional queries for an arbitrary constant d using the standard range trees [3] approach. We describe how we can construct a d-dimensional data structure if we know how to construct a $(d-1)$-dimensional data

structure. A range tree T_d is constructed on the set of d-th coordinates of all points. An arbitrary interval $[a_d, b_d]$ can be represented as a union of $O(\log n)$ node ranges. Hence, an arbitrary d-dimensional query $Q = Q^{d-1} \times [a_d, b_d]$ can be represented as a union of $O(\log n)$ queries $Q_1, \ldots, Q_t$, where $t = O(\log n)$ and $Q_i = Q_{d-1} \times rng(v_i)$ for some node v_i of T. In each node v of T we store a $(d-1)$-dimensional data structure D_v that contains the first $d-1$ coordinates of all points whose d-th coordinates belong to $rng(v)$. D_v supports modified domination queries in $d-1$ dimensions: for a $(d-1)$-dimensional query rectangle Q, D_v outputs all colors that dominate $Q \times rng(v)$. Using D_{v_i} we can find (at most $(1/\tau)$) colors that dominate $Q_i = Q' \times rng(v)$. Since Q is a union of $O(\log n)$ ranges Q_i, we can identify a set $\mathcal{C}$ that contains $O((1/\tau)\log n)$ candidate colors by answering $O(\log n)$ modified $(d-1)$-dimensional domination queries. As follows from Observation 1, only a color from $\mathcal{C}$ can dominate Q. Hence, we can identify all colors that τ-dominate Q by answering $O((1/\tau)\log n)$ d-dimensional range counting queries. Thus the query time for d-dimensional queries can be computed with the formula $q(n,d) = O(\log n)q(n,d-1) + O((1/\tau)\log n)c(n,d)$, where $q(n,d)$ is the query time for d-dimensional domination queries and $c(n,d)$ is the query time for d-dimensional counting queries. We can answer d-dimensional range counting queries in $O(\log^{d-1} n)$ time and $O(n \log^{d-1} n)$ space [5]. We can answer one-dimensional domination queries in $O(\log n)$ time by Theorem 1. Therefore d-dimensional domination queries can answered in $O((1/\tau)\log^d n)$ time.

We can apply the reduction to rank space technique [10, 5] and replace all point coordinates with labels from $[1, n]$. This will increase the query time by an additive term $O(\log n)$. Since point coordinates are bounded by n, we can apply Theorem 2 and answer one-dimensional domination queries in $O((\log\log n)^2)$ time using a $O(n)$ space data structure. Since the space usage grows by a $O(\log n)$ factor with each dimension, our data structure uses $O(n \log^{d-1} n)$ space.

Theorem 4 *There exists a data structure that supports domination queries in d dimensions in $O((1/\tau)\log^d n)$ time and uses $O(n \log^{d-1} n)$ space.*

The same range trees approach can be also applied to the dynamic one-dimensional data structure for domination queries. Since one-dimensional dynamic domination queries can be answered in $O((1/\tau)\log^2 n)$ time and dynamic range counting queries can be answered in $O(\log^d n)$ time and $O(n \log^{d-1} n)$ space, d-dimensional domination queries can be answered in $O(\log^{d+1} n)$ time, and the space usage is $O((1/\tau)n \log^{d-1} n)$. Since updates are supported in $O(\log^2 n)$ (amortized) time in one-dimensional case and update times grow by $O(\log n)$ factor with each dimension, d-dimensional data structure supports updates in $O(\log^{d+1} n)$ (amortized) time.

Theorem 5 *There is a dynamic data structure that answers domination queries in d dimensions in $O((1/\tau)\log^{d+1} n)$ time, uses $O((1/\tau)n \log^{d-1} n)$ space, and supports insertions in $O((1/\tau)\log^{d+1} n)$ time and deletions in $O((1/\tau)\log^{d+1} n)$ amortized time.*

Conclusion

We presented data structures for a new variant of colored range reporting problem. Our data structures use pseudo-linear space and report all τ-dominating colors in poly-logarithmic time in the case when the parameter τ is small, *i.e.* constant or poly-logarithmic in n. It would be interesting to construct efficient data structures for larger values of τ.

Acknowledgment

We would like to thank Mark de Berg and Herman Haverkort for stimulating discussions and for suggestions concerning the new variant of colored range searching problem.

References

[1] P. K. Agarwal, S. Govindarajan, S. Muthukrishnan, *Range Searching in Categorical Data: Colored Range Searching on Grid*, Proc. ESA 2002, pp. 17-28.

[2] A. Andersson, M. Thorup, *Dynamic Ordered Sets with Exponential Search Trees*, J. ACM 54(3): 13(2007).

[3] J. L. Bentley, *Multidimensional Divide-and-Conquer*, Commun. ACM 23(1980), 214-229.

[4] M. de Berg, H. J. Haverkort, *Significant-Presence Range Queries in Categorical Data*, Proc. WADS 2003, pp. 462-473.

[5] B. Chazelle, *A Functional Approach to Data Structures and its Use in Multidimensional Searching*, SIAM J. on Computing 17(1988), 427-462.

[6] P. van Emde Boas, *Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space*, Inf. Process. Lett. 6(3): 80-82 (1977).

[7] D. Gupta, R. Janardan and M. Smid, *Further Results on Generalized Intersection Problems: Counting, Reporting, and Dynamization*, J. of Algorithms 19(1995), 282-317.

[8] R. Janardan, M. Lopez, *Generalized Intersection Searching Problems*, Internat. J. Comput. Geom. Appl. 3 (1993), 39-69.

[9] N. Madhusudhanan, P. Gupta, A. Mitra, *Efficient Algorithms for Range Queries in Protein Sequence Analysis*, Proc. CCCG 2005, pp. 146-149.

[10] M. H. Overmars, *Efficient Data Structures for Range Searching on a Grid*, J. Algorithms 9(1988), 254-275.

Data Structures for Restricted Triangular Range Searching

Nadia M. Benbernou* Mashhood Ishaque† Diane L. Souvaine‡

Abstract

We present data structures for triangular emptiness and reporting queries for a planar point set, where the query triangle contains the origin. The data structures use near-linear space and achieve polylogarithmic query times.

1 Introduction

Simplex range searching (emptiness, reporting, counting) [1] is a fundamental problem in computational geometry. Given a set S of n points, a simplex emptiness query asks whether a given query simplex contains a non-empty subset of S. A simplex reporting query asks for a report of all points of S inside the query simplex, and a simplex counting query asks for the total number of such points.

In this paper we consider the restricted version of simplex (triangular) emptiness and reporting queries for points in a plane, where each query triangle contains the origin. Since we can triangulate any triangle containing origin into three triangles such that each triangle has one vertex incident on origin, we can assume wlog that each query triangle has one vertex at origin. The same idea works for any convex polygon containing the origin, but the number of query triangles is equal to the number of sides in the polygon.

1.1 Our Results

We present near-linear-space data structures for the following queries. All reporting queries incur an additional cost of $O(k)$ where k is the number of objects to be reported.

- Restricted triangular emptiness and reporting queries in $O(\lg^2 n)$ time. [Section 2]
- Restricted triangular emptiness and reporting queries in $O(2^{1/\epsilon} \lg n)$ time. [Section 3]
- Restricted triangular emptiness queries in $O(\lg n)$ time. [Section 4]
- Triangular emptiness and reporting queries in $O(\text{polylog } n)$ time with high probability, where the vertices of the query triangle are randomly chosen from the given point set. [Section 5]
- Ray intersection detection and reporting queries among an arrangement of n lines in $O(\text{polylog } n)$ time. [Section 6]
- Non-orthogonal square emptiness and reporting queries in $O(\text{polylog } n)$ time. [Section 7]

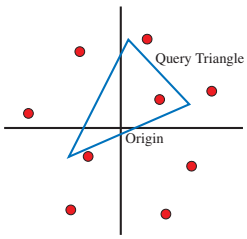


Figure 1: Restricted Triangular Range Queries

1.2 Related Results

Chazelle [5] showed that in the arithmetic model $\Omega(n^{1/2})$ time is needed to answer a triangular counting query using linear space. Similarly Brönnimann *et al.* [4] gave a lower bound of $\Omega(n^{1/3})$ for halfplane range counting in semigroup arithmetic model. For triangular reporting queries, Chazelle and Rosenberg [9] showed that, on a pointer machine, a query time of $O(n^\delta + k)$ requires $\Omega(n^{2(1-\delta)-\epsilon})$ space. Consequently polylogarithmic query time requires close to quadratic space. Although halfplane range counting is as hard as triangular range counting, halfplane reporting is significantly easier than triangular reporting. Chazelle *et al.* [6] gave a linear-space data structure for halfplane range reporting that achieves $O(\lg n + k)$ query time. The data structure maintains nested (peeling) convex layers for the given point set. Similarly for halfplane emptiness queries, a linear-space data structure that maintains convex hull of the given point set allows the queries to be answered in $O(\lg n)$ time.

The best known data structure for triangular emptiness (reporting) that uses $O(n \lg n)$ space, achieves a query time of $O(n^{1/2+\epsilon})$ (additional $O(k)$ for reporting). The data structure is based on Matoušek's technique of simplicial partitioning with low crossing number [15].

Using near quadratic space it is possible to support triangular range searching in polylogarithmic time.

*Massachusetts Institute of Technology, nbenbern@mit.edu. Partially supported by AFOSR grant FA9550-07-1-0538.

†Tufts University, mishaq01@cs.tuft.edu. Partially supported by NSF grant CCF-0431027.

‡Tufts University, dls@cs.tuft.edu. Partially supported by NSF grant CCF-0431027.

Chazelle *et al.* [10] gave a $O(n^{2+\epsilon})$ space data structure that supports reporting queries in $O(\lg n + k)$ time. Goswami *et al.* [14] presented a $O(n^2)$ space data structure that can support triangular reporting queries in $O(\lg^2 n + k)$ and triangular counting (and hence emptiness) queries in $O(\lg n)$ time.

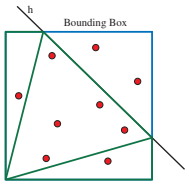


Figure 2: Halfplane Queries to Triangular Queries

The best lower bounds known for the triangular emptiness queries comes from the lower bounds for halfplane emptiness queries. There is a simple reduction from halfplane range queries to triangular range queries (see Figure 2). Erickson [13] showed that for any data structure supporting halfplane emptiness queries will have a lower bound of $\Omega(\lg n)$ for query, $\Omega(n)$ for space, and $\Omega(n \lg n)$ for preprocessing. While the bounds for halfplane emptiness are tight, there is no matching upper bound for triangular emptiness queries.

2 Simple Data Structures with $O(\lg^2 n)$ Query

Given a set of n points in the plane, sort the points rotationally in counter-clockwise order around the origin. Assign as ID to each point its order in the sorted list of points. Now consider any single wedge formed by two rays emanating from the origin. Let i be the first and j be the last point inside the wedge that would be hit if we were to sweep rotationally around origin using a ray that goes from the right boundary of the wedge to the left boundary. Observe that all the points inside the wedge are consecutive (with wrap-around) in the sorted order, see Figure 3. For any given wedge, we can find the points i and j in $O(\lg n)$ time. For any triangular emptiness/reporting query Δ_{abc} with a vertex, b, coincident with the origin, we can extend the two sides as rays $\overrightarrow{ba}$ and $\overrightarrow{bc}$ away from the origin to form a wedge. Now if we had a halfplane emptiness/reporting data structure over the points in this wedge, we could answer triangular emptiness and reporting queries by querying the halfplane bounded by $\overleftrightarrow{ac}$.

A naïve data structure for restricted triangular emptiness queries would be to store a convex hull for each pair of indicies (i, j), supporting queries in $O(\lg n)$ time, and using $O(n^3)$ space. The preprocessing time would be $O(n^3)$ because we can compute the convex hull in linear time for points in sorted order. For triangular reporting we would store nested convex layers instead of convex hulls. The query time would be $O(\lg n + k)$, space would be $O(n^3)$, and the preprocessing would be $O(n^3 \lg n)$.

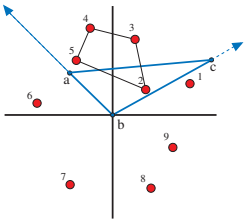


Figure 3: Points Sorted Around the Origin

Since the halfplane emptiness and reporting queries are decomposable, we could build a simpler data structure. Given the set of points in sorted order, build the dynamic convex hull structure of Overmars and van Leeuwen [16]. The data structure needs $O(n)$ space, $O(n \lg n)$ preprocessing and the query takes $O(\lg^2 n)$. To answer an emptiness query, the leaves in the dynamic convex hull tree corresponding to two extreme points inside the wedge are located. Then by climbing up the tree from the two leaves until the least common ancestor is reached, up to $O(\lg n)$ convex hull structures are collected. Together these convex hull cover all the points between and including the two extreme points. The query is answered by querying each of these $O(\lg n)$ structures for halfplane emptiness. For restricted triangular reporting, a convex peeling layers structure is stored at each node. The data structure needs $O(n \lg n)$ space , $O(n \lg^2 n)$ preprocessing and support queries in $O(\lg^2 n + k)$ time. Daescu *et al.* [11] used a similar idea to build a data structure for halfplane farthest-point queries.

3 Data Structures with $O(2^{1/\epsilon} \lg n)$ Query

In this section we present an near-linear-space data structure for restricted triangular emptiness/reporting queries that achieves logarithmic query time. We start with the naïve data structures from previous section that use $O(n^3)$ space and achieves $O(\lg n)$ query time. Then we recursively apply the space-reducing transformation from Aronov *et al.* [2] to provide a $O(n^{1+\epsilon})$ space data structure. The space-reducing transformations preserve the $O(\lg n)$ query time, but the constant is exponential in $1/\epsilon$.

Here is how the transformation works. Given the set of n points sorted around the origin, select every mth point to be a breakpoint. For each breakpoint m_i, compute the convex hull (convex layers for reporting) for each sequence of points starting at m_i whose length is a power of two. This constitutes linear space for each of the breakpoints. In addition, we compute this data structure recursively for each half-open interval $[m_i, m_{i+1})$ formed by the breakpoints m_i and m_{i+1}. Let $M(n)$ be the size of the data structure on n points.

The recurrence relation (same as in [2]) for the space of the data structure after one space-reducing transformation:

$$M(n) = (n/m + 1)M(m) + O(n^2/m)$$

Applying $(1/2 + 1/\epsilon)$ transformations for any given $\epsilon > 0$, as in [2], yields the desired space and preprocessing of $O(n^{1+\epsilon})$. Although the published proof for the recurrence relation is incorrect, there is an easy fix (omitted) suggested by Erik Demaine [12]–an author of that paper.

To answer a triangular emptiness (reporting) query, identify in $O(\lg n)$ time the extreme points i and j inside the wedge; let m_i and m_j be the breakpoints inside the wedge, closest to points i and j respectively; open intervals (i, m_i) and (m_j, j) do not contain any breakpoint, thus we must have a recursive data structure for them; two convex hull (convex layers) structures will cover all the points in the interval $[m_i, m_j]$. For a reporting query we may report some points twice. The recurrence relation for query time is given as follows, with a solution of $O(2^{1/\epsilon} \lg n)$ (as in [2]):

$$Q(n) = 2Q(m) + O(\lg n)$$

4 Emptiness Queries in $O(\lg n)$ Time

We apply the fractional cascading technique [7] to the $O(\lg^2 n)$ time emptiness query data structure given in Section 2. The modified data structure supports emptiness queries in $O(\lg n)$ time, but the space becomes $O(n \lg n)$.

The basic idea is to reduce a halfplane emptiness query to the problem of finding the extreme point in a given direction. A query halfplane is empty if and only if the extreme (farthest) point in the direction perpendicular to the query line (defining the halfplane) is not contained in the halfplane. With each extreme point we can associate a half-open interval of slopes. We can store a sorted array of these intervals and for a given slope find the extreme point in $O(\lg n)$ time using binary search. We store one such sorted array (corresponding to extreme points in a node's subtree) at each node in the data structure. Notice to answer a restricted triangular emptiness query we still need to perform $O(\lg n)$ extreme point queries, but we can apply fractional cascading here. So only the first extreme point query takes $O(\lg n)$ time, and the queries after that can be answered in constant time.

Achieving $O(\lg n + k)$ time for reporting queries remains a key open problem, as Chazelle and Liu [8] showed that the fractional cascading technique does not generalize to planar maps.

5 A Probabilistic Data Structure

For a given set of n points, there are $\binom{n}{3}$ triangles with vertices in the point set. For a query triangle chosen randomly out of these $\binom{n}{3}$ triangles, there exist near-linear-space data structures that support triangular emptiness and reporting queries in $O(\lg n)$ time with probability $(1 - 1/n)$. Even in case of failure the data structures do not answer queries incorrectly; instead they identify in $O(\lg n)$ time whether the given query can be answered. Although it might seem that the same result could be achieved using ϵ-nets, the obvious strategies for doing so fail.

The key idea is to use the result by Aronov *et al.* [3] which says that for a subset of these $\binom{n}{3}$ triangles of size at least n^2, there exists a point that is inside a fraction of the triangles in the subset. Thus starting from the set of all $\binom{n}{3}$ triangles, we find the deepest point. Using this point as origin we build restricted triangular query data structures over the given point set. We then remove the triangles containing this deepest point, from the set of triangles. We repeat the method on the remaining subset until there are only $O(n^2)$ triangles left. Since we discard a fraction of triangles in every step, we build $O(\lg n)$ restricted triangular query data structures which together can handle $(n^3 - n^2)$ triangles. For a triangular query we can find in $O(\lg n)$ time if there is some data structure whose origin is contained in the query triangle. If there exists such a data structure, we use it to answer the triangular emptiness/reporting query; otherwise we indicate failure or perform a linear-time exhaustive search that would result in $O(\lg n)$ expected time.

6 Ray Intersection Detection and Reporting

In the data structures for restricted triangular range queries if the points are sorted by their x-coordinate instead of radially sorted, the data structures can support axis-parallel three-sided trapezoidal queries. Using two such trapezoidal queries, we can answer double-wedge emptiness and reporting queries where one of the lines forming the wedge is vertical (see Figure 4).

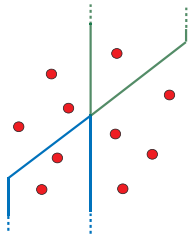


Figure 4: Trapezoidal Range Queries

Since such a double-wedge corresponds to a ray in the dual world, the data structures can be used to answer ray intersection detection and reporting queries among an arrangement of n lines. The data structure for detecting intersection uses $O(n \lg n)$ space and support queries in $O(\lg n)$ time. For intersection reporting the query time is $O(\lg^2 n + k)$ using $O(n \lg n)$ space, or $O(\lg n + k)$ using $O(n^{1+\epsilon})$ space.

7 Non-Orthogonal Square Range Searching

By building a range tree of three-sided trapezoidal query data structures, we could support right-triangular queries, where the base of the triangle is axis-parallel. The space for the data structure increases and the query time slows down by a factor of $O(\lg n)$.

Since a non-orthogonal square can be partitioned into eight axis-parallel right triangles, the data structures for axis-parallel right triangles also support non-orthogonal square (or rectangles with constant aspect ratio–fat rectangles) emptiness and reporting queries .

A brief sketch of the proof: from the highest vertex of the given square draw a vertical line segment down to one of the non-adjacent sides. Similarly from the lowest vertex draw a vertical line segment upwards. Let x be the side-length then each vertical segment has a length in the interval $[x, \sqrt{2}x]$. The two diagonals of the square intersect at a distance $\frac{1}{\sqrt{2}}x$ from each vertex. Therefore, the downward vertical segment goes below and the upward vertical segments goes above this point of intersection. Thus we can draw horizontal segment from the lower (upper) endpoint of the downward (upward) vertical segment to the other vertical segment. Notice the argument does not hold for long skinny non-orthogonal rectangles.

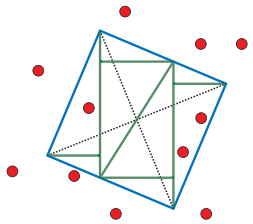


Figure 5: Non-Orthogonal Square Range Queries

8 Concluding Remarks

It is impossible to achieve $O(\text{polylog } n)$ time for triangular counting queries using $O(n \text{ polylog } n)$ space, even when the query triangle contains the origin; since a halfplane counting query can be reduced in constant time to a restricted triangular counting query. Thus the lower bound for halfplane range counting queries [4] applies to restricted triangular range counting as well.

The fact that the data structures for restricted triangular queries support both emptiness and reporting queries in $O(\text{polylog } n)$ time indicates that the techniques for restricted case will not extend to general triangular emptiness. The lower bound on triangular reporting queries [9] on a pointer machine, indicates that any near-linear-space data structure achieving $O(\text{polylog } n)$ time for triangular emptiness must handle emptiness queries differently from reporting queries.

Acknowledgment

This paper has benefited from the suggestions of anonymous referees.

References

[1] P. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J.E. Goodman, and R. Pollack, editors, *Adv. in Disc. and Comp. Geometry*, volume 23, pages 1–56. AMS Press, Providence, RI, USA, 1999.

[2] Boris Aronov, Prosenjit Bose, Erik D. Demaine, Joachim Gudmundsson, John Iacono, Stefan Langerman, and Michiel H. M. Smid. Data structures for halfplane proximity queries and incremental voronoi diagrams. In *LATIN*, pages 80–92, 2006.

[3] Boris Aronov, Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, and Rephael Wenger. Points and triangles in the plane and halving planes in space. *Discrete Comput. Geom.*, 6(5):435–442, 1991.

[4] Hervé Brönnimann, Bernard Chazelle, and János Pach. How hard is half-space range searching? *Discrete & Computational Geometry*, 10:143–155, 1993.

[5] Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *JAMS*, 2:637–666, 1989.

[6] Bernard Chazelle, Leo J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

[7] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[8] Bernard Chazelle and Ding Liu. Lower bounds for intersection searching and fractional cascading in higher dimension. *J. Comput. Syst. Sci.*, 68(2):269–284, 2004.

[9] Bernard Chazelle and Burton Rosenberg. Simplex range reporting on a pointer machine. *Comput. Geom. Theory Appl.*, 5(5):237–247, 1996.

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

[11] Ovidiu Daescu, Ningfang Mi, Chan-Su Shin, and Alexander Wolff. Farthest-point queries with geometric and combinatorial constraints. *Comput. Geom. Theory Appl.*, 33(3):174–185, 2006.

[12] Erik D. Demaine. Personal communication, 2007.

[13] Jeff Erickson. Space-time tradeoffs for emptiness queries. *SIAM J. Comput.*, 29(6):1968–1996, 2000.

[14] Partha P. Goswami, Sandip Das, and Subhas C. Nandy. Simplex range searching and k nearest neighbors of a line segment in 2d. In *SWAT*, pages 69–79, London, UK, 2002. Springer-Verlag.

[15] Jirí Matousek. Geometric range searching. *ACM Comput. Surv.*, 26(4):421–461, 1994.

[16] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

A generalization of Apollonian packing of circles

Gerhard Guettler* Colin Mallows†

Abstract

Three circles touching one another at distinct points form two curvilinear triangles. Into one of these we can pack three new circles, touching each other, with each new circle touching two of the original circles. In such a sextuple of circles there are three pairs of circles, with each of the circles in a pair touching all four circles in the other two pairs. Repeating the construction in each curvilinear triangle that is formed results in a generalized Apollonian packing. We can invert the whole packing in every circle in it, getting a "generalized Apollonian super-packing". Many of the properties of the Descartes configuration and the standard Apollonian packing carry over to this case. In particular, there is an equation of degree 2 connecting the bends (curvatures) of a sextuple; all the bends can be integers; and if they are, the packing can be placed in the plane so that for each circle with bend b and center (x, y), the quantities $bx/\sqrt{2}$ and by are integers.

Recently there has been renewed interest in a very old idea, that of Apollonian packing of circles, in which an initial configuration of three mutually tangent circles is augmented by repeatedly drawing new circles in each curvilinear gap. See for example Mumford et al [8]. We can also study "super-Apollonian" packings which are obtained by repeatedly inverting an Apollonian packing in every circle in it. It is a remarkable fact that Apollonian and super-Apollonian packings exist in which all the bends (curvatures) are integers. This property was studied in detail by Graham et al [3], and the group theory associated with these packings has been studied by the same authors [4-6]. Also, if all the bends are integers, the super-Apollonian packing can be placed in the plane so that all the "bend times center" quantities are integers. Several extensions of the Apollonian idea have been studied, for example Mauldon [7] studied configurations in which adjacent circles do not touch but have constant "separation".

Our own interest lies in extending these ideas in new directions, particularly by packing not one but three circles within each triangular gap, thus forming sextuples of circles, and in exploring the degree to which the theory associated with the classical packings can be extended to cover this case. We find that all the bends in such a generalized packing can be integers; and there are results relating to the positions of the centers of the circles that directly generalize those found by Lagarias et al [2] in the classical Descartes-Apollonian case.

Figure 1 shows the four possible configurations of a sextuple. There can be zero, one, or two circles wth bend zero (i.e. straight lines), and at most one bend can be negative, as in case (a).

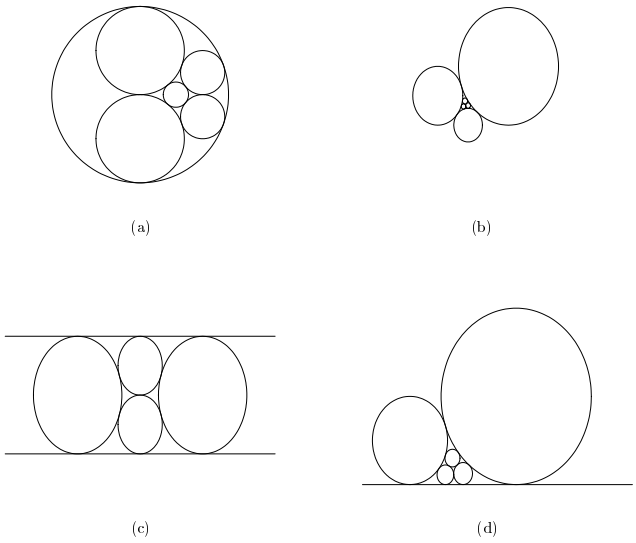


Figure 1: Sextuple configurations

These configurations generalize the classical Descartes configuration, in which just one circle is placed in a curvilinear triangle. Such a sextuple of circles forms an $n = 4$ example of what we call a "ball-bearing" configuration, in which a ring of n circles (each touching two others) have the property that there are "inner" and "outer" circles that each touch all n circles in the ring. The n=3 case reproduces the classical Descartes configuration. With $n = 4$ the circles come in three pairs, with each of the circles in a pair touching all four circles in the other two pairs. The circles of a pair do not touch one another. The sextuple thus has the symmetry of the vertices of an octahedron (or of the faces of a cube), rather than the symmetry of a tetrahedron as in the Descartes case.

*University of Applied Sciences Giessen Friedberg (Germany), dr.gerhard.guettler@swd-servotech.de

†Avaya Labs, Basking Ridge NJ USA 07920, colinm@research.avayalabs.com

Repeating the construction in each curvilinear triangle that is formed results in a "generalized Apollonian packing". See Figures 2 (based on Figure 1(a)) and 3 (based on Figure 1(c)). Here, and in subsequent figures, we include only circles with bend less than 100.

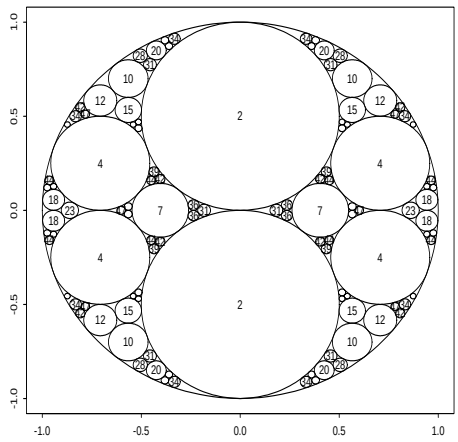


Figure 2: A generalized Apollonian packing

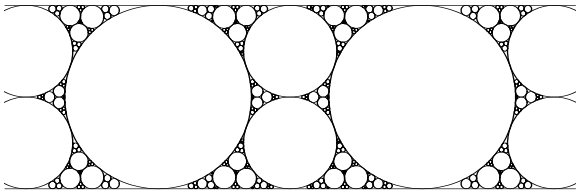


Figure 3: Another generalized Apollonian packing

Many of the properties of the Descartes configuration and the standard Apollonian packing carry over to this case. In particular:

(i) Given a ring of four circles, formed by two pairs of circles in a sextuple, there is a quadratic equation whose coefficients involve the bends of these four circles, the roots of which are the bends of the other pair of circles in the set. Explicitly,

$$2x^2 - x\sigma + \tau - \frac{3}{8}\sigma^2 = 0 \qquad (1)$$

where $\sigma = b_1 + b_1' + b_2 + b_2', \tau = {b_1}^2 + {b_1'}^2 + {b_2}^2 + {b_2'}^2$ This generalizes the classical Descartes equation. Replacing each bend by the corresponding bend*(complex) center gives another result which generalizes the "Complex Descartes Theorem" of [2].

(ii) There is an analog of "Descartes reflection" (see [2]) in which three circles (one from each pair in a sextuple, these three circles occupying a curvilinear triangle formed by the other three circles of the sextuple) are replaced by three circles occupying the other triangle formed by these three circles, thus forming another sextuple. Given a sextuple, this operation can be performed in eight different ways. Iteration of this operation creates a generalized Apollonian packing in which the interiors of all circles are disjoint. A packing is determined by any three touching circles within it.

(iii) all six bends of the circles in a sextuple can be integers. Examples: in Figure 1(c), the bends are (0,2; 0,2; 1,1); in Figure 1(a) they are (-1,7; 2,4; 2,4). This property is inherited by all derived circles.

(iv) if all bends of a sextuple are integers, the sextuple can be placed in the plane so that for each circle with bend b and center (x, y), the "bend times center" quantities (bx, by) have both $bx/\sqrt{2}$ and by integers. This property is inherited by the generalized packing based on this sextuple.

(v) The construction of the generalized packing can be realised by integral linear operations acting on matrices representing the sextuples. These matrices could be 6 x 4 matrices with each row containing the "abbc" or "augmented bend, bend times center" coordinates introduced in [2]. The abbc coordinates of a circle C with bend ($= 1/\text{radius}$) b and center (x, y) is the vector $\mathbf{a}(C) = (\bar{b}, b, bx, by)$ where $\bar{b}$ is the bend of the circle that is the inverse of C in the unit circle, namely

$$\bar{b} = b(x^2 + y^2) - 1/b \qquad (2)$$

However it is convenient to represent a sextuple by a 4 x 4 matrix that we call $\mathbf{F}(C)$ in which the first three rows contain the abbc coordinates of three of the circles in the sextuple (one from each pair) and the fourth row is the average of two rows that represent a pair of circles in the sextuple (this average is the same for each of the three pairs). This row does not represent a circle.

(vi) There are dual operations acting on the right, which represent Mobius transformations.

(vii) Among the sextuples with integer bends, there are "root" sextuples (see [3] and [5]) having the property that any application of the reflection operation in (ii) results in circles with larger bends. These root sextuples can be found by applying a reduction algorithm, just as in the Descartes case. Except for the special sextuple with bends (0,2; 0,2; 1,1) (Figure 1(a)), each root sextuple has exactly one circle with negative bend. We have a conjecture as to the number of root sextuples with smallest bend $-n$.

(viii) By inverting a generalized Apollonian packing in each circle in the packing, and then again in every circle, and so on, we obtain a "generalized Apollonian super-packing", directly analogous to the Apollonian super-packing studied in [5]. There is essentially just one super-packing in which all bends are integral. This super-packing can be placed in the plane, in exactly four ways, so that each $bx/\sqrt{2}, by$ is integral. In each version

of this super-packing, there is a basic rectangle $(0, \sqrt{2})$ x $(0, 1)$ which repeats by translation and reflection to cover the whole plane. See Figure 4.

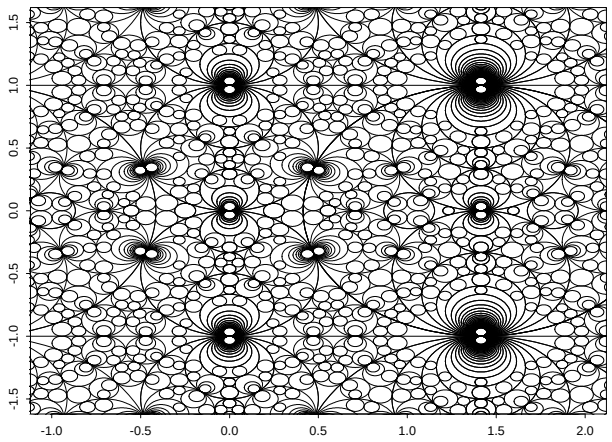


Figure 4: A generalized Apollonian super-packing

(ix) Each primitive integral sextuple appears exactly once in the basic $(0, \sqrt{2})$ x $(0, 1)$ rectangle of the super-packing. Computation suggests that there are some symmetries within the basic rectangle, like those shown in [5].

(x) One can consider "ball-bearing" structures of circles, in which a ring of n balls (each touching two neighbors) have the property that there exist "inner" and "outer" circles that each touch each of the "balls" in the ring. The case $n = 3$ reproduces the Descartes configuration; the case $n = 4$ gives the sextuples studied in this paper. The bends of all $n+2$ circles can be integral only when $n = 3, 4, 6$. There is a quadratic equation whose roots are the bends of the "inner" and "outer" circles, and whose coefficients involve the bends of the circles in the ring.

(xi) There is an analog of the Farey series and the associated Ford circles, in which at every stage we insert two new fractions (and two new touching circles) instead of just one, between every existing adjacent pair of fractions. See Figure 5.

There are several open questions.

Is the conjectured formula for the number of root sextuples correct?

Are the conjectured symmetries within the basic cell of the super-packing valid?

Do all integers arise as bends of circles in generalized Apollonian packings?

Is the Hausdorff dimension of our generalized super-packing the same as in the Apollonian case?

Are there other ways to generalize the classical Apollonian packing?

Are there other ways to pack integer-bend circles?

What about higher dimensions?

References

[1] Aharonov, D. and Stephenson, K. Geometric sequences of discs in the Apollonian packing. Algebra i Analiz. 3:104-140, 1997

[2] Lagarias, J.C., Mallows, C.L., and Wilks, A.R Beyond the Descartes circle theorem. *Amer. Math. Monthly* 109:338-361, 2002.

[3] Graham, R.L., Lagarias, J.C., Mallows, C.L., Wilks, A.R., and Yan, C. Apollonian circle packings: number theory. *J. Number Theory* 100:1-45, 2003.

[4] Graham, R.L., Lagarias, J.C., Mallows, C.L., Wilks, A.R., and Yan, C. Apollonian circle packings: Geometry and Group Theory I. The Apollonian Group . *Discrete and Computational Geometry* 34:547-585, 2005.

[5] Graham, R.L., Lagarias, J.C., Mallows, C.L., Wilks, A.R., and Yan, C. Apollonian circle packings: Geometry and Group Theory II. Super-Apollonian Group and Integral Packings. *Discrete and Computational Geometry* 35:1-36, 2006.

[6] Graham, R.L., Lagarias, J.C., Mallows, C.L., Wilks, A.R., and Yan, C. Apollonian circle packings: Geometry and Group Theory III. Higher Dimensions. *Discrete and Computational Geometry* 35:37-72, 2006.

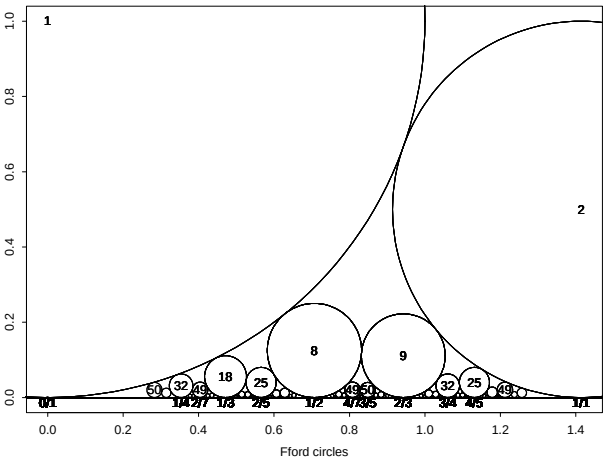


Figure 5: Generalized Ford circles

[7] Mauldon, J. G. Sets of equally inclined spheres. Canadian J. Math. 14:509-516, 1962.

[8] Mumford, D, Series, C, and Wright, D. Indra's Pearls Cambridge U.P. 2002

A note on α-drawable k-trees

David Bremner* Jonathan Lenchner† Giuseppe Liotta‡ Christophe Paul§ Marc Pouget¶
Svetlana Stolpner‖ Stephen Wismath**

Abstract

We study the problem of realizing a given graph as an α-complex of a set of points in the plane. The graphs we consider are trees and 2-trees. In the case of 2-trees, we confine our attention to the realizability of graphs as the α-complex minus faces of dimension two; in other words, realizability of the graph in terms of the 1-skeleton of the α-complex of the point set. We obtain both positive (realizability) and negative (non-realizability) results.

1 Preliminaries

Consider a finite set S of points in the plane and a non-negative real number α. For each $p \in S$, let $B_p(\alpha) = \{x : \|x - p\| < \alpha\}$ be a disk centered at p with radius α. Let $B(S, \alpha)$ denote the union of balls $B_p(\alpha)$. We can decompose this union by intersecting each ball $B_p(\alpha)$ with the Voronoi cell V_p of p into convex pieces $BV_p(\alpha) = B_p(\alpha) \cap V_p$. Define the *$\alpha$-complex* as the nerve of the decomposition of $B(S, \alpha)$ by $BV_p(\alpha)$ or the set of all simplices $\sigma \subseteq S$ such that $\bigcap_{p\in\sigma} BV_p(\alpha) \neq \emptyset$. For details, see [6, 1]. We use $\alpha(S)$ to refer to the α-complex for the set of points S for this fixed value of the radius α. The 1-skeleton $\alpha_1(S)$ of $\alpha(S)$ is the collection of 1-dimensional faces in $\alpha(S)$.

We shall find the following graph useful: the Gabriel graph of S, $GG(S)$, contains an edge between any pair of points p and q whenever the disk having the line segment pq as its diameter is empty. The edges of the $GG(S)$ are those Delaunay edges that intersect their dual Voronoi edges.

A k-tree is a graph obtained from a k-clique by 0 or more iterations of adding a new vertex joined to exactly k vertices of a k-clique in the old graph. A partial k-tree is a subgraph of a k-tree. Trees are 1-trees.

*Faculty of Computer Science, University of New Brunswick bremner@unb.ca
†IBM T.J. Watson Research Center, lenchner@us.ibm.com
‡School of Computing, University of Perugia, liotta@diei.unipg.it
§LIRMS CNRS-Université de Montpellier II, paul@lirmm.fr
¶INRIA Lorraine - Loria, marc.pouget@loria.fr
‖School of Computer Science, McGill University, sveta@cim.mcgill.ca
**Department of Mathematics and Computer Science, University of Lethbridge, wismath@cs.uleth.ca

2 Introduction

The problem of characterizing those geometric graphs that satisfy some proximity rule has a long tradition in the computational geometry literature. This tradition is justified in part by the theoretical interest of the associated questions in their own right, and in part by the variety of application areas where proximity graphs are used as descriptors of the shape of a set of points. Extensive surveys about different proximity rules with their applications can be found in [10, 16].

The characterization problem for proximity graphs can naturally be expressed as a graph drawing question. Indeed, for a proximity rule $\mathcal{P}$ and a family of graphs $\mathcal{G}$ we can say that a member $G \in \mathcal{G}$ is *$\mathcal{P}$-drawable* if there exists a set S of distinct points in the plane such that the geometric graph constructed on S by using rule $\mathcal{P}$ is isomorphic to G; we call this geometric graph a *$\mathcal{P}$-drawing* of G. Characterizing $\mathcal{P}$-drawable graphs corresponds to describing the combinatorial properties of the associated $\mathcal{P}$-drawings. Different families of $\mathcal{P}$-drawable graphs have been studied in the literature, including Gabriel drawable graphs, Delaunay drawable triangulations, and sphere of influence drawable graphs (see, *e.g.*, [2, 4, 9, 15]). Those trees that can be drawn as the minimum spanning tree of a set of points in the plane are studied in [5, 11, 14]. The interested reader is referred to [12] for a survey of proximity drawability problems and more references on the topic.

This paper initiates the study of the combinatorial properties of α-complexes of set of points in the plane. α-complexes are a fundamental object in computational topology [8] and have applications in such areas as structural molecular biology [7] and shape analysis [3].

We say that a graph G with n vertices is *α-drawable* if there exists a set S of n distinct points in the plane such that the α-complex of S is a straight-line drawing of G for some value of the parameter α. We call such an α-complex of S an *α-drawing of G*. We present some negative and positive results about those trees and partial 2-trees that are α-drawable. A detailed description of the results in this note follows.

- Regarding trees, we show differences between α-drawable trees and other well-studied families of proximity drawable trees. Namely, we show that that the family of α-drawable trees is a subset of the

relative neighborhood drawable trees and a subset of those trees that are drawable as the minimum spanning tree of a set of points. We also prove that there exist α-drawable trees that are not Gabriel-drawable.

- We exhibit a simple partial 2-tree that is not α-drawable. Motivated by the observation that the above counterexample for 2-trees is a series-parallel graph whose planar embeddings all have some interior vertex, we show that all biconnected outerplanar graphs are α-drawable.

Our characterizations of α-drawable graphs are based on constructive proofs that give rise to linear time drawing algorithms, assuming the real RAM model of computation.

3 Results on Trees

Lemma 1 *Let (u, v) be an edge of $GG(S)$. If $d(u, v) \geq 2\alpha$, then $(u, v) \notin \alpha(S)$. If $d(u, v) < 2\alpha$, then $(u, v) \in \alpha(S)$.*

Theorem 2 *If $\alpha(S)$ is a tree, it is the Euclidean minimum spanning tree of S.*

Proof. Suppose $\alpha(S)$ were a tree but not the minimum spanning tree, $MST(S)$. Then there would be an edge (u, v) in $\alpha(S)$ not in $MST(S)$, such that $d(u, v) < 2\alpha$. Vertices u and v are connected by a path in $MST(S)$ and adding edge (u, v) to $MST(S)$ creates a cycle. We know that all edges in $MST(S)$ are Gabriel edges (since $MST \subseteq GG$). Suppose that the cycle made by adding (u, v) to $MST(S)$ does not contain an edge of length $\geq 2\alpha$. Then all edges in the path from u to v in $MST(S)$ are Gabriel edges of length $< 2\alpha$. By Lemma 1, these edges are in $\alpha(S)$. But so is (u, v). Thus, $\alpha(S)$ contains a cycle. This is impossible as it is a tree. Therefore, an edge of $MST(S)$ along the path from u to v is of length $\geq 2\alpha$. It may be exchanged with (u, v) to obtain a lighter $MST(S)$. If $\alpha(S)$ is a tree, there cannot be an edge in $\alpha(S)$ that is not in $MST(S)$. □

Corollary 3 *If $\alpha(S)$ is a tree, it contains exactly those Gabriel edges whose length is $< 2\alpha$.*

3.1 Non-realizability results

Lemma 4 *Let $\alpha(S)$ be a tree. For two edges (u, v), (w, v) sharing a common vertex v, $\angle uvw > \pi/3$. Moreover, $\angle uvw$ is the largest angle in $\triangle uvw$.*

Proof. Suppose that $\angle uvw \leq \pi/3$. Then (u, w) is not the longest edge of $\triangle uvw$, *i.e.* $d(u, w) \leq d(u, v) < 2\alpha$, $d(u, w) \leq d(w, v) \leq 2\alpha$. Since $(u, w) \notin T$ and $d(u, w) < 2\alpha$, by Corollary 3, $(u, w) \notin GG(S)$. Let p be a point inside the diametric disk of u and w. Then $d(u, p) < d(u, v)$ and $d(w, p) < d(w, v)$. One of (u, p), (w, p) is not in T as it is a tree. Either (u, v) may be replaced with the shorter edge (u, p) or (w, v) with (w, p) to make a lighter spanning tree of S than T. This contradicts Theorem 2. We have shown that it is not possible that $\angle uvw \leq \pi/3$.

It is not possible that $\triangle uvw$ has a larger angle than $\angle uvw$ since that would mean that the longest side of $\triangle uvw$ is not (u, w) and that T is not a $MST(S)$. □

Corollary 5 *The maximum vertex degree of an α-drawable tree is at most 5 for any possible value of α.*

This is in contrast with a generic Euclidean minimum spanning tree, which has vertex degree of at most 6 [14].

Lemma 6 *A tree T consisting of two adjacent vertices of degree 5 and additionally only leaf nodes is not α-drawable for any possible value of the parameter α.*

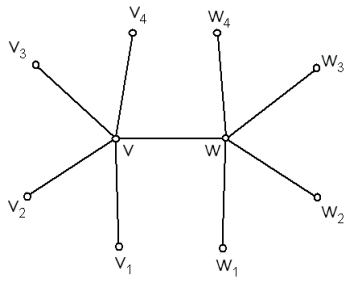


Figure 1: Two adjacent degree 5 vertices v and w.

Sketch of Proof. Suppose $\alpha(S)$ is a realization of T as an α-complex for a given α. Let v and w be adjacent degree 5 vertices in $\alpha(S)$. Let v_i be the leaves adjacent to v and w_i be the leaves adjacent to w. See Fig. 1. Either $\angle v_2vw \geq \angle vww_3$ or $\angle v_2vw < \angle vww_3$. If it is the latter, we rotate the drawing so that w_3 takes the place of v_2 and v_2 the place of w_3. Hereafter, we assume that $\angle v_2vw \geq \angle vwv_3$.

By Lemma 4, $\angle v_2vv_3 > \pi/3$, $\angle v_3vv_4 > \pi/3$ and $\angle w_4ww_3 > \pi/3$. Combining this with the fact that $\angle v_2vw \geq \angle vww_3$, we get $\angle v_4vw + \angle vww_4 < \pi/2$. Assume that $\angle v_4vw < \pi/2$ (an analogous argument can be made in case $\angle vww_4 < \pi/2$). We know that $\alpha(S)$ must not contain (v_4, w_4). By Corollary 3, if this edge is not present, it is either because (v_4, w_4) is not a Gabriel edge or because $d(v_4, w_4) \geq 2\alpha$.

First, we show that (v_4, w_4) must be a Gabriel edge. Suppose the contrary. Then $\angle v_4ww_4 \geq \pi/2$. Write $\angle v_4vw$ as $\pi/2 - \beta$ and $\angle w_4wv$ as $\pi/2 + \gamma$, where $\gamma < \beta$. Then $\angle vv_4w \geq \pi - \pi/2 + \beta - \gamma > \pi/2$. This implies that the Gabriel disk of (v, w) is not empty and that (v, w) is not in $\alpha(S)$, a contradiction.

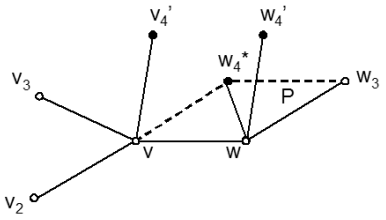


Figure 2: v_4 lies on (v, v_4'), w_4 lies on (w, w_4').

Last, we show that necessarily $d(v_4, w_4) < 2\alpha$. Let L be the length of the longest edge in $\alpha(S)$. Let v_4' be a point such that $\angle v_4' vw = \angle v_4 vw < \pi/2$ and $d(v_4', v) = L$. We then choose point w_4', satisfying $\angle v_4' vw + \angle vww_4' < \pi$ and $d(w, w_4') = L$, such that w_4 lies along (w, w_4'), see Fig. 2. We try to choose v_4 and w_4 as far apart as possible.

One possibility is to place w_4 so that $d(v, w_4) = d(w, w_3)$ and $d(v, w) = d(w_4, w_3)$. In this case w_4 is a vertex of a parallelogram P, w_4^*, see Fig. 2. Since $\angle v_2 vw \geq \angle vww_3, \angle v_2 vv_3 > \pi/3$ and $\angle v_3 vv_4 > \pi/3$, it follows that $\angle v_4 vw_4^* < \pi/3$. Since $d(v, v_4) \leq L$ and $d(v, w_4^*) \leq L$, this implies that $d(v_4, w_4^*) < L$. Thus, this placement for w_4 is not sufficiently far from v_4. Note that neither $\angle w_4^* ww_3 > \pi/2$ nor $\angle vww_4^* > \pi/2$ for that would mean that one of (v, w) or (w, w_3) is not a Gabriel edge and may not be in $\alpha(S)$.

Consider placing w_4 inside P. Note that $\angle vww_3 \leq \angle v_2 vw < \pi$, since $\angle v_2 vv_3 + \angle v_3 vv_4 + \angle v_4 vw > \pi$. Then we have the following 3 cases: (1) Suppose $\angle vww_4 < \pi/2$ and $\angle w_4 ww_3 < \pi/2$. Then (v, w_4) and (w_4, w_3) are Gabriel edges and since they are not in $\alpha(S)$, $d(v, w_4) > L$, $d(w_4, w_3) > L$ by Corollary 3. No such placement of w_4 is possible inside P; (2) Suppose $\angle vww_4 \geq \pi/2$ and $\angle w_4 ww_3 < \pi/2$. Then (w_4, w_3) is a Gabriel edge and since it is not in $\alpha(S)$, $d(w_4, w_3) > L$, while (v, w_4) is not a Gabriel edge. As $\angle vww_4 > \pi/2$, this angle is greater than $\angle vww_4^*$. Increasing $\angle vww_4$ decreases $\angle w_4 ww_3$. Thus, w_4 lies inside $\triangle vw_4^* w$ and $d(v, w_4) \leq L$. As (v, w_4) is a Gabriel edge and $d(v, w_4) \leq L$, $(v, w_4) \in \alpha(S)$, a contradiction; (3) Suppose $\angle vww_4 < \pi/2$ and $\angle w_4 ww_3 \geq \pi/2$. Then (v, w_4) is a Gabriel edge and since it is not in $\alpha(S)$, $d(v, w_4) > L$, while (w_4, w_3) is not a Gabriel edge. This is impossible, by the same argument as in (2). Thus, w_4 may not be placed in P.

Consider moving w_4 outside of P so that $d(w_4, w_3) = d(w_4^*, w_3)$. Recall that (w_4^*, w_3) is a Gabriel edge in P. As $d(w, w_4)$ increases, $\angle w_4 ww_3$ decreases, so (w_4, w_3) remains a Gabriel edge. Thus w_4 may not be placed closer to w_3. As $d(w, w_4)$ increases, $d(v_4, w_4)$ first decreases and then increases. It can be shown (using similar arguments as those presented already) that when $d(w, w_4) = L$, $d(v_4, w_4) \leq L$. Thus, no choice of w_4 such that $d(w_4, w_3) = d(w_4^*, w_3)$ is sufficiently far from v_4, and therefore, the same is true if $d(w_4, w_3) > d(w_4^*, w_3)$.

Therefore, (v_4, w_4) is a Gabriel edge such that $d(v_4, w_4) \leq L < 2\alpha$ and must be in $\alpha(S)$. □

3.2 Realizability results

Lemma 7 *$\alpha(S)$ can be a tree with arbitrarily many adjacent degree four vertices.*

Sketch of Proof. See Fig. 3. □

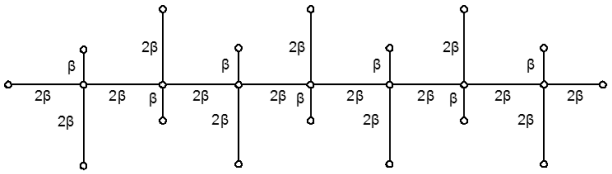


Figure 3: An α-drawing of a "caterpillar graph" whose non-leaf vertices have degree 4. In the figure, $\frac{2}{\sqrt{5}}\alpha \leq \beta < \alpha$.

We conclude this section by comparing α-drawable trees with other well-known families of proximity drawable trees.

Theorem 8 *The family of α-drawable trees is a proper subset of the family of trees that have a minimum spanning tree realization. It is also a proper subset of the relative neighborhood drawable trees. Also, there exist α-drawable trees that are not Gabriel drawable.*

Sketch of Proof. All trees whose maximum vertex degree is at most five are relative neighborhood drawable and also admit a realization as the Euclidean minimum spanning tree of a set of points in the plane [2, 14]. On the other hand, the tree in Fig. 1 containing two adjacent degree five vertices is not α-drawable by Lemma 6. Also, no tree having two adjacent vertices of degree four is Gabriel drawable [2], while, by Lemma 7 it may be α-drawable. □

4 Results on 2-Trees

4.1 Non-realizability results

Lemma 9 *There are 2-trees that are not α-drawable for any possible value of α.*

Sketch of Proof. Consider the partial 2-tree ABU given in Fig. 4.

It can be shown that ABU is not α-drawable for any possible value of α (proof omitted). The only completion of ABU in a 2-tree with 5 vertices is ABU augmented by the edge (a, c). This 2-tree is well-known as the *3-sun*. Therefore, a 3-sun cannot be realized as $\alpha_1(S)$ for any point set S. □

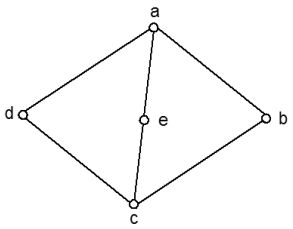


Figure 4: The partial 2-tree ABU.

4.2 Realizability results

Theorem 10 *Every biconnected outerplanar graph is α-drawable for any possible value of α.*

Sketch of Proof. Let G be a biconnected outerplanar graph. Construct a special dual for G by adding a vertex for each bounded face of the graph and a vertex for each edge on the unbounded face. Connect with edges those vertices corresponding to the bounded faces that share an edge and also those vertices corresponding to the bounded faces that have an edge on the unbounded face with the vertex corresponding to that edge. This dual is a tree with no degree 2 vertices. By the results of [13], this tree may be realized as a Voronoi diagram of a set of points S. If we set the scale of this drawing to be sufficiently small, for the given α, the α-balls touch the Voronoi edges for any pair of primal edges of G. □

5 Open Problems

1. Are all binary trees α-drawable? Are all binary trees up to some maximum depth k α-drawable?
2. Is a Gabriel drawable tree always α-drawable?
3. Which partial 2-trees are α-drawable?
4. If $\alpha(S)$ is a tree and we consider any subtree of $\alpha(S)$, is it true that the subtree is $\alpha(S')$ on the restricted set of vertices S'? If true, this would immediately settle the following problem, generalizing Lemma 6:
5. Is any tree containing two adjacent degree 5 vertices α-drawable?

6 Acknowledgements

We gratefully acknowledge conversations with Raimund Seidel, Muhammad Abubakr (who suggested considering the ABU graph) and other participants of the 2008 McGill-INRIA Workshop on Computational Topology held at the Bellairs Research Institute in Holetown, Barbados. We are especially thankful for the tireless work of the organizers of this workshop, Sue Whitesides, Hazel Everett and Sylvain Lazard.

References

[1] N. Akkiraju, H. Edelsbrunner, M. Facello, P. Fu, E. P. Mucke, and C. Varela. Alpha shapes: definition and software. *Internat. Comput. Geom. Software Workshop*, 1995.

[2] P. Bose, W. Lenhart, and G. Liotta. Characterizing proximity trees. *Algorithmica*, 16:83–110, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).

[3] H.-L. Cheng, H. Edelsbrunner, and P. Fu. Shape space from deformation. *Pacific Graphics*, pages 104–113, 1998.

[4] M. B. Dillencourt. Realizability of Delaunay triangulations. *Inform. Process. Lett.*, 33(6):283–287, Feb. 1990.

[5] P. Eades and S. Whitesides. The realization problem for Euclidean minimum spanning trees is NP-hard. *Algorithmica*, 16:60–82, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).

[6] H. Edelsbrunner. The union of balls and its dual shape. *Discrete Computational Geometry*, 13:415–440, 1995.

[7] H. Edelsbrunner. Biological applications of computational topology. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.

[8] H. Edelsbrunner and J. Harer. Persistent homology — a survey. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Twenty Years After*. AMS Press, to appear.

[9] M. Jacobson, M. Lipman, and F. Morris. Trees that are sphere of influence graphs. *Appl. Math. Letters*, 8(6):89–93, 1995.

[10] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 80(9):1502–1517, Sept. 1992.

[11] M. Kaufmann. Polynomial area bounds for mst embeddings of trees. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing*, volume 4875 of *LNCS*, pages 88–100. Springer, 2007.

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

[13] G. Liotta and H. Meijer. Voronoi drawings of trees. *Computational Geometry: Theory and Applications*, 24(3):147–178, 2003.

[14] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discrete and Computational Geometry*, 8:256–293, 1992.

[15] M. Soss. On the size of the Euclidean sphere of influence graph. In *Canadian Conference on Comput. Geom.*, 1999.

[16] G. Toussaint. Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining. *Computational Geometry and Applications*, 15(2):101–150, 2005.

VC-Dimension of Visibility on Terrains

James King*

Abstract

A guarding problem can naturally be modeled as a set system $(\mathcal{U}, \mathcal{S})$ in which the universe $\mathcal{U}$ of elements is the set of points we need to guard and our collection $\mathcal{S}$ of sets contains, for each potential guard g, the set of points from $\mathcal{U}$ seen by g.

We prove bounds on the maximum VC-dimension of set systems associated with guarding both 1.5D terrains (monotone chains) and 2.5D terrains (polygonal terrains). We prove that for monotone chains, the maximum VC-dimension is 4 and that for polygonal terrains, the maximum VC-dimension is unbounded.

1 Introduction

Terrain Guarding A 1.5D (resp. 2.5D) terrain is a continuous piecewise linear univariate (resp. bivariate) function. In other words, a 1.5D terrain is a simple polygonal chain that intersects any vertical line at at most one point and a 2.5D terrain is a polygonal mesh with no holes that intersects any vertical line at at most one point.

On a terrain T, either 1.5- or 2.5-dimensional, we say that two points see each other if the line segment between them does not pass under T. To guard T optimally we must find a minimum set $G \subset T$ of points on the terrain such that every point on T is seen by a point in G.

Guarding 1.5D terrains is not known to be NP-hard but no polynomial-time exact algorithm has been found. The best polynomial-time algorithm found so far is a 5-approximation algorithm[1] [10]. Guarding 2.5D terrains is NP-complete, as proved by Cole and Sharir [4].

Set Cover and VC-Dimension Set Cover is a well-studied NP-complete optimization problem. Given a universe $\mathcal{U}$ of elements and a collection $\mathcal{S}$ of subsets of $\mathcal{U}$, Set Cover asks for the minimum subset $\mathcal{C}$ of $\mathcal{S}$ such that $\bigcup_{S \in \mathcal{C}} S = \mathcal{U}$. In other words, we want to cover all of the elements with the minimum number of sets in $\mathcal{S}$.

In general, Set Cover is not only difficult to solve exactly (see, *e.g.*, [7]) but is also difficult to approximate – no polynomial time approximation algorithm can have an $o(\log n)$ approximation factor unless NP $\subseteq$ DTIME($n^{\log \log n}$) [6].

However, some instances of Set Cover (we refer to instances as *set systems*), are more complex than others. VC-dimension is a measure of the complexity of a set system $(\mathcal{U}, \mathcal{S})$. Consider a set $S \subseteq \mathcal{U}$. There are $2^{|S|}$ possible subsets of S. We say that S is *shattered* by a collection $\mathcal{C}$ of $2^{|S|}$ sets if, for each of the $2^{|S|}$ subsets of S, there is a set in $\mathcal{C}$ that contains those elements of S but no other elements of S. The VC-dimension of a set system $(\mathcal{U}, \mathcal{S})$ is the maximum d for which a set of d elements from $\mathcal{U}$ can be shattered by sets $\mathcal{C} \subseteq \mathcal{S}$.

VC-Dimension and Approximate Set Cover Set Cover is hard to approximate in general, but set systems with low VC-dimension are simpler and, intuitively, should be easier to approximate. Brönnimann and Goodrich [3] provide a polynomial time $O(d \log(d \cdot \mathrm{OPT}))$-approximation algorithm for instances of Set Cover with VC-dimension d, where OPT is the size of the optimum solution. When d is bounded from above by a constant, this gives an $O(\log \mathrm{OPT})$ approximation factor.

Set Systems of Guarding Problems Guarding problems can naturally be expressed as instances of Set Cover. For an instance of a guarding problem, the associated set system $(\mathcal{U}, \mathcal{S})$ is constructed with $\mathcal{U}$ containing the points that need to be guarded and $\mathcal{S}$ containing, for each potential guard g, the set of points that g can see. For the sake of brevity we sometimes refer to the VC-dimension of a guarding problem; by this we mean the maximum possible VC-dimension of a set system associated with an instance of the problem.

The classic *art gallery problem*, the problem of guarding the interior of a polygon, is perhaps the best-known and best-studied guarding problem. If the polygon can have holes, the problem is as hard to approximate as general instances of Set Cover [5]. However, if the polygon to be guarded is simple (*i.e.* contains no holes), the associated set system has constant VC-dimension [9] (it is known to be at least 6 and at most 23 [11]). This means that the technique of Brönnimann and Goodrich leads to an $O(\log \mathrm{OPT})$-approximation algorithm, which is the best known. Guarding simple art galleries is known to be APX-hard [5] but the exact ap-

*Department of Computer Science, McGill University, jking@cs.mcgill.ca

[1] An error in the paper was found after publication, and the only fix found so far increases the approximation factor from 4 to 5.

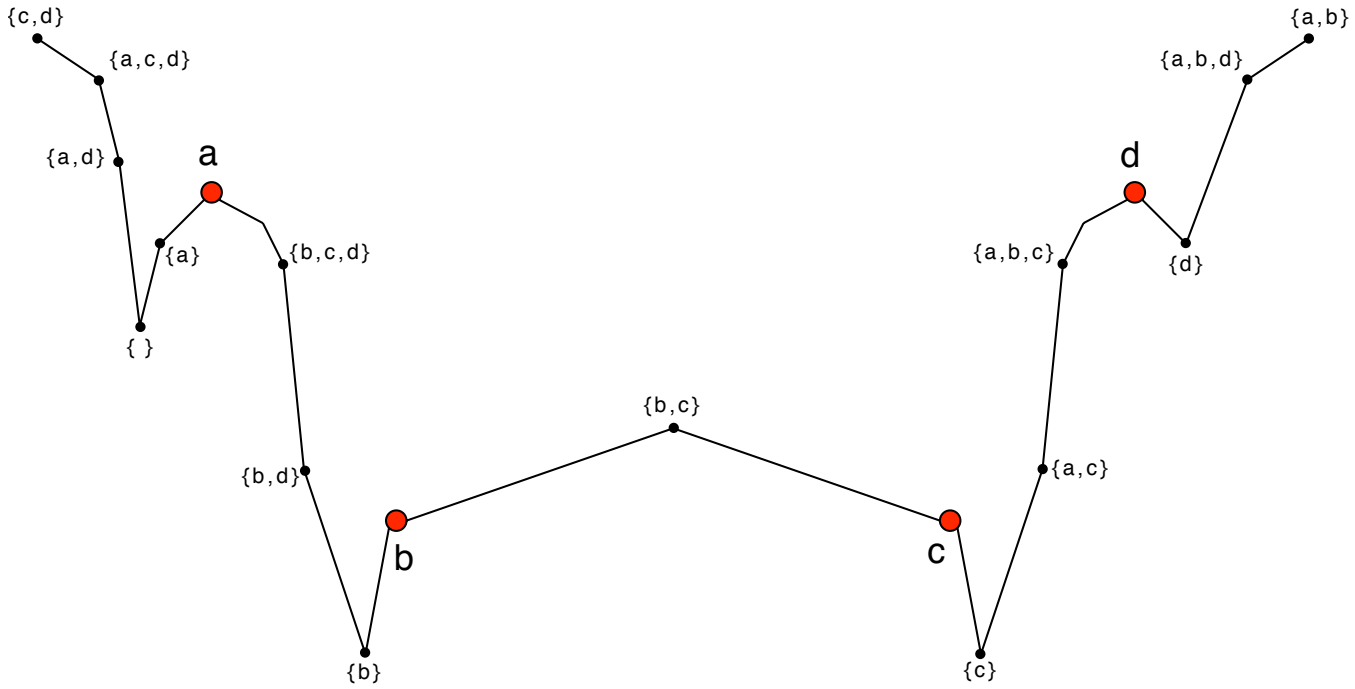


Figure 1: A monotone chain with 4 points, a, b, c, d, that are shattered by 16 guards. The guard seeing $\{a, b, c, d\}$ is not pictured, but a very high vertex on the left end of the terrain would see all other vertices. Each of the other 15 guards is labeled with the subset of $\{a, b, c, d\}$ that it sees.

proximability is unknown.

Isler *et al.* [8] consider guarding the exterior of polygons and polyhedra. For polygons they show that the maximum VC-dimension is 2 when guards must lie on a circle containing the polygon and 5 when guards can lie anywhere outside the convex hull of the polygon. For polyhedral galleries in $\mathbb{R}^3$ they prove that the maximum VC-dimension is unbounded when guards must lie on a sphere containing the gallery.

Our Contribution In section 2 we prove that the maximum VC-dimension of guarding a 1.5D terrain is 4 with matching upper and lower bounds. In section 3 we show that the VC-dimension of guarding a polygonal terrain is unbounded, via a reduction from polygons with holes.

2 VC-Dimension of Guarding 1.5D Terrains

To prove that a monotone chain can have VC-dimension 4, we simply provide an example of a terrain with 4 points that are shattered by 16 guards (see figure 1).

For points a, b on an x-monotone chain, we say that $a < b$ if a is to the left of b. The Order Claim [2] states that, for points a, b, c, d with $a < b < c < d$, if a sees c and b sees d then a sees d.

For any point set P that is shattered by a set of guards G let $g(p_1, \ldots, p_k)$ denote the guard in G that sees $p_1, \ldots, p_k \in P$ but no other points in P. We will now argue, using only the Order Claim, that no set P of size 5 can be shattered. This gives us the upper bound of 4 for the VC-dimension.

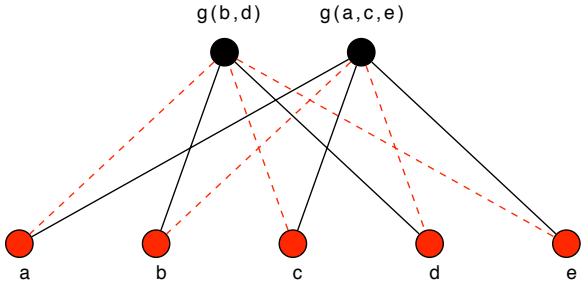
Let $P = \{a, b, c, d, e\}$ and assume without loss of generality that $a < b < c < d < e$. We can see (figures 2(a) and 2(b) may help) that $g(a, c, e)$ and $g(b, d)$ will contradict the order claim unless either

- $g(b, d) < c$ and $d < g(a, c, e)$, or
- $g(a, c, e) < b$ and $c < g(b, d)$.

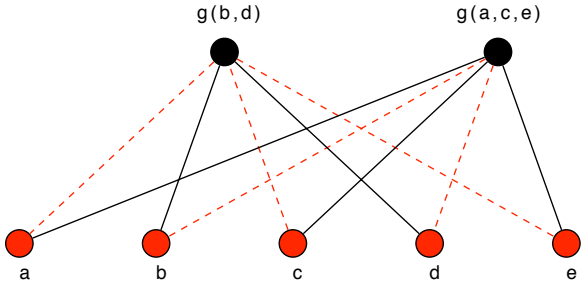
We assume the former without loss of generality. Now consider $g(b, c, e)$. There are four potential ranges that we consider placing $g(b, c, e)$ in:

- left of $g(b, d)$
- between $g(b, d)$ and d
- between d and $g(a, c, e)$
- right of $g(a, c, e)$.

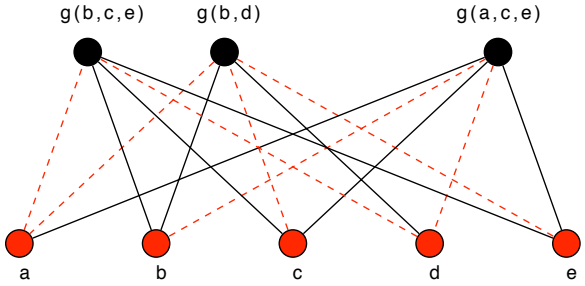
It is not difficult to verify that placing $g(b, c, e)$ in any of these four ranges would contradict the Order Claim (see figure 2(c) for an example). Therefore 5 points on a monotone chain cannot be shattered and no monotone chain can have VC-dimension greater than 4.



(a) In this configuration the Order Claim is contradicted by $g(b,d)$, $g(a,c,e)$, d, and e.



(b) In this configuration the Order Claim is not contradicted.



(c) The Order Claim is now contradicted by the addition of $g(b,c,e)$, regardless of its position. In this configuration the Order Claim is contradicted by $g(b,c,e)$, $g(b,d)$, c, and d.

Figure 2: Examples of configurations of G and P for the proof that no 5 points on a 1.5D terrain can be shattered. Solid lines indicate clear lines of sight. Dashed lines indicate blocked lines of sight.

3 VC-Dimension of Guarding 2.5D Terrains

Set Cover can be reduced to the problem of guarding the perimeter of a polygon with holes using guards on the perimeter (§4 of Eidenbenz *et al.* [5]). As a direct consequence, for any finite set system $(\mathcal{U}_1, \mathcal{S}_1)$, there exists a polygon with holes whose associated set system is $(\mathcal{U}_2, \mathcal{S}_2)$ such that $\mathcal{U}_1 \subseteq \mathcal{U}_2$ and $\mathcal{S}_1 \subseteq \mathcal{S}_2$. This implies that a polygon with holes can have arbitrarily large VC-dimension.

For any polygon A with holes we show how to construct a polygonal terrain of equal or grater VC-dimension. The idea behind building T is simple. Lines of sight between points on A are blocked by the exterior of A. On our terrain T we will build corresponding mountains to block lines of sight.

We start with T as a horizontal rectangle at altitude 0 that will act as a bounding box for A. We then trace the perimeter of A on this rectangle and call it A_T. A_T partitions T into two open sets, T^- which corresponds to the interior of A and T^+ which corresponds to the exterior of A, including the holes.

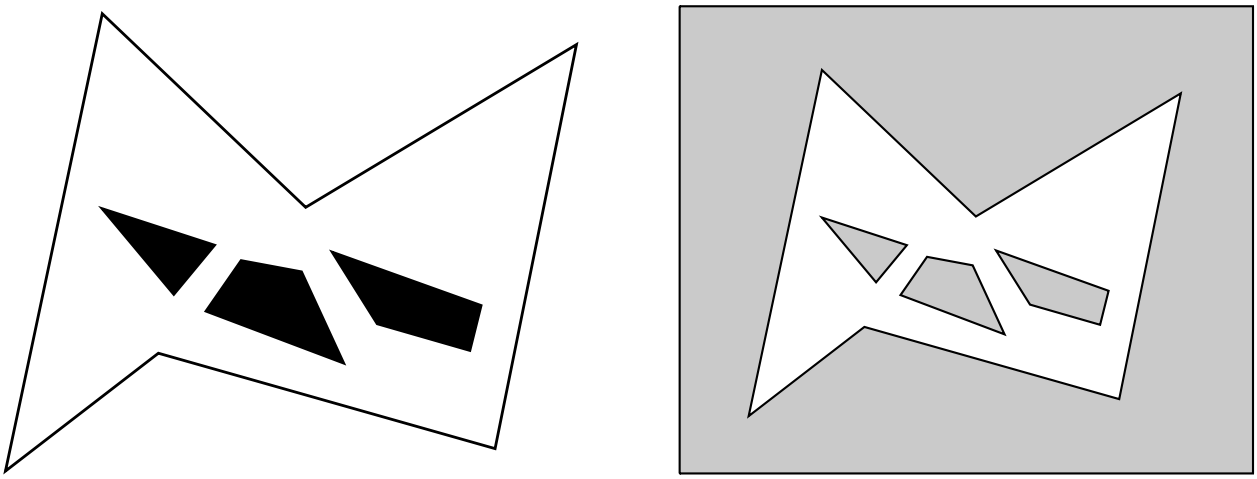
In terms of vertical projections, A_T, T^- and T^+ will remain fixed as we change T. However, T^- will be lowered and T^+ will be raised. There are many ways to perform this raising and lowering, but perhaps the most elegant is the method of raising roofs from *straight skeletons* (Aichholzer and Aurenhammer [1], in particular §4). We raise T^+ based on its straight skeleton and lower T^- based on its straight skeleton. The result is that every point in T^+ has positive altitude and every point in T^- has negative altitude. Only A_T and the rectangular perimeter of T will be at altitude 0. See figure 3 for an example.

We can now verify that two points p, q on A_T see each other if and only if the corresponding points p', q' on A see each other. Since p and q are both at altitude 0, all of (p, q) is at altitude 0. If p sees q then the open line segment (p, q) contains no point below T so no point on (p, q) can be the vertical projection of a point in T^+. The corresponding open line segment (p', q') therefore cannot intersect the exterior of A, so p' and q' must see each other. Therefore p sees q if and only if p' sees q', and the converse can be proved similarly.

From any polygon with holes, we can construct a 2.5D terrain with equal or greater VC-dimension, so 2.5D terrains have unbounded VC-dimension.

References

[1] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Computing and Combinatorics*, pages 117–126, 1996.



(a) The polygon A with holes indicated in black.

(b) A simplified top view of the associated terrain T. Black lines indicate A_T and the terrain's perimiter, both at altitude 0. T^- (white) has negative altitude while T^+ (shaded) has positive altitude.

Figure 3: A polygon A and a top view of the associated terrain T.

[2] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5D terrain guarding. *SIAM Journal on Computing*, 36(6):1631–1647, 2007.

[3] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(1):463–479, 1995.

[4] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *Journal of Symbolic Computation*, 7(1):11–30, 1989.

[5] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[6] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.

[7] M. Garey and D. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

[8] V. Isler, S. Kannan, K. Daniilidis, and P. Valtr. VC-dimension of exterior visibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):667–671, 2004.

[9] G. Kalai and J. Matoušek. Guarding galleries where every point sees a large area. *Israel Journal of Math*, 101(1):125–139, 1997.

[10] J. King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *LATIN 2006*, volume 3887 of *Lecture Notes in Computer Science*, pages 629–640, 2006.

[11] P. Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104(1):1–16, 1998.

Polygons Folding to Plural Incongruent Orthogonal Boxes

Ryuhei Uehara*

Abstract

We investigate the problem of finding orthogonal polygons that fold to plural incongruent orthogonal boxes. There are two known polygons that fold to produce two incongruent orthogonal boxes. In this paper, we show that there are infinite such polygons. We also show that there exists a tile that produces two incongruent orthogonal boxes.

1 Introduction

Polygons that can fold to a convex polyhedron have been investigated since Lubiw and O'Rourke posed the problem in 1996 [4]. Recently, Demaine and O'Rourke published a book about geometric folding algorithms that includes many results about such polygons [2, Chapter 25]. One of the many interesting problems in this area is that whether there exists a polygon that folds to plural incongruent orthogonal boxes. Biedl et al. answered "yes" by finding two polygons that fold to two incongruent orthogonal boxes [1] (see also [2, Figure 25.53]). However, are these two polygons exceptional? We show that the answer is "no." In this paper, we first report that there are more than two thousands such polygons of several sizes. These polygons were found by a randomized algorithm that repeatedly produces many nets of orthogonal boxes at random, and matches them in a huge hash table. Some of those polygons can be extended to general size. Using this fact, we also show that there exist an infinite number of polygons that can fold to two orthogonal boxes. Moreover, we show that there exists a simple polygon that can fold to two orthogonal boxes, and that tiles the plane. This pattern may be used to produce two kinds of boxes of two different volumes on demand without loss of material.

2 Preliminaries

In this paper, we concentrate on orthogonal polygons that consist of unit squares. For a positive integer S, we denote by $P(S)$ the set of three integers a, b, c with $0 < a \le b \le c$ and $ab + bc + ca = S$, i.e., $P(S) = \{(a, b, c) \mid ab + bc + ca = S\}$. Clearly, it is necessary to satisfy $|P(S)| \ge k$ to have a polygon of size $2S$ that can fold to k incongruent orthogonal boxes. For example, the two known polygons in [1] correspond to $P(11) = \{(1, 1, 5), (1, 2, 3)\}$ and $P(17) = \{(1, 1, 8), (1, 2, 5)\}$. Similarly, we have $P(15) = \{(1, 1, 7), (1, 3, 3)\}$, $P(23) = \{(1, 1, 11), (1, 2, 7), (1, 3, 5)\}$, $P(35) = \{(1, 1, 17), (1, 2, 11), (1, 3, 8), (1, 5, 5)\}$, $P(47) = \{(1, 1, 23), (1, 2, 15), (1, 3, 11), (1, 5, 7), (3, 4, 5)\}$, $P(59) = \{(1, 1, 29), (1, 2, 19), (1, 3, 14), (1, 4, 11), (1, 5, 9), (2, 5, 7)\}$, and so on.

Let B be an orthogonal box of size $a \times b \times c$. Then there are six faces that consist of two rectangles of size $a \times b$, $b \times c$, and $c \times a$, respectively. We regard each rectangle as a set of unit squares. That is, B consists of $2(ab + bc + ca)$ unit squares. Then, for B, we define a dual graph $G(B) = (V, E)$ of B as follows; V is the set of $2(ab + bc + ca)$ unit squares, and E contains an edge $\{u, v\}$ iff two unit squares u and v share an edge on B, or they are incident on B. It is easy to see that $G(B)$ is a 4-regular graph of $2(ab+bc+ca)$ vertices, and hence $|E| = 4(ab+bc+ca)$. Then we have the following observation:

Observation 1 *Let T be a spanning tree of $G(B)$ for some B. For every edge $\{u, v\}$ not in T, we cut the edge shared by two unit squares u and v on B. Then, we obtain a net P of B.*

That is, we can make a net P of B for any orthogonal box B. In the case, we say that the spanning tree T *produces* P. However, spanning trees themselves are not good to represent nets of a box. Suppose that a polygon P can fold to an orthogonal box B. In general, P contains a rectangle R of size $a \times b$ with $a > 1$ and $b > 1$. Then, no spanning tree T generates P since T forces unnecessary cuts of inside of R. The following lemma patches this problem.

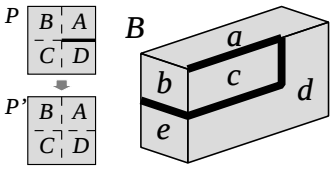


Figure 1: Gluing.

Figure 2: A half of a nonsimple polygon folding to a box.

Lemma 1 *Let P be a polygon that can fold to a box B. If P has a cut between two unit squares A and D*

*School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp

in Figure 1, we glue them and obtain P'. Then P' also can fold to B.

Proof. Since B is a convex orthogonal box, it follows. □

Repeating the gluing in Lemma 1, we obtain a polygon P that has no two consecutive identical edges, which means P contains no unnecessary cuts. From the viewpoint of programming, it is sufficient to represent each polygon P by a usual 0/1 matrix in a natural way, and ignore such cuts. One may think that any polygon that can fold to a box is simple. However, it is not the case.

Lemma 2 *Let B be an orthogonal box and P a polygon that can fold to B. Then, P is not necessarily simple.*

Proof. For B of size $1 \times 2 \times 3$, we make a (half of) polygon P as in Figure 2. Then, clearly, P is a polygon that can fold to B, but P is not simple. □

3 Algorithm

Our algorithm is a quite simple randomized one described below:

Input : S with $|P(S)| > 1$;
Output: Polygons of size $2S$ that fold to plural boxes;

clear a hash table H;
while *true* **do**
  choose a type $t = (a, b, c)$ in $P(S)$ at random;
  generate a spanning tree T of $G(B)$ for an orthogonal box B of size $a \times b \times c$ at random;
  represent a polygon P corresponding to T by a 0/1 matrix;
  if (t', P) *is in H with $t \neq t'$* **then** output P (and all associate types);
  if P *is not in H* **then** add (t, P) into H;
end

We aim at finding polygons shared by two or more types. Hence, the algorithm ignores weak points mentioned in Preliminaries. More precisely, the algorithm has the following flaws; (1) it does not generate the polygons *uniformly* at random, (2) some polygons overlap (by Lemma 2). Moreover, even if the polygon P does not overlap, two nonincident squares on the box B can share a common edge on P (by Lemma 2; if we have a cut between d and e in Figure 2, a, b, c, and d make a hole in P). Since the information is not represented on a 0/1 matrix, (3) some polygons P contain holes and have to be cut in differently to produce two distinct boxes.

Fortunately, the flaws cause few errors through our experiments; in fact, among 2165 outputs, the algorithm produced 2139 simple polygons, which are solutions, and only 26 non-simple polygons, which are not solutions. We note that from the algorithmic point of view, it is easy to check (2) and (3) in linear time when the algorithm outputs each solution.

Table 1: Experimental results (1)

$2S(S)$	$\lvert P(S)\rvert$	$\sim$RG($\times 10^7$)	Sols	Errs
22(11)	2	6.7	541	15
30(15)	2	18.6	72	1
34(17)	2	28.4	708	0
38(19)	2	30.4	41	0
46(23)	3	191.0	660	8
54(27)	3	126.7	3	0
58(29)	3	89.3	37	0
62(31)	3	82.4	5	0
64(32)	3	204.8	56	2
70(35)	4	91.3	14	0
88(44)	4	217.0	2	0
94(47)	5	51.3	0	0
118(59)	6	35.5	0	0
Total	-	-	2139	26

Table 2: Experimental results (2)

$2S(S)$	Types	Sols	Errs
46(23)	(1,1,11), (1,3,5)	568	3
	(1,2,7), (1,3,5)	92	5
54(27)	(1,1,13), (3,3,3)	2	0
	(1,3,6), (3,3,3)	1	0
58(29)	(1,1,14), (1,4,5)	37	0
62(31)	(1,3,7), (2,3,5)	5	0
64(32)	(1,2,10), (2,2,7)	50	2
	(2,2,7), (2,4,4)	6	0
70(35)	(1,1,17), (1,5,5)	3	0
	(1,2,11), (1,3,8)	11	0
88(44)	(2,2,10), (1,4,8)	2	0

4 Experimental results

We first ran the algorithm on a laptop (IBM ThinkPad X40: 1 Processor with 1.5GB Memory). This generated approximately 3×10^6 polygons in 1 hour, and obtained around 100 solutions for $P(11)$. To experiment more efficiently, we used a supercomputer (SGI Altix 4700: 96 Processors with 2305GB Memory). We used an implement of the Mersenne Twister algorithm[1] to generate random numbers. Our results are summarized in Tables 1 and 2. In Table 1, "$2S(S)$" denotes the (half) size of a polygon, "$|P(S)|$" denotes the number of distinct box types, "RG" denotes the number of random generations, "Sols" denotes the number of simple polygons that can fold to two incongruent orthogonal boxes, and "Errs" denotes the number of non simple polygons. For example, for $P(11)$, the algorithm generates around 6.7×10^7 nets of boxes of size $(1, 1, 5)$ *or* $(1, 2, 3)$, and we have 556 outputs. Among them, 15 polygons have a hole, and hence we have 541 distinct simple polygons that can fold

[1] `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`

to boxes of size $(1, 1, 5)$ *and* $(1, 2, 3)$. In total, we have 2139 distinct simple polygons that can fold to two incongruent orthogonal boxes. For each S with $|P(S)| > 2$, more details can be found in Table 2. All cases are checked in parallel on the machine, and the computations take from a few days to a few weeks (we stopped execution when each process requires too much memory). Some solutions are illustrated in Figure 7, and all solutions can be found at `http://www.jaist.ac.jp/~uehara/etc/origami/nets/index-e.html`. After these experiments, we still have no polygon that can fold to three (or more) incongruent orthogonal boxes. We note that some values of S are related; for example, the solutions for $P(11)$ give the solutions for $P(44)$ by dividing a unit square into four unit squares. Although we have 541 solutions for $P(11)$ after 6.7×10^7 random generations (it takes 3 days), we have only two solutions for $P(44)$ after 217.0×10^7 random generations (it takes 1 month). These two solutions for $P(44)$ do not correspond to any solution for $P(11)$. Some special polygons found in the solutions are below.

Tiling The discovered polygonal patterns reminded us of *tilings*. Indeed, there exists a simple polygon that can fold to two incongruent orthogonal boxes and it forms a tiling. The polygon in Figure 3 can fold to two boxes of size $1 \times 1 \times 8$ and $1 \times 2 \times 5$, and it tiles the plane.

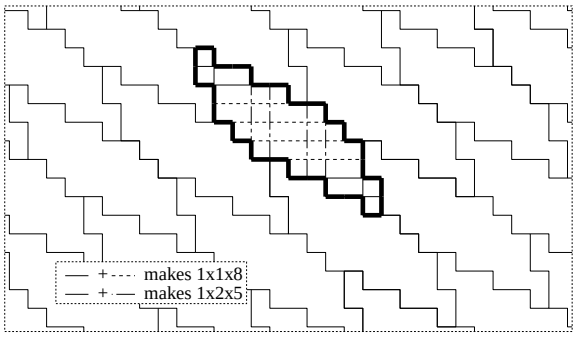


Figure 3: Polygon folding to two boxes of $1 \times 1 \times 8$ and $1 \times 2 \times 5$, and tiling the plane.

We note that the boxes with the common polygon form "*double packable solids*" introduced by Akiyama [3, Section 3.5.2]. Moreover, we can make two kinds of the boxes of volumes 8 and 10 on demand!

Disjoint crease patterns There exists a simple polygon that can fold to two incongruent orthogonal boxes and that foldings to two boxes are disjoint; the last polygon in Figure 7 satisfies the property.

Cross-free patterns There exists a simple polygon that can fold to two incongruent orthogonal boxes and that foldings to two boxes are cross free. The second last polygon in Figure 7 satisfies the property. We note that the previously known results in [1] also satisfy the property.

We have not checked if there exists a simple polygon such that foldings are disjoint and cross free.

5 Infinite polygons

A natural question is whether or not there are infinite distinct[2] polygons that can fold to plural boxes? The answer is "yes." Some polygons obtained by the experiments can be generalized. From two of them, we have the following theorem.

Theorem 3 *For any positive integer k, there is a distinct polygon that can fold to two incongruent orthogonal boxes of sizes (1)* $1 \times 1 \times (6k + 2)$ *and* $1 \times 5 \times 2k$*, and (2)* $1 \times 1 \times (8k + 11)$ *and* $1 \times 3 \times (4k + 5)$.

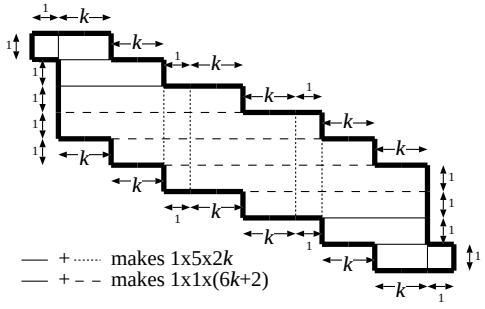


Figure 4: Polygon folding to two boxes of $1 \times 1 \times (6k+2)$ and $1 \times 5 \times 2k$ by stretch.

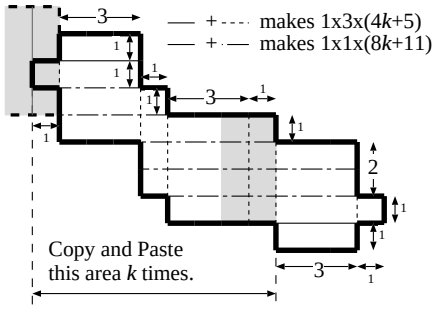


Figure 5: Polygon folding to two boxes of $1 \times 1 \times (8k+11)$ and $1 \times 3 \times (4k + 5)$ by spiral.

[2]Precisely, *distinct* means $gcd(a, b, c, a', b', c') = 1$ for two boxes of size $a \times b \times c$ and $a' \times b' \times c'$.

Proof. The first one is obtained by stretching a polygon. For any positive integer k, Figure 4 gives a polygon that satisfies (1). The second one is obtained by a spiral extension of a polygon. For any positive integer k, we copy in the leftside polygon in Figure 5 and glue it to the leftmost square (with overlapping at gray areas) and repeat it k times. Then the polygon satisfies (2). □

Corollary 4 *There exist an infinite of distinct polygons that can fold to two incongruent orthogonal boxes.*

6 Concluding remarks

From the theoretical point of view, uniform random generation and enumeration of all simple polygons for a given box are interesting problems. However, those algorithms are not necessarily useful to find polygons that can fold to plural incongruent orthogonal boxes. Indeed we search "similar" polygons heuristically to find such polygons.

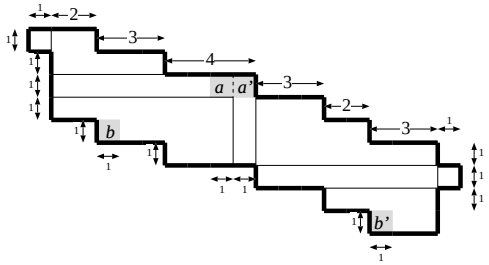


Figure 6: A polygon folding to two boxes of $1 \times 1 \times 17$, $1 \times 5 \times 5$, and "close" to $1 \times 3 \times 8$.

It is an open question if a polygon exists that can fold to three or more orthogonal boxes. The author conjectures "yes;" through experience, there is a polygon that seems to be "close" to the answer. The polygon in Figure 6 can fold to two boxes of size $1 \times 1 \times 17$ and $1 \times 5 \times 5$ in the similar ways in Figure 4. Moreover, it also can fold to the box of size $1 \times 3 \times 8$ with only two overlapping squares (and hence with two holes); a and b overlap with a' and b', respectively (with a cut between a and a').

Acknowledgements

The author thanks Joseph O'Rourke, who gave an interesting talk at JAIST, which involved the author in this work. The author also thank to Erik Demaine, who kindly sent the note [1] to the author.

References

[1] T. Biedl, T. Chan, E. Demaine, M. Demaine, A. Lubiw, J. I. Munro, and J. Shallit. Notes from the University of Waterloo Algorithmic Problem Session. Sept. 8 1999.

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

[3] M. Kano, M.-J. P. Ruiz, and J. Urrutia. Jin Akiyama: A Friend and His Mathematics. *Graphs and Combinatorics*, 23[Suppl]:1–39, 2007.

[4] A. Lubiw and J. O'Rourke. When Can a Polygon Fold to a Polytope? Technical Report Technical Report 048, Department of Computer Science, Smith College, 1996.

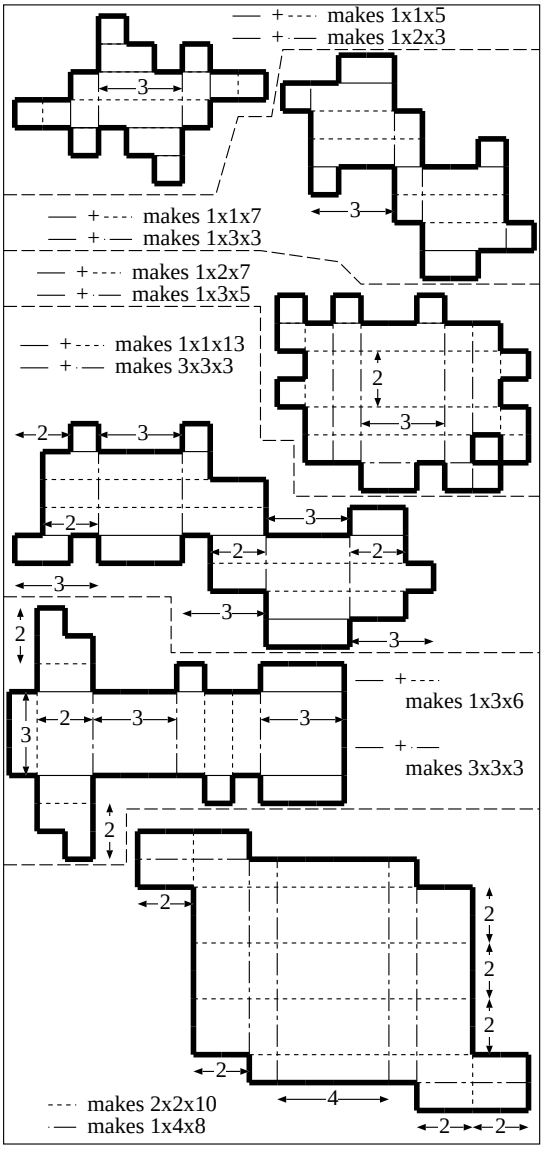


Figure 7: A part of solutions.

A Class of Convex Polyhedra with Few Edge Unfoldings*

Alex Benton† Joseph O'Rourke‡

Abstract

We construct a sequence of convex polyhedra on n vertices with the property that, as $n \to \infty$, the fraction of its edge unfoldings that avoid overlap approaches 0, and so the fraction that overlap approaches 1. Nevertheless, each does have (several) nonoverlapping edge unfoldings.

1 Introduction

An *edge unfolding* of a polyhedron is a cutting of the surface along its edges that unfolds the surface to a single, nonoverlapping piece in the plane. It has long been an open question of whether or not every convex polyhedron has an edge unfolding.[1] See [DO07, Chap. 22] for background and the current status of this problem.

An early empirical investigation of this question led to the conjecture that a random edge unfolding of a random convex polyhedron of n vertices leads to overlap with probability 1 as $n \to \infty$, under any reasonable definition of "random" [SO87].[2] It is easy to see that the cuts must form a spanning tree of the polyhedron vertices. It is known that there are $2^{\Omega(\sqrt{F})}$ cut trees for a polyhedron of F faces. So the conjecture says that "most" of the exponentially many cut trees lead to overlap. Of course, even if most unfoldings overlap in this sense, this is entirely compatible with the hypothesis that there always exists at least one non-overlapping unfolding.

No progress has been made on this random-unfolding conjecture (as far as we know), but Lucier [Luc06] was able to disprove several unfolding conjectures by carefully arranged polyhedra that force what he calls *2-local overlap*. Although not all our overlaps are 2-local, they are k-local (in Lucier's notation) for some small k, so our work can be viewed as following the spirit of his investigations.

In this note we construct an infinite sequence of convex polyhedra with the property that most of its unfoldings overlap, in the sense that, as $n \to \infty$, the number of its edge unfoldings that overlap approaches 1. A consequence is that no probabilistic argument could establish that every convex polyhedron has an edge unfolding.[3]

2 Banded Hexagon

The construction is based on a particular example from [O'R07], which showed that it is impossible to extend band unfoldings to obtain edge unfoldings of prismatoids. The details of the motivation for that work are not relevant here, but we employ its central construction, which we now describe.

Consider a hexagon formed by replacing each side of an equilateral triangle with two nearly collinear edges. The hexagon is then surrounded by a band of six identical quadrilaterals, forming a slight convexity at all edges. See Fig. 1. The six vertices of the hexagon A

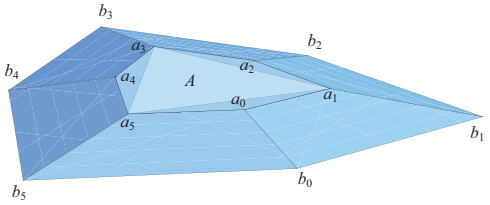


Figure 1: Banded hexagon from [O'R07].

are $(a_0, \ldots, a_5)$, and each is connected to its counterpart b_i on the outer rim of the band. The slight convexity means that the curvature at the a_i vertices is small. Cutting and flattening a vertex opens it by an amount equal to the curvature.

The key property of this *banded hexagon* is as follows.

Property 1 (Hexagon Overlap) *If only one band edge a_ib_i is cut (as part of the cut tree), so that the six quadrilateral faces of the band remain connected together, and all but one of the hexagon edges a_ia_{i+1} are cut, then the unfolding overlaps.*

Fig. 2(a-c) illustrates the opening at a_3, and (d-f) the opening at a_0. The other possibilities are symmetric.

3 Banded Geodesic Domes

For the purposes of [O'R07], the band quadrilaterals were chosen to be trapezoids. However, that is not an

*An earlier, full version is available at `http://arxiv.org/abs/0801.4019`

†DAMTP, Centre for Mathematical Science, Cambridge University, Cambridge CB3 0WA, UK. `A.Benton@damtp.cam.ac.uk`.

‡Department of Computer Science, Smith College, Northampton, MA 01063, USA. `orourke@cs.smith.edu`.

[1] `http://cs.smith.edu/~orourke/TOPP/P9.html#Problem.9`

[2] Data summarized in [DO07, p. 315].

[3] We owe this point to a referee.

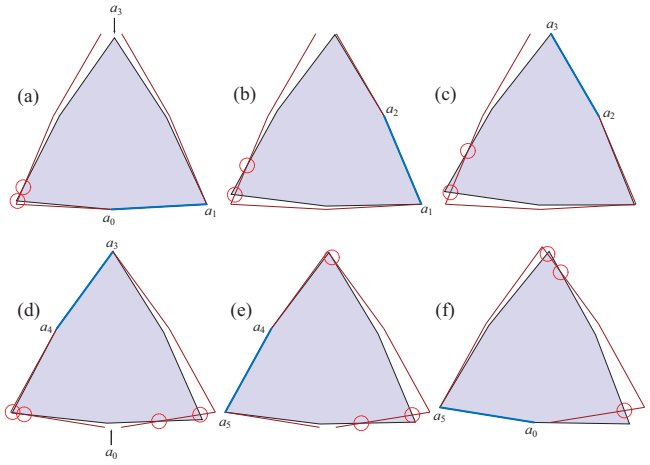


Figure 2: Placements of A when a_3 is cut (top row) and when a_0 is cut (bottom row). The attachment edge of the band to A is blue. Circles indicate overlap. The band lies outside the red rim. [Fig. 3 in [O'R07].]

essential property, and we modify the construction here so that the quadrilaterals remain congruent but are no longer trapezoids. The Hexagon Overlap property only relies on small curvature at the a_i, and the hexagon A having three acute angles (at $\{a_1, a_3, a_5\}$) interspersed with three nearly π-angles (at $\{a_0, a_2, a_4\}$). See ahead to Fig. 5.

With this flexibility, it is possible to glue together copies of the banded hexagon construction onto a triangulated surface. We choose to use "geodesic domes" as our base polyhedron (henceforth: *geodomes*), a repeated meshing starting with the icosahedron that has nearly equilateral faces. Fig. 3 illustrates two *levels* of the geodome construction, with each triangle face replaced by a banded hexagon. Let P_L be the banded geodome refined to level L. Level L=0 is based on the icosahedron. Level L=1 partitions each face of the icosahedron into four equilateral triangles, and projects to the circumscribing sphere. And so on. The number of faces, edges, and vertices of the completed construction for P_L are: F=140·4^L, E=300·4^L, n=V=160·4^L.

We can drive $n\to\infty$ by choosing larger and larger values of L. At L=3, there are $n = 10242$ vertices.

4 Unfoldings

Although the point of this note is that these banded geodomes are in some sense difficult to edge-unfold, in fact each of the P_L we constructed can unfold without overlap. Fig. 4 shows unfoldings found by a yet-to-be-thwarted unfolding procedure described in [Ben08]. Although we have not attempted to formally prove it, it seems likely that banded geodomes for any L can be edge-unfolded similarly, roughly by following the geodesics.

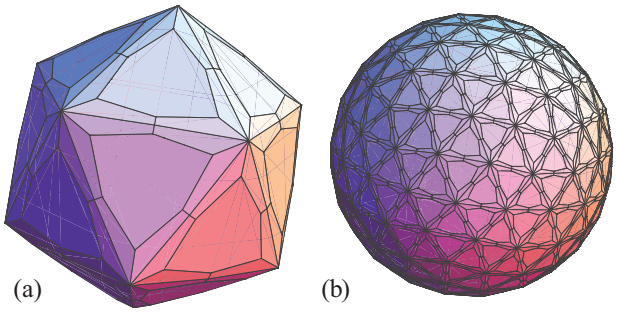


Figure 3: Banded geodomes for levels $L = 0, 2$.

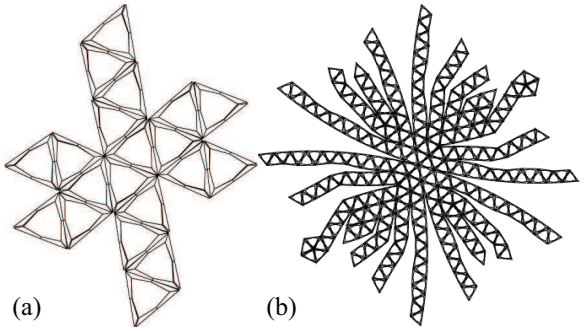


Figure 4: Edge unfoldings of the banded geodomes in Fig. 3.

All of these unfoldings have the property that each hexagon has two or more band cuts incident to its vertices (although these cuts are below the resolution of all but L=0 in Fig. 4(a)). We see how this avoids the Hexagon Overlap property in the next section.

5 Proof

Overview. The proof has the following overall structure. First we establish that at least a positive fraction $\rho > 0$ of all cut trees that span a finite-sized connected region C of the surface of P_L satisfy the Hexagon Overlap property, and so force unfolding overlap. Thus, at most $(1-\rho)$ of those trees avoid overlap. Then a cut tree that avoids overlap everywhere in the unfolding must avoid local overlap in each of these regions. Because the regions are a finite-size, as $L\to\infty$, the number k of regions also gets arbitrarily large. Thus the fraction of trees that avoid overlap everywhere is at most $(1-\rho)^k$, which goes to 0 as $k\to\infty$.

Connection Tree. The cut tree T is a spanning tree of the polyhedron vertices. The dual *connection* tree $T^\triangle$ is a spanning tree of the faces. In $T^\triangle$, two face nodes are connected if the faces share an uncut edge. T and $T^\triangle$ each uniquely determine the other. In this section we reason mostly with $T^\triangle$.

One Hexagon. Focus on one hexagon A of the polyhedron P. Referring to Fig. 5, let $e_i = a_i a_{i+1}$, and $u_i = a_i b_i$. The conditions that lead to Hexagon Overlap are: exactly one e_i is not cut, and exactly one u_i is cut. In terms of the dual tree $T^\triangle$, this means that the hexagon is a leaf node, surrounded by a band path of length 5, as in the figure. Clearly there are 6^2 such dual tree patterns leading to Hexagon Overlap (6 choices for e_i and 6 for u_j), when one banded hexagon is considered in isolation.

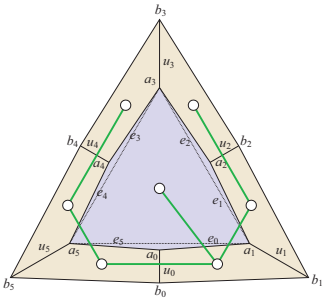


Figure 5: e_0 is not cut and u_3 is cut. All other e_i are cut and all other u_j are not cut. Dual tree $T^\triangle$ is shown.

Tiling Clusters. Now we consider a group of 16 banded hexagons, which together form a nearly equilateral triangular cluster, as shown in Fig. 6. Let h be the central

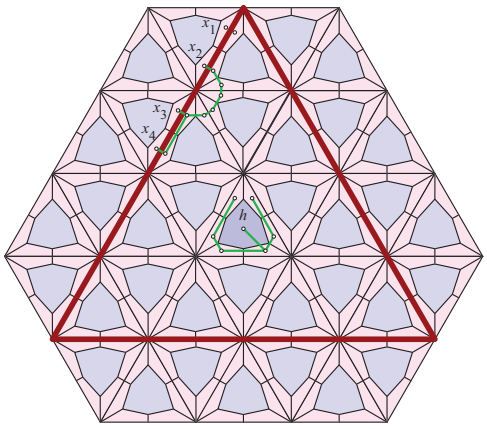


Figure 6: C: 16 banded hexagons, with central h. $x_1, \ldots, x_{24}$: surrounding quadrilateral nodes.

banded hexagon in a cluster C. The choice of the size and shape of C is somewhat arbitrary. Our specific choice is motivated by two concerns: (1) The surface of P_L is nearly an equilateral lattice tiling of banded hexagons, and so can itself be tiled by copies of the nearly equilateral C, for appropriate L. (2) The central h is sufficiently "buffered" from the boundary of C, in this case by the 15 other banded hexagons of C, for a counting argument to go through. Both of these points will be revisited below.

Counting Overlapping Trees. We now argue that there are at least a positive fraction $\rho > 0$ of trees spanning C that induce local overlap.

Let $T^\triangle$ be a dual spanning tree of P, and denote by $G^\triangle$ the forest with all nodes in C deleted. There are in general many ways to complete $G^\triangle$ to be a spanning tree of P. The exact number of completions is difficult to count because it depends on the structure of $G^\triangle$. However, we can easily obtain a crude upper bound as follows. Let E_C be the number of dual edges in C; an explicit count shows that $E_C = 228$. Any completion must either use or not use each dual edge in C. Of course many of these "bit patterns" will not complete $G^\triangle$ to a tree, or not to a spanning tree. But every valid completion corresponds to one of these bit patterns. Therefore, the total number of completions m satisfies $m \leq 2^{E_C}$.

Let o be the number of completions of $G^\triangle$ that lead to unfolding overlap. Again it would be difficult to count o exactly, but we know that the 36 patterns leading to Hexagon Overlap in h must be avoided, for each forces local overlap. Moreover, because of the buffer around h in C, all of these 36 patterns are part of some valid completion, regardless of the structure of $T^\triangle$ outside C. We justify this last claim below, but for now proceed with the argument, assuming $o \geq 36$.

Let $\rho = o/m$ be the fraction of completions of $G^\triangle$ that lead to overlap. We have a lower bound on o and an upper bound on m, so together they provide a lower bound on the ratio ρ: $\rho \geq 36/2^{228} \approx 10^{-67}$. The exact value of this fraction ρ is not relevant to the argument; we only need that $\rho > 0$ so that $1-\rho < 1$.

Buffering. We return to the claim that h is sufficiently buffered within C so that for each tree that spans C, there are at least the 36 overlapping variants identified above. First we explain why the more natural choice of $C = h$ does not suffice. Suppose the forest $G^\triangle$ has a structure such that choosing an edge dual to u_i within h creates a cycle. Then it is not a option to select this edge to complete $G^\triangle$ to a tree. If this occurs for two or more of the u_i, then the Hexagon Overlap pattern of Fig. 5 cannot occur within h. Thus, the structure of $G^\triangle$ outside C forces avoidance of the Hexagon Overlap property inside C. Thus, not every C contains a hazard to be avoided, so to speak. We now show that our choice for C provides sufficient buffering.

Let $x_1, x_2, \ldots, x_{24}$ be the 24 quadrilateral nodes surrounding and just outside C, each with a dual edge that crosses into C. Each can be viewed as the root of a tree in the forest $G^\triangle$. We now show that the 36 critical patterns are part of some completion of $G^\triangle$ to a tree that spans C and therefore all of P_L. We first connect up all these trees in the forest into one tree via connections through the quadrilaterals incident to the border of C. One way to do this is to proceed sequentially from x_1 to

x_{23}, connecting x_i to x_{i+1} if their two subtrees are not yet connected, but not making the connection if they already are part of the growing connected component. (For example, in Fig. 6, perhaps x_1 does not need to be connected to x_2, but $\{x_2, x_3, x_4\}$ should receive connections.) This connects all of $G^\triangle$ into a single tree without employing any of the nodes of the central h. For each of the 36 overlap patterns for h, we are free to connect up the remainder of C into a spanning tree structure, which clearly can be done in many ways. Therefore, for any tree that spans P_L and C, there are at least 36 variants inside C that overlap, and so $o \geq 36$.

Global Argument. Let $H = 20{\cdot}4^L$ be the number of hexagons in the polyhedron P_L. We showed above that at most $1-\rho$ of the dual cut tree patterns inside a given cluster avoid overlap there (for if we fall into the ρ fraction, overlap is forced).

Imagine now constructing a complete tree $T^\triangle$ cluster-by-cluster in the tiling, by choosing all the nodes and arcs in $T^\triangle$ that span one cluster C, before moving to the next cluster. This is would be an odd way to build the tree, but with appropriate foresight, any tree could be constructed in this manner. Selecting the subforest to span a particular C leads us into the analysis of above: no matter what the structure of $G^\triangle$ already fixed outside of C, there is a fraction ρ of subforests that must be avoided inside C.

In order to avoid overlap in the complete unfolding, one of these overlap-avoiding patterns must be selected for each of the $\lfloor H/16 \rfloor$ clusters that tile the surface. Thus, the fraction of trees that avoid overlap within all clusters simultaneously is at most $(1-\rho)^{\lfloor H/16 \rfloor}$.

Finally, as $L \to \infty$, $H \to \infty$, and the overlap-avoiding fraction of all unfoldings goes to 0, while the overlap fraction goes to 1. This is the main claim of this note.

6 Empirical Data

The argument above only establishes a (very) loose upper bound on the ratio of the overlap-avoiding unfoldings to the total number of unfoldings. The looseness of the argument is dramatically revealed by empirical results. For the L=0 banded geodome, our bound says that the overlap fraction is at least 10^{-67}, whereas we found that out of 5.5 million random cut trees, 99.9998% of the corresponding unfoldings overlap.

Some understanding of this high frequency of overlap is provided by the empirical observation that, in our random unfoldings, about 70% unfolded the seven faces of a banded hexagon connected together as a unit. This fraction is stable and apparently independent of L (and therefore of n).[4] And when a banded hexagon is unfolded as a unit, the empirically observed frequency of local overlap is about 50%. Thus, we would expect the fraction $1 - (1-0.7{\cdot}0.5)^H$ of all unfoldings to overlap. For L=0, H=20, this formula (using more accurate frequencies) evaluates to 99.97%. This suggests that local overlap (within one banded hexagon unit) accounts for the majority of overlaps, for counting all overlaps only increases the frequency to 99.9998%.

Another test establishes the empirical difficulty of unfolding banded polyhedra. In Fig. 7 we compare the percentages of random unfoldings that overlap among banded simplicial polyhedra, with the similar percentage for random convex polyhedra with the same number of faces. Although both curves approach 100% as F increases (in accord with the [SO87] conjecture), the banded curve approaches 100% much more rapidly.

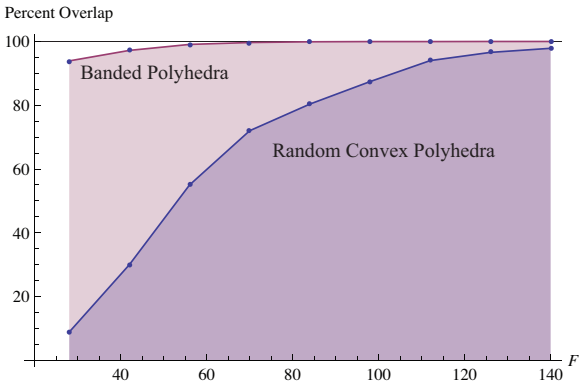


Figure 7: Percent of random unfoldings that overlap, for banded polyhedra and for random convex polyhedra (convex hulls of random points on a sphere). The F=140 point corresponds to the L=0 geodome.

Acknowledgments. The second author thanks Erik Demaine for several enlightening discussions on the topic of this paper.

References

[Ben08] Alex Benton. *Unfolding Polyhedra.* PhD thesis, Cambridge University, March 2008.

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

[Luc06] Brendan Lucier. Local overlaps in special unfoldings of convex polyhedra. In *Proc. 18th Canad. Conf. Comput. Geom.*, pages 97–100, 2006.

[O'R07] Joseph O'Rourke. Band unfoldings and prismatoids: A counterexample. Technical Report 087, Smith College, October 2007. arXiv:0710.0811v2 [cs.CG]; `http://arxiv.org/abs/0710.0811`.

[SO87] Catherine Schevon and Joseph O'Rourke. A conjecture on random unfoldings. Technical Report JHU-87/20, Johns Hopkins Univ., Baltimore, MD, July 1987.

[4] We have not attempted a theoretical explanation for this data.

Inverting Linkages with Stretch

Youichi Fujimoto* Mitsuo Motoki† Ryuhei Uehara‡

Abstract

We consider inversion of linkages on 2-dimensional plane. Inversion of a linkage is a transform by continuous moves of vertices to its mirror image. There exist noninvertible linkages due to fixed-length links. To invert such noninvertible linkages, we have to relax some constraints. We allow variable-length links instead of fixed-length links since any linkage can be invertible by expanding sufficiently all links. We introduce a notion "stretch ratio" as a measure of length change, and analyze upper/lower bounds to invert polygons, outerplanar graphs, and wheels.

1 Introduction

A *linkage* is a collection of line segments joined at their endpoints to form a graph. A segment endpoint is called *joint* or *vertex*. The line segments are called *links*. In ordinary linkage, the length of each link is fixed. While we can ignore the length of edges in general graph embedding, a linkage must be embedded with keeping the lengths of edges. We allow self-intersection, i.e., any pair of links or vertices, they can not only intersect each other, but locate on the same position. In this paper, we consider linkages on Euclidean 2-dimensional plane.

A linkage is considered as a physical model which consists of straight rigid bars and joints permitting arbitrary bend. Linkages have a number of practical applications such as smooth graph deformation, analysis of physical structures, motion planning in robotics and molecular modeling.

A *configuration* of a linkage is a specification of the location of all the vertices. Given an initial configuration and a final configuration, the *reconfiguration* is continuous moves of vertices from the initial configuration to the final one, keeping all links rigid. Due to rigidity of links, there may exist pairs of initial and final configurations without reconfiguration. It is known that it is PSPACE hard to determine whether there exists reconfiguration for a given initial configuration and final configuration of a linkage on 2-dimensional plane [3]. *Inversion* of a linkage is a special reconfiguration from a given configuration to its mirror-image configuration. We say that a linkage is *invertible* if there exists the inversion of a given configuration. For example, any triangle linkage has unique configuration, and thus noninvertible. Lenhart and Whitesides showed it is easy to distinguish invertible polygon on 2-dimensional plane [4].

It is easy to see that every linkage can be invertible if we can change the length of links. Hence, by relaxing the constraints on the length of links, that is, using variable-length links instead of fixed-length links, we can invert even noninvertible linkages. It is practical to consider such linkages. For example, in molecular modeling, the distances between molecules are not strictly fixed [2]. Another example is telescoping arms in robotics. We introduce a notion "stretch ratio" to denote how much each link can be stretched. When we can independently stretch any link with length l from l/α to αl for some constant $\alpha \geq 1$, we say that the *stretch ratio* of the linkage is α. Note that $\alpha = 1$ means ordinary linkages.

Thus the problem now is to obtain the minimum stretch ratio to invert a given linkage, that is, we extend decision problem to optimization problem which are more practical problem of minimizing the stretch ratio.

We analyze upper and lower bounds of stretch ratio to invert polygons, outerplanar graphs, and wheels. For polygons and outerplanar graphs, we show a constant tight upper bound. We can also calculate the optimal stretch ratio for each polygon or outerplanar graph. On the other hand, we show a pessimistic results for general planar graphs. We show there is no constant upper bound of stretch ratio to invert wheels. A good news is that we can invert a wheel with an optimal stretch ratio with few exceptions.

2 Inverting Polygons with Stretch

As mentioned before, Lenhart and Whitesides [4] proved the necessary and sufficient condition of invertible polygon on 2-dimensional plane.

Theorem 1 *[4] A polygon is invertible iff the lengths of the second and third longest links sum to no more than the sum of the lengths of the remaining links.*

Thus we consider noninvertible polygons only. First, we show an upper bound of stretch ratio to invert polygons.

Theorem 2 *Any polygon can be invertible within stretch ratio at most* $\sqrt{2}$.

Proof. First we show that, for any polygon, we can divide into two paths so that the length of the longer path is at most

*School of Information Science, Japan Advanced Institute of Science and Technology

†School of Information Science, Japan Advanced Institute of Science and Technology, mmotoki@jaist.ac.jp

‡School of Information Science, Japan Advanced Institute of Science and Technology, uehara@jaist.ac.jp

twice of the length of the shorter path. For any two vertices v and u, let $L_l(v_1,v_2)$ ($L_s(v_1,v_2)$, resp.) be the length of the longer (shorter, resp.) path. Now suppose v_1 and v_2 be vertices where $L_l(v_1,v_2)-L_s(v_1,v_2)$ is the minimum among all the pairs of two distinct vertices. We can also divide the longer path at a vertex v_3 so that the length $L_s(v_2,v_3)$ of path between v_3 and v_2 (or v_1, but w.l.o.g. we can assume v_2) is at most half of $L_L(v_1,v_2)$ because of the triangle inequality. Thus, if $L_l(v_1,v_2) > 2L_s(v_1,v_2)$, we can easily see

$$\begin{aligned} &L_l(v_1,v_3)-L_s(v_1,v_3)\\ &\quad=(L_s(v_1,v_2)+L_s(v_2,v_3))-(L_l(v_1,v_2)-L_s(v_2,v_3))\\ &\quad=L_s(v_1,v_2)-L_l(v_1,v_2)+2L_s(v_2,v_3)\\ &\quad\le L_s(v_1,v_2)<L_l(v_1,v_2)-L_s(v_1,v_2). \end{aligned}$$

This is a contradiction.

Therefore, by shortening the longer path by factor $1/\sqrt{2}$ and lengthening the shorter path by factor $\sqrt{2}$, we can invert any polygon with in stretch ratio at most $\sqrt{2}$. □

We remark that this upper bound $\sqrt{2}$ is tight. The worst case is an equilateral triangle. While inverting an equilateral triangle, one vertex must cross the opposite link. Thus the stretch ratio $\sqrt{2}$ is necessary and sufficient.

We can also compute the optimal stretch ratio to invert a given polygon.

Theorem 3 *If a polygon with n links is not invertible, the optimal stretch ratio of the polygon is $\sqrt{\frac{l_2+l_3}{\sum_{i=1}^{n} l_i-(l_2+l_3)}}$, where l_i is the length of the i-th longest link.*

Proof. In the inversion procedure, we first lengthen or shorten each link, then invert the polygon and finally restore each link to the original length.

First, we show that we can obtain an optimal stretch ratio by determining the second and third longest links after the first step. Let l_i' be the length of the i-th longest link after first lengthening or shortening. Since the obtained polygon is invertible, we have $l_2'+l_3' = \sum_{i=1}^{n} l_i' - (l_2'+l_3')$ by Theorem 1. We call the links with length l_2' and l_3' by shortened links, and the others are called lengthened links. On the inversion with optimal stretch ratio, we can assume that we all shorten (lengthen, resp.) the shortened (lengthened, resp.) links by an identical ratio. Suppose links with length l_i and l_j are the shorten links ($1 \le i < j \le n$). Then the stretch ratio is given by $\alpha_{i,j} = \sqrt{\frac{l_i+l_j}{\sum_{k=1}^{n} l_k-(l_i+l_j)}}$.

By considering following 3 cases, we show that $\alpha_{2,3}$ is the minimum, which concludes the proof.

Case 1 $i=1$ **and** $j \in \{2,3\}$**:** It is easy calculation to show $\alpha_{1,2} \ge \alpha_{2,3}$ and $\alpha_{1,3} \ge \alpha_{2,3}$.

Case 2 $i=3$ **and** $j \ge 4$**:** In this case, the j-th longest link lengthen to the third longest link. This means that during lengthening and shortening links, the lengths of links with original length l_2 and l_j will match. Let $\beta = \sqrt{l_2/l_j}$ be the stretch ratio at the moment. It is easy to see $\beta \ge \alpha_{2,3}$.

Case 3 otherwise: We can show similar to the case 2 by replacing l_2 with l_3. □

It is easy to see that we can apply these results to outerplanar graphs. We say that a cycle in an outerplanar graph is *minimal* if it does not include other cycle inside in the outerplanar embedding of the graph. Each edge is included in at most two minimal cycles. By constructing a dual graph and removing a vertex corresponding to the outer plane, we have a tree. Therefore we can invert each minimal cycle one by one in depth-first-search order. While inverting one minimal cycle, we stretch the other cycles similar to chords. Therefore, we have the following corollaries.

Corollary 4 *Any outerplanar graph can be invertible within stretch ratio at most $\sqrt{2}$.*

Corollary 5 *If an outerplanar graph with n vertices is not invertible, the optimal stretch ratio is computable in $O(n)$ time.*

3 Hardness of Inverting Wheels with Stretch

It is a natural question to ask whether any planar graph is invertible within some constant stretch ratio. Unfortunately, we can prove that there is no constant upper bound of the stretch ratio to invert general planar graphs.

We show that wheels has no constant upper bound of the stretch ratio. Let W_n be a wheel with an equilateral $(n-1)$-gon $v_0,v_1,v_2,\ldots,v_{n-3},v_{n-2},v_0$ and a center vertex c on the barycenter. Let $e_i = \{v_i,v_{i+1}\}$ for each i with $0 \le i \le n-3$ and $e_{n-2} = \{v_0,v_{n-2}\}$.

We first show that the center vertex c must traverse one edge of the outer polygon.

Lemma 6 *When W_n is inverted, c traverses a point on e_i for some i with $0 \le i \le n-2$.*

Proof. We employ topological deformations. First, we reduce the cycle $(v_0,v_1,\ldots,v_{n-2},v_0)$ to a closed curve and a center vertex c on the plane (Figure 1(a)(b)). Similarly, the inverse of W_n can be reduced to a closed curve and the same center point c (Figure 1(d)(c)). Then those curves are inverted with respect to the center c. That is, the winding numbers of the curves around c are $+1$ and -1, respectively. Hence, to invert the curve, the winding number has to be changed from $+1$ to -1. We cannot change the number without crossing the center c over the curve. Therefore, c traverses a point on the cycle. □

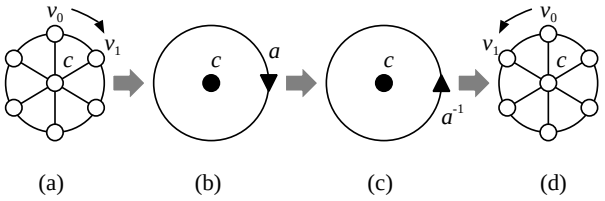


Figure 1: Inverting a wheel

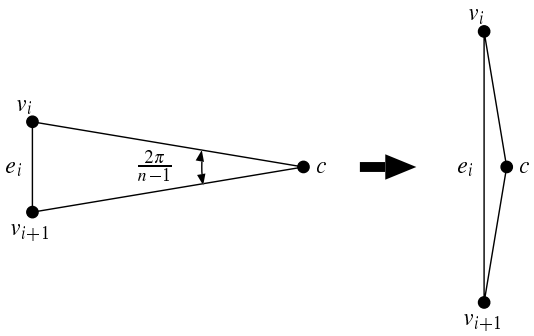


Figure 2: A triangle cv_iv_{i+1}

Theorem 7 *For any constant $\alpha > 1$, a wheel W_n is not invertible within stretch ratio α, where n satisfies $\sqrt{1/\sin\frac{\pi}{n-1}} > \alpha$, i.e., $n > 1 + \pi/\arcsin\frac{1}{\alpha^2}$.*

Proof. By Lemma 6, the center vertex c must traverse an edge of outer polygon. Suppose c traverses e_i (Figure 2). Thus we have to shorten links cv_i and cv_{i+1} by factor $\sqrt{\sin\frac{\pi}{n-1}}$ and to lengthen e_i by factor $1/\sqrt{\sin\frac{\pi}{n-1}}$. This stretch ratio is monotone increasing with n. □

On the other hand, we also have the following positive result.

Theorem 8 *For $n = 7$ or $n \geq 9$, we can invert a wheel W_n within stretch ratio $\sqrt{1/\sin\frac{\pi}{n-1}}$ with $O(n)$ motions.*

Since this upper bound is equal to the lower bound in Theorem 7, this is the optimal stretch ratio.

The proof is constructive. Due to the page limitation, we omit the algorithm description and proof. Hereafter, we show an algorithm sketch only.

Let W_n be a wheel with an equilateral $(n-1)$-gon $v_0, v_1, v_2, \ldots, v_k, v_{-k}, v_{-(k-1)}, \ldots, v_{-1}, v_0$ and a center vertex c on the barycenter, where $k = \lfloor (n-1)/2 \rfloor$ and if n is odd v_k and v_{-k} are identical. Let m be a line passes through c and v_0. We design an algorithm that reconfigures a wheel W_n onto the line m. In each step, we move 2 or 4 vertices onto m. The algorithm consists of three phases.

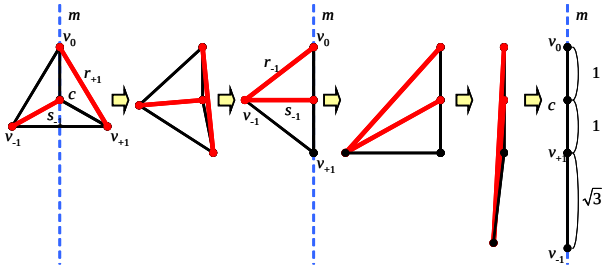


Figure 3: Inversion of W_4

The first phase is used when n is even, i.e., v_k and v_{-k} are not identical. In this phase, we move v_k and v_{k-1} onto m. We remark that, if n is odd, v_k is already on m.

During the second phase, we move 4 vertices on m at once. At i-th iteration, we move v_i, v_{-i}, v_{k-i}, and $v_{-(k-i)}$.

If 2 vertices remains after $\lfloor k/2 \rfloor$ iterations, i.e., if k is even, we proceed to the third phase. In this phase we move $v_{k/2}$ and $v_{-k/2}$ onto m.

Our algorithm, however, cannot invert neither W_4, W_5, W_6, nor W_8 within stretch ratio $\sqrt{1/\sin\frac{\pi}{n-1}}$. For example, a lower bound of the stretch ratio to invert W_4 is $\sqrt{2}$ ($> \sqrt{1/\sin\frac{\pi}{3}}$) since the outer polygon of W_4 forms an equilateral triangle. For W_4, we can also show the following nontrivial upper bound (see Figure 3).

Theorem 9 *W_4 can be invertible within stretch ratio $\sqrt{1+\sqrt{3}}$.*

4 Concluding Remarks

Of course, inversion linkages with stretch in 3 or higher dimension space is also included in future works.

Another future work is the problem to compute the optimal ratio to invert a given general graph. It might be easier to determine whether a given graph can be inverted with a given stretch ratio. We conjecture that the problems are PSPACE-hard in general. For example, fix the stretch ratio to 1, that is, without stretching. Then the problem becomes the decision problem that asks if a given linkage can be inverted in 2D space. This is a special case of the usual linkage reconfiguration problem. The linkage reconfiguration problem is PSPACE-hard even for trees in general [1].

References

[1] H. Alt, C. Knauer, G. Rote and S. Whitesides. On the Complexity of the Linkage Reconfiguration Problem. B 03-02, Freie Universität Berlin", 2003.

[2] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms*. Cambridge University Press, 2007.

[3] J. E. Hopcroft, D. Joseph, S. Whitesides. Movement Problems for 2-Dimensional Linkages. *SIAM J. Comput.*, 13(3): 610–629, 1984.

[4] W. J. Lenhart and S. H. Whitesides. Reconfiguring closed polygonal chains in Euclidean d-space. *Discrete Comput. Geom.*, 13:123–140, 1995.

Polynomial irreducibility testing through Minkowski summand computation

Deepanjan Kesh and Shashank K Mehta*

Abstract

In this paper, we address the problem of deciding absolute irreducibility of multivariate polynomials. Our work has been motivated by a recent work due to Gao et. al. [1, 2, 3] where they have considered the problem for bivariate polynomials by studying the integral decomposability of polygons in the sense of Minkowski sum. We have generalized their result to polynomials containing arbitrary number of variables by reducing the problem of Minkowski decomposability of an integer (lattice) polytope to an integer linear program. We also present experimental results of computation of Minkowski decomposition using this integer program.

1 Introduction

Let $f = \sum_\alpha c_\alpha X^\alpha$ be a polynomial where $\alpha \in \mathbb{N}^n$ and the coefficients c_α are from a field, say K. The lattice polytope $New(f) = conv(\{\alpha | c_\alpha \neq 0\})$ is called the Newton polytope of f. A lattice polytope $\mathcal{P}$ is *integrally decomposable* if there exist non-trivial lattice polytopes $\mathcal{Q}$ and $\mathcal{R}$ such that $\mathcal{P}$ is their Minkowski sum, denoted as $\mathcal{Q} + \mathcal{R}$. Ostrowski [4] observed that if f, g, h are polynomials such that $f = g \cdot h$, then $New(f) = New(g) + New(h)$. This gives a simple irreducibility criterion for polynomials [2].

Lemma 1 *Let $f \in K[x_1, \ldots, x_n]$ and it is not divisible by any x_i for any i. If the Newton polytope of f is integrally indecomposable, then f is absolutely irreducible.*

Thus the integral indecomposability of the Newton polytope is a sufficient condition for testing the absolute irreducibility of a polynomial. Efficient decomposition algorithms are given by Silverman and Stein [9] and Emiris and Tsigaridas [8] for polygons and by Mount and Silverman [7] for 3-dimensional polytopes. Gao and Lauder [1] showed that the problem is NP-complete even in two-dimensions. They gave a pseudo-polynomial time algorithm to solve the integral decomposition of polygons, and a randomized heuristic algorithm for polytopes of higher dimensions [1, 3]. We present an exact criterion for integral decomposition of arbitrary dimensional lattice polytopes. We show that an integral decomposition of a polytope exists if and only if its edge-graph has a graph-minor satisfying certain conditions. The criterion is general and applies to non-lattice polytopes as well. In the rest of the discussion, polytopes or convex polytopes would refer to lattice convex polytopes unless stated otherwise.

*Department of Computer Science and Engineering, IIT Kanpur, Kanpur - 208016, {deepkesh, skmehta}@cse.iitk.ac.in

2 Oriented Walks and Oriented Weights

An *oriented walk* in an undirected graph $G = (V, E)$ is a non-empty sequence of vertices $w = v_0, \ldots, v_k$, not necessarily distinct, such that $e_i = v_i v_{i+1}$ is an edge of G for all $0 \leq i < k$. The orientation of e_i in w is in the direction $\overrightarrow{v_i v_{i+1}}$. We denote the walk in the reverse orientation, $v_k, v_{k-1}, \ldots, v_0$, by w^r. If $v_0 = v_k$, then the oriented walk is said to be *closed*. An oriented closed walk $v_0, \ldots, v_{k-1}, v_0$ with $k \geq 3$ is said to be a *simple* if $v_i \neq v_j$ for all $0 \leq i < j \leq k-1$. Simple closed walks are also called *cycles*. All closed walks of the form $v_0, v_1 \ldots, v_{k-1}, v_k, v_{k-1}, \ldots, v_1, v_0$ are called *zero-walks*.

We define the *oriented sum* of two oriented closed walks (or two sets of oriented closed walks) to be that collection of oriented closed walks which results after canceling each pair of occurrences of an edge which are in opposite orientations. The traditional concept of *cycle space* in algebraic graph theory is defined over the finite field $\mathbb{F}_2$ [5]. In this sense, the sum cancels each pair of occurrences of an edge without consideration of their orientations. For example, let *abcda* and *abdca* be two closed walks in a graph. Then the oriented sum of the two is $\{abca, abda\}$ while the algebraic sum is *acbda*. Observe that the oriented sum is a commutative and associative operation.

The *oriented weight* W for a graph G, is a mapping from the oriented edges of G to K^n for some fixed n such that $W(xy) = -W(yx)$ for each edge xy. We extend this mapping to oriented walks as follows. Let $w = v_0 v_1 \ldots v_k$ be an oriented walk, then $W(w) = \sum_{i=0}^{k-1} W(v_i v_{i+1})$. Thus $W(w^r) = -W(w)$ and the oriented weight of every zero-walk is zero. An oriented weight W for a graph is said to be *non-singular* if $W(w) = 0$ for each oriented closed walk w in the graph.

Observation 1 *If w_1 and w_2 are oriented closed walks (or sets of walks) in a graph on which an oriented weight W is defined, then $W(w_1 + w_2) = W(w_1) + W(w_2)$.*

Proposition 2 *Let G be any graph with oriented weight W. Let w be any non-zero oriented closed walk in G, not necessarily simple, then there exists oriented cycles*

$w_1, \ldots, w_k$, possibly with multiplicity, such that $W(w) = W(w_1) + \cdots + W(w_k)$.

A trivial consequence of this result is that the oriented weight of any oriented walk can be expressed as the linear sum of the oriented weight of some oriented cycles with integer coefficients.

A subset of oriented cycles, $\mathcal{B}$, is called an *oriented basis* if the weight of every closed non-zero walk can be expressed as the sum of the oriented weights of some of the oriented cycles in $\mathcal{B}$, with integer coefficients.

Through out this paper we will only deal with oriented walks, oriented sum, oriented weight, and oriented basis. Therefore for simplicity we may often drop the adjective *oriented*.

3 Oriented Bases

In this section we describe two oriented bases. The first is applicable only to the edge-graphs of polytopes and the second is for general graphs.

Theorem 3 *Let G be the edge graph of a polytope. Then 2-face cycles of the polytope, each oriented in any one direction, form a basis of G.*

Next we show that a set of fundamental cycles of a graph also forms a basis. Let $G = (V, E)$ be a graph and $T \subseteq G$ be one of its spanning trees. Let $\overleftrightarrow{c_e}$ denote the unoriented cycle in the graph $T \cup \{e\}$ for some non-tree edge e of G. Then the set of fundamental cycles (w.r.t. T) is the collection $\{c_e | e \in E(G) \setminus E(T)\}$, where c_e is $\overleftrightarrow{c_e}$ oriented in any one direction. We assign a unique integer between 1 and $|E(G)|$ to each edge in G such that the integer assigned to any edge in $E(G) \setminus E(T)$ is greater than all the integers assigned to edges in $E(T)$. Let c be a cycle or a set of cycles of G. Then $le(c)$ denotes that edge in c which has the largest integer assignment.

Observation 2 *For every cycle c, $le(c) \in E(G) \setminus E(T)$.*

Theorem 4 *Fundamental cycles, each orieted in any one direction, form an oriented basis.*

Proof. In view of Proposition 2 it is sufficient to show that the weight of every set of cycles can be expressed as the sum of the weights of some fundamental cycles with integer coefficients. Assume that it is not true. So there is at least one set of oriented cycles whose weight cannot be expressed as the sum of weights of fundamental cycles. Let c be such a set such that label of $le(c)$ is smallest. Let $e = le(c)$. Then, by observation 2, $e \in E(G) \setminus E(T)$ where T is some fixed spanning tree. Let the fundamental cycle of e in G w.r.t. T be c_e, oriented in one of the two ways. Suppose e occurs in c for k_1 times in the same orientation as in c_e and for k_2 times in the opposite orientation. Define a new set of oriented cycles c' as $c + k_1.c_e^r + k_2.c_e$, where $k.x$ denotes the sum of k copies of x.

The new set c' of cycles has the property that the label of $le(c')$ is strictly less than the label assigned to e. From the assumption $W(c')$ can be expressed as the sum of the weights of fundamental cycles, say, $W(c') = W(c_1) + \cdots + W(c_m)$ where each c_i is an oriented fundamental cycle. Then $W(c) = W(c_1) + \cdots + W(c_m) + (k_1 - k_2).W(c_e)$. This contradicts the assumption that weight of c cannot be expressed as the sum of the weights of oriented fundamental cycles. □

4 Convex polytopes

In this section, we state a few basic facts about convex polytopes. The reader can find more details in [6].

A polytope is the convex-hull of a set of points in $\mathbb{R}^n$. In this paper a polytope refers only to the "shape" and the orientation of a polytope so its position in the space is ignored. Let $\mathcal{P}$ be a polytope in $\mathbb{R}^n$. Then $face_\omega(\mathcal{P})$ denotes the face of $\mathcal{P}$ with an outer normal ω, given by $\{x \in \mathcal{P} | \omega.x \geq \omega.y \ \forall y \in \mathcal{P}\}$. The set of all the outer normals of a face f of $\mathcal{P}$ is denoted by $N_{\mathcal{P}}(f)$ and is called the normal cone of the face f. The Minkowski sum of polytopes $\mathcal{Q}$ and $\mathcal{R}$ is the object given by $\mathcal{Q} + \mathcal{R} = \{x + y : x \in \mathcal{Q}, y \in \mathcal{R}\}$ which is also a polytope. The locations of $\mathcal{Q}$ and $\mathcal{R}$ only affect the location of $\mathcal{Q}+\mathcal{R}$, not its shape or orientation. Polytope $\mathcal{Q}$ is said to be a Minkowski summand of a polytope $\mathcal{P}$ if there is a polytope $\mathcal{R}$ such that $\mathcal{P} = \mathcal{Q} + \mathcal{R}$. Let $\mathcal{P}$ be a polytope in $\mathbb{R}^n$. Then $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ is called the edge-graph of $\mathcal{P}$ where $V_{\mathcal{P}}$ is the set of vertices (0-faces) of the polytope and $E_{\mathcal{P}}$ is the set of its edges (1-faces). We shall use the same symbol, to denote the position vector of a polytope vertex and the corresponding graph vertex.

Lemma 5 *For any direction ω, $face_\omega(\mathcal{Q} + \mathcal{R}) = face_\omega(\mathcal{Q}) + face_\omega(\mathcal{R})$.*

Lemma 6 *Let $\mathcal{P} = \mathcal{Q} + \mathcal{R}$. Let f_1 and f_2 be faces of $\mathcal{Q}$ and $\mathcal{R}$ respectively with $N_{\mathcal{Q}}(f_1) \cap N_{\mathcal{R}}(f_2) \neq \emptyset$, then $f_1 + f_2$ is a face of $\mathcal{P}$ with the normal cone being $N_{\mathcal{Q}}(f_1) \cap N_{\mathcal{R}}(f_2)$*

Lemma 7 *Let $\mathcal{P} = \mathcal{Q} + \mathcal{R}$ and $f \subset \mathcal{P}$ be a face. Then there exists unique faces $f_1 \subset \mathcal{Q}$ and $f_2 \subset \mathcal{R}$ such that $f = f_1 + f_2$.*

Lemma 8 *For every face f of a polytope in $\mathbb{R}^n$, $dim(f) + dim(N(f)) = n$, where $dim(\cdot)$ denotes the dimension.*

Lemma 9 *Let v be a vertex of a face $face_\omega(\mathcal{P})$ and u_0 be any other vertex of a polytope $\mathcal{P}$. Then there is a* monotonic path *in the edge graph $u_0, u_1, \ldots, u_j, \ldots, u_k (= v)$ such that $(u_{i+1} - u_i).\omega > 0$ for all $0 \leq i \leq j$ and $(u_{i+1} - u_i).\omega = 0$ for all $j \leq i < k$.*

4.1 Geometric Weight and Derived Weight

The oriented weight $W = \{w_{uv} = v - u\}_{uv \in E_{\mathcal{P}}}$ assigned to $G_{\mathcal{P}}$ is called the *geometric weight* of $G_{\mathcal{P}}$, where $v - u$ is the displacement vector from vertex u to vertex v in the space.

Observation 3 *The geometric weight of an edge graph of a polytope is non-singular.*

Consider a graph G with non-singular weight $W = \{w_{xy}\}_{xy \in E(G)}$. Then the weight $W_\alpha = \{\alpha_{xy}.w_{xy}\}_{e \in E(G)}$, where $0 \leq \alpha_{xy} = \alpha_{yx} \leq 1$ for all $xy \in E(G)$, is referred as *derived weight* of W if it is also non-singular. Further, the weight given by $\{(1 - \alpha_{xy}).w_{xy}\}_{xy \in E(G)}$ is denoted by $W_{1-\alpha}$. Since α's are independent of the orientation of the edge, we may express $\alpha_{xy} = \alpha_{yx}$ by α_e where e denotes the corresponding edge.

Observation 4 *Let W be a non-singular weight of some graph G. Then W_α is a derived weight iff $W_{1-\alpha}$ is also a derived weight.*

4.2 Polytope of embedding

Let G be a connected graph with a non-singular weight W where the vectors in the weight belong to $\mathbb{R}^n$. Let v_0 be a fixed vertex of G. We embed each vertex of G into $\mathbb{R}^n$ by a mapping $\phi_W : V(G) \to \mathbb{R}^n$ as follows. $\phi_W(v_0) = \vec{0}$; and for all $u \in V(G) - \{v_0\}$, $\phi_W(u) = W(P_u)$ where P_u is any arbitrary walk from v_0 to u in G. The mapping ϕ_W is well defined as W is non-singular. The convex-hull of the point set $\{\phi_W(u) : u \in V(G)\}$ defines a polytope denoted by $\phi_W(G)$. Vertices of this polytope are obviously from the set $\{\phi_W(u) : u \in V(G)\}$. We show that the converse is also true. It may be noted that the choice of v_0 is immaterial since it does not affect the shape or the orientation of the resulting polytope.

Let $G_{\mathcal{P}}$ be the edge graph of polytope $\mathcal{P}$ and W its geometric weight. Let W_α be a derived weight from W. Then the polytope $\phi_{W_\alpha}(G_{\mathcal{P}})$ is called a *derived polytope* of $\mathcal{P}$ and denoted by $\mathcal{P}_\alpha$. For simplicity we shall use ϕ_α in place of ϕ_{W_α}, where W should be clear from the context. We have the following important result.

Lemma 10 *For each vertex v of $\mathcal{P}$, $\phi_\alpha(v)$ is a vertex of $\mathcal{P}_\alpha$.*

Lemma 11 *Every derived polytope is a Minkowski summand of the original polytope.*

Proof Sketch If $\mathcal{P}_\alpha$ is a derived polytope of $\mathcal{P}$, then we show that $\mathcal{P} = \mathcal{P}_\alpha + \mathcal{P}_{1-\alpha}$. □

Next we will show the converse.

Lemma 12 *Let $\mathcal{Q}$ be a Minkowski summand of a polytope $\mathcal{P}$ then it is a derived polytope of $\mathcal{P}$.*

Proof Sketch Let $\mathcal{P} = \mathcal{Q} + \mathcal{R}$. If e is an edge of $\mathcal{P}$, then there exists unique edge-edge or edge-vertex pair $e' \in \mathcal{Q}$ and $e'' \in \mathcal{R}$ such that $e = e' + e''$. Let W be the geometric weight of $G_{\mathcal{P}}$ and W_α be its derived weight with $\alpha_e = |e'|/|e|$ for all $e \in E_{\mathcal{P}}$. Then $\mathcal{Q}$ is equal to the derived polytope $\mathcal{P}_\alpha$. □

Combining lemma 11 and 12 we have the main result.

Theorem 13 *For any polytope $\mathcal{P}$, a polytope $\mathcal{Q}$ is a Minkowski summand iff $\mathcal{Q}$ is some derived polytope of $\mathcal{P}$.*

The theorem can be equivalently stated as following.

Corollary 14 *A polytope has a proper Minkowski summand iff its edge graph has a proper derived weight (neither all α_e are 0 nor are all 1).*

Corollary 15 *For any lattice polytope $\mathcal{P}$, a lattice polytope $\mathcal{Q}$ is a Minkowski summand iff $\mathcal{Q}$ is a derived polytope $\mathcal{P}_\alpha$ such that all components of $\alpha_e.(\vec{v} - \vec{u})$ are integers for all edges $e = uv \in E_{\mathcal{P}}$.*

5 Computation of Minkowski summand

The Corollary 14 suggests that to discover a Minkowski summand of a polytope we only need to find if its edge graph has a derived weight. In this section we formulate a linear program (LP) which is feasible if and only if a derived weight exists.

Let $\mathcal{P}$ be a polytope. Each edge of the polytope $e = uv$, has the geometric weight $w_{uv} = \vec{v} - \vec{u}$ (equivalently $w_{vu} = \vec{u} - \vec{v}$). To compute a derived weight, we define a variable x_e for each edge e. The weight $\{w'_{uv} = x_e.(\vec{v} - \vec{u})$ would be a derived weight if and only if the weight of each basis cycle is zero (Theorem 3). The problem can be stated as a linear feasibility program.

Let $\mathcal{B}$ be a basis of $G_{\mathcal{P}}$. Let $c \in \mathcal{B}$ be denoted as $u_0, u_1, \ldots, u_m, u_{m+1}(= u_0)$, where u_j are the vertices on the cycle and let the edge $u_j u_{j+1}$ be denoted by e_j. Then the linear feasibility program (LP) is

$\sum_j x_{e_j}.(\vec{u_{j+1}} - \vec{u_j}) = \vec{0}, \quad \forall c \in \mathcal{B},$ **[P1]**
subject to
$0 \leq x_e \leq 1, \ \forall e \in E_{\mathcal{P}}; \ \sum_{e \in E_{\mathcal{P}}} x_e > 0;$
and $\sum_{uv \in E_{\mathcal{P}}} (1 - x_{uv}) > 0.$

The solution of the LP gives a derived weight of $G_{\mathcal{P}}$. The corresponding polytope, which is a summand of $\mathcal{P}$, can be computed using the embedding described in the previous section. A trivial solution of this LP is $x_e = c$ where c is a constant in the interval $(0, 1)$. This gives Minkowski summands, both of which are *similar* to the original polytope.

If $\mathcal{P}$ is a lattice polytope and the summand should also be a lattice polytope, then we need to satisfy an additional condition that $x_e(\vec{v} - \vec{u})$ has all integral components, i.e., $x_e.gcd(\vec{v} - \vec{u})$ must be an integer (recall

that $gcd(\vec{a})$ is the gcd of all the components of $\vec{a}$). This additional condition transforms the LP into the following linear integer feasibility program (IP) by defining integral variables y_e for $x_e.gcd(\vec{v}-\vec{u})$.

$\sum_j y_{e_j}.(\vec{u_{j+1}}-\vec{u_j})/gcd(\vec{u_{j+1}}-\vec{u_j}) = \vec{0}, \quad \forall c \in \mathcal{B}, \quad$ **[P2]**
subject to
$0 \leq y_{uv} \leq gcd(\vec{v}-\vec{u}), \ \forall e \in E_{\mathcal{P}}$;
$\sum_{e\in E_{\mathcal{P}}} y_e > 0$; and $\sum_{e\in E_{\mathcal{P}}}(gcd(\vec{v}-\vec{u})-y_e) > 0$, where y_e are integer variables.

The number of variables in the IP is equal to the number of the edges in the polytope, $|E_{\mathcal{P}}|$. The number of equations is n times the number of cycles in the basis, which is $|E_{\mathcal{P}}| - |V_{\mathcal{P}}| + 1$ in case $\mathcal{B}$ is the set of fundamental cycles.

6 Experimental Results

We have discussed earlier that Gao and Lauder [1] have shown that the Minkowski decomposition of convex lattice polytope is an NP-complete problem even in 2 dimensions. Therefore no exact method is expected to be polynomial in complexity. In this section we show that the proposed solution based on solving an integer linear program is a reasonably practical approach.

Given positive integers d and an n, we randomly generate n lattice points in $\mathbb{R}^d$. In the first step we compute the edge-graph of the convexhull of these points. In the second step we solve the integer program P2. The edges of the polytope are computed by solving a linear program for each pair of vertices, checking whether the line segment connecting them is a face or not. We use GLPK (GNU linear programming kit) to solve the LP's and the IP. The experiments were carried out on a 32-bit machine running on Intel Pentium 4 processor with 2 GB RAM and the code was written in the `C` programming language.

We ran ten instances of each case and reported the average time in the Tables 1 and 2. As the method is exact the success rate is always 100%. The times consumed in the two steps are reported separately to highlight the fact that the first step used up most of the time. This is because we could not find an efficient algorithm to compute the edges of a polytope. From Gao and Lauder's experiments [3] we see that their method is more reliable for higher dimensions (d) and smaller point-sets (n). In lower dimensions our method is competative with their method in terms of the time. Since our method is exact, we believe its complements their algorithm.

Table 1: Time(secs) to find the edges

Dimension, d	**Points,** n			
	10	50	100	200
2	0.13	0.38	0.49	0.54
5	0.46	9.24	30.39	106.76
10	0.49	16.93	89.17	590.94
20	0.50	18.88	113.93	851.37

Table 2: Time(secs) to decide indecomposability, i.e., time to solve IP

Dimension, d	**Points,** n			
	10	50	100	200
2	0.00	0.01	0.01	0.01
5	0.01	0.08	0.17	0.35
10	0.01	0.42	1.72	6.48
20	0.02	0.81	3.74	18.70

7 Conclusion

We have presented a criterion for Minkowski decomposition, general as well as integral. This reduces the problem of computing Minkowski summand into a linear (integer) program. We have reported experimental results. The performance of this approach can be improved significantly by using an efficient algorithm to compute the edges of the polytope. We believe this would give a performance comparable with the heuristic method proposed in [1].

References

[1] S. Gao and A. G. B. Lauder Decomposition of Polytopes and Polynomials. *Discrete and Computational Geometry* , 26:89–104, 2001.

[2] S. Gao Absolute irreducibility of polynomials via Newton polytopes. *J. of Algebra* , 231:501–520, 2001.

[3] S. Gao and A. G. B. Lauder Fast absolute irreducibility testing via Newton Polytopes. *preprint http://www.math.clemson.edu/faculty/Gao/papers/fastabs.pdf*, 14 pages, 2004.

[4] A. M. Ostrowski Uber die Bedeutung der Theorie der konvexen Polyeder fur die formale Algebra. *Jahresberichte Deutsche Math. Verein* , 30:98–99, 1921.

[5] Reinhard Diestel Graph Theory. *Graduate text in Mathematics, Springer*, 173, July 2005

[6] Bernd Sturmfels Gröbner Bases and Convex Polytopes. *University Lecture Series, American Mathematical Society*, 8, December 1995

[7] D. Mount and R. Silverman Combinatorial and computational aspects of Minkowski decomposition. *Contemporary Mathematics* , 199:107-124, 1991.

[8] I. Emiris and E. Tsigaridas Minkowski decomposition of convex lattice polygons. *Algebraic geometry and geometric modelling. Mathematics and Visualization, Springer* , 207–224, 2005.

[9] R. Silverman and A. Stein Algorithms for the decomposition of convex polygons. *Comtemporary Mathematics* , 119:159–168, 1991.

Convex Hull of the Union of Convex Objects in the Plane: an Adaptive Analysis

Jérémy Barbay* Eric Y. Chen†

Abstract

We prove a tight asymptotic bound of $\Theta(\delta \log(n/\delta))$ on the worst case computational complexity of the convex hull of the union of two convex objects of sizes summing to n requiring δ orientation tests to certify the answer. Our algorithm is deterministic, it uses portions of the convex hull of input objects to describe the final convex hull, and it takes advantage of easy instances, such as those where large parts of two objects are horizontally or vertically separated.

1 Introduction

An adaptive analysis of the computational complexity of a problem considers more parameters than the mere size n of the instance to be solved, such as the size of the result, or more sophisticated measures of the difficulty of the instance. A particular case of this approach has been applied to some fundamental problems in computational geometry, under the name of "output-sensitive" complexity analysis, where instances with a small output are considered easier, such as for the computation of the intersection points of a set of line segments [1, 2, 5, 6], or of the convex hull, discussed here.

The computational complexity of the convex hull has been studied in the worst case over instances of size n [11], over instances of size n and output size h [7], and over instances formed by polygonal chains with a parameterized number of self intersections [9]. Nielsen and Yvinec [10] studied the computation of the convex hull of the union of convex objects in the plane such that the convex hull of any pair of objects can be computed in constant time (such as discs or simple convex objects). Under those conditions, they proposed an algorithm which complexity is expressed as a function of the output size and of the maximum number of intersections of each object with others [10, page 4].

We also consider the computational worst case complexity of the convex hull of the union of convex objects in the plane, but in a different context than Nielson and Yvinec, where the convex hull of any pair of objects can be much more difficult to compute, sometime as much as to require linear time. Since the computation of the planar convex hull can be reduced to the computation of the lower and upper hulls, which can then be merged in constant time, we focus on the computation of the planar upper hull of the convex hull of the union of upper hulls. Let $I = \{A_1, A_2\}$ be an instance composed of two upper hulls in the Euclidean plane, of respective sizes n_1 and n_2 and each given as an array containing the coordinates of its points, ordered in clockwise order: we consider the problem of computing the upper hull of the minimal convex hull containing every point from the instance I, which we call the *merged hull* of I for short.

In this paper, we describe our algorithm, the adaptive analysis of its complexity and the matching adaptive computational lower bound, all in a model where only orientation tests (testing which side of a line is a point, in clockwise order) are allowed on the input. Our algorithm (Section 2) computes the merged hull taking advantage of the ordered representation of the upper hulls in sorted arrays. The analysis of this algorithm introduces the notion of *certificate* of an instance, from which we define our measure of difficulty over the instances. We highlight other application of our techniques in Section 4.

2 Convex Hull Problem

Before introducing our algorithm and its analysis, we introduce the concept of *certificate* of an instance, and some basic techniques on upper hulls.

2.1 Notion of Certificate

An *orientation test* is a constant time operation, which can determine which side a point lies to a line. Whereas an algorithm will perform many orientation tests to solve an instance, only a few are required to check that the result is correct. There is a similar situation in the comparison model: the binary search algorithm performs $\lceil \lg n \rceil + 1$ comparisons to search for the insertion rank of an element in a sorted array, whereas at most two comparisons are required to check the validity of the answer.

Given two upper hulls A and B, the merged hull is easier to compute for some instances than for some oth-

*DCC (Departamento de Ciencias de la Computación), Universidad de Chile, Santiago, Chile, jeremy.barbay@dcc.uchile.cl

†CSCS (Cheriton School of Computer Science) University of Waterloo, Canada. y28chen@uwaterloo.ca

ers. For example, Figure 1 show two instances, each composed of two hulls A and B at various distances from each other. The two instances have exactly the same input size n, and exactly the same output (the hull A), but they are much different in difficulty: one can *verify* that the edges of A exactly form the merged hull in two orientation steps in the first case, while n orientation steps are required in the second case.

We define the notion of *certificate* of an instance as a set of orientation tests which permit to check the validity of the result of the algorithm, not only justifying the presence of each point of the output, but also the exclusion of the other points:

Definition 1 *Consider k upper hulls $A_1, \ldots, A_k$ of respective sizes $n_1, \ldots, n_k$ and their merged hull A, expressed as intervals on $A_1, \ldots, A_k$. A* certificate *of A is a set of orientation tests "$A_i[p]$ is right to the line $\overrightarrow{A_j[q]A_j[k]}$", such that the convex hull of any instance satisfying those orientation tests is given by the description of A. The* size *δ of a certificate is the number of orientation tests composing it.*

Figure 1: The same output, different difficulty

2.2 Basic Operations

An easy type of instances (i.e. of small size of certificate δ) is that where a large section of one of the components of the instance can be "eliminated" by a simple orientation test, in the sense that the corresponding points will not contribute to the merged hull.

Observation 1 *Given a line $\overrightarrow{l}$ and an upper hull A, if the point $A[p]$ is right to $\overrightarrow{l}$ and the slope of $A[p]A[p+1]$ is smaller than the slope of $\overrightarrow{l}$, then all points right of $A[p]$ are right to $\overrightarrow{l}$; if the point $A[p]$ is right to $\overrightarrow{l}$ and the slope of $A[p-1]A[p]$ is greater than the slope of $\overrightarrow{l}$, then all points left of $A[p]$ are right to $\overrightarrow{l}$.*

Another characteristic which can make instances easier, is that it is not necessary to perform a binary search for the two edges of a hull intersecting a line on the whole upper hull at each search: a doubling search [3] algorithm permits to amortize the cost of each search over the whole hull:

Observation 2 *Given a line $\overrightarrow{l}$ and an upper hull A, the edge $(A[p], A[p+1])$ which intersects $\overrightarrow{l}$, if any, can be found in $\mathcal{O}(\log p)$ orientation tests.*

The same holds when searching for the tangent of a hull passing by a specific point: once again a doubling search [3] algorithm permits to amortize the costs of each search over the whole hull.

Observation 3 *The tangent $(x, A[p])$ of a point x with an upper hull A, if any, can be found in $\mathcal{O}(\log p)$ orientation tests.*

Many convex hull algorithms depend on computing the *common tangent* between two convex hulls in linear time. Using the previous observations, we show that if the upper hulls are horizontally separated, this common tangent can be found in time logarithmic not only in the size of the convex hulls [8], but even in time adaptive in the *positions* of the endpoints of the common tangent (i.e. closer to the left extremity is easier).

Lemma 2 *Given upper hulls A and B of respective sizes n and m, horizontally separated by a vertical line l, the common tangent $(A[p], B[q])$ from A to B can be computed in $\mathcal{O}(\log p + \log q)$ orientation steps.*

Proof. This idea is inspired by the prune-and-search method proposed by Kirkpatrick and Snoeyink [8], where the common tangent between two non-intersecting hulls can be computed in $O(\log n_1 + \log n_2)$ time.

Without loss of generality, suppose that A is to the left of l and that B is to its right. The basic operation performed by the algorithm considers the edge in clockwise order $(a, a') \in A$ and an edge $(b, b') \in B$:

1. If both a' and b' are left of the line $\overrightarrow{ab}$, then the points of B at the left of b (inside points of B) can be ignored.

2. Symmetrically, if both a' and b' are right the line $\overrightarrow{ab}$, then the points of A at the right of a (inside points of A) can be ignored.

3. If a' is right to $\overrightarrow{ab}$ and b' is left to it, then the points of A at the right of a and the points of B at the left of b (inside points) can be ignored.

4. If a' is left to $\overrightarrow{ab}$ and b' is right to it, which hull gets reduced depends of the slopes of (a, a') and (b, b'). If the line (a, a') cuts l above the line (b', b) then the points of A at the left of a can be ignored. Otherwise, the points of B at the right of b' can be ignored.

□

The result of Lemma 2 is still useful when only large parts of the upper hulls (and not the upper hulls themselves) are horizontally separated. Such parts which are separated from each other can be found using Observations 2 and 3.

3 Convex Hull of the Union of two objects

Beside the pedagogical interest to expose a simpler algorithm before the more complicated one, the algorithm for the union of two convex object is of independent interest:

- We prove that the certificate it finds is always optimal (which is not the case for the more general algorithm).
- It can be used as a building block for other union algorithms.

3.1 Adaptive Algorithm

Theorem 3 *The description of the merged hull of two upper hulls of respective sizes n_1 and n_2 can be computed in $\mathcal{O}(\delta(\log(n_1/\delta)+\log(n_2/\delta)))$ orientation tests, for an instance of certificate size δ.*

Proof. Let be A and B the two hulls forming the instance, of respective sizes n_1 and n_2. Without loss of generality, we prolongate each hull at each extremity by one vertical edge going to $-\infty$: hence all lines intersecting exactly once a hull are tangents to it, and other lines intersecting a hull once will intersect it a second time.

The algorithm 1 computes a description of the merged hull as a sequence of intervals over $1,\dots,n_1$ and $1,\dots,n_2$, representing consecutive points in A and B. To compute this description, the algorithm traverses the two hulls from left to right through doubling searches, searching for crossing points and discarding whole intervals of points in each hull, after certifying that they cannot contribute to the merged hull.

The invariant is quite simple: each iteration of the loop reduces the instance by discarding more points in each hull, and identifies the rightmost point yet certified to be in the merged hull, a, and its hull of origin A. In particular, all the points on the left of a in A which have not been output yet are certified to be part of the merged hull. Figures 2, 3 and 4 illustrate how the algorithm reduces the instance depending on the position of the intersection of A with the tangent from a to B:

- If $a = a'$, as all the points left of a in A are certified to be in the merged hull, and the points immediately on the right of a in A are right to $\overrightarrow{ab}$, the next point confirmed to be in the merged hull is b, hence reducing the instance by one point.

Algorithm 1 Convex Upper Hull of Two Objects

```
Identify the starting point a, and its hull A.
repeat
  Search in the other hull B for the tangent (a, b).
  Search in the hull A for its rightmost intersection
  a' with the line (a, b).
  if a = a' then
    Output and further ignore points of A left of a.
    Switch (a, A) to (b, B).
  else if a' is on the right of b then
    Further ignore points of B left of b.
    Update a to a' (not ignoring its predecessors).
  else
    Find the common tangent (c, d) between the
    points of A left of a' and the points of B right of
    b, separated by the vertical line passing by b.
    Output and further ignore points of A left of c.
    Switch (a, A) to (d, B).
  end if
until no point is left in any other hull than A.
Output all remaining points of A.
```

- If a' is on the right of b, as B is right to its tangent $\overrightarrow{ab}$, it is below the arc from A between a and a': the points of B between a and a' (or at least those between a and b, which have already been identified) can be further ignored, and all points on the left of a' (included) are certified to be part of the merging hull.
- If a' is on the left of b, A crosses and goes to the right of $\overrightarrow{ab}$ before potentially crossing B: some points in the right of b in B will contribute to the merged hull, and Lemma 2 indicates how to find them.

The algorithm outputs interval of points from the input hulls to describe the merged hull, performing exactly δ iterations because it computes the shortest certificate.

Each iteration corresponds to at most two doubling searches in each hull. As none of the δ doubling searches ever overlap, the number of orientation test in each hull of size n sums up to less than $\delta\log(n/\delta)$, by concavity of the log, hence the result. □

3.2 Optimality of Certificate

Lemma 4 *To determine the upper hull of two upper hulls, $\Omega(\delta)$ orientation tests are required.*

Proof. The certificate used in Algorithm 1 contains δ orientation tests. We prove that any certificate verifying the merged upper hull requires $\delta/4$ orientation tests.

Based on three testing cases in Algorithm 1, we categorize orientation tests into three types. In case 1 and

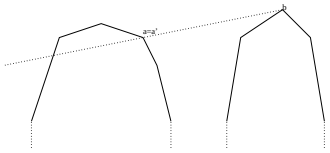


Figure 2: $a = a'$

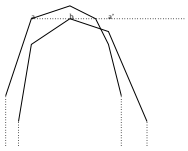


Figure 3: a' is on the right of b

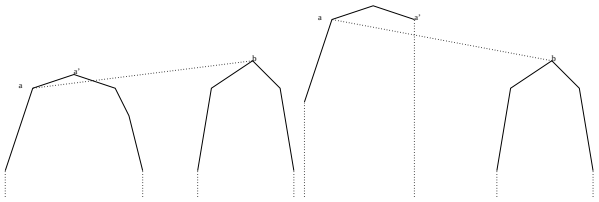


Figure 4: a' is on the left of b

3, the line in the orientation test is from a common tangent between A and B, and, in case 2, it is a tangent from a vertex in A to B. Grouping every 4 adjacent orientation tests together, we can have $\delta/4$ groups.

Then we prove the result by an adversary argument. Consider the vertical strip covered by 4 orientation tests in a group. The merged upper hull between A and B cannot be verified with only two orientation tests. We need at least one extra orientation tests in each group. Therefore, we need at least $\delta/4$ orientation tests. □

4 Conclusion

Convex hull instances with a very large set of points will not appear "out of nowhere": most likely, they will be formed of several objects from a library, for each of which a convex hull can be precomputed. In this context, we have given an algorithm to compute a description of the convex hull of the union of two convex objects which can be used recursively to merge k convex objects. Our algorithm takes advantage of instances where the relative positions of the objects makes the convex hull easier to compute.

As the basic operations are clearly identified in each algorithm, our results are easily generalizable to the transdichotomous computational model as well: each of the basic operation can be supported in time $\mathcal{O}(\log n/\log\log n)$ using a precomputed index [4], changing the complexity of the algorithm to $\mathcal{O}(\delta k \log n/\log\log n)$.

Acknowledgements: Many thanks to Alejandro López-Ortiz for discussing this direction of research, and to Timothy Chan and Alejandro Salinger for pointing to previous works.

References

[1] I. J. Balaban. An optimal algorithm for finding segments intersections. In *Proc. 11th Sympos. on Comput. Geom.*, pages 211–219, 1995.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[3] J. L. Bentley and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Information processing letters*, 5(3):82–87, 1976.

[4] T. M. Chan. Point location in o(log n) time, voronoi diagrams in o(n log n) time, and other transdichotomous results in computational geometry. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 333–342, 2006.

[5] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 1997.

[7] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 1986. 15(1):287–299.

[8] D. G. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *WADS '95: Proceedings of the 4th International Workshop on Algorithms and Data Structures*, pages 183–193, London, UK, 1995. Springer-Verlag.

[9] C. Levcopoulos, A. Lingas, and J. S. B. Mitchell. Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In *SWAT '02: Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, pages 80–89, 2002.

[10] F. Nielsen and M. Yvinec. Output-sensitive convex hull algorithms of planar convex objects. *International Journal of Computational Geometry and Applications*, 8(1):39–66, 1998.

[11] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

Polar Diagram of Moving Objects

Mojtaba Nouri Bygi* Mohammad Ghodsi†

Abstract

Many important problems in Computational Geometry needs to perform some kind of angle processing. The Polar Diagram [4] is a locus approach for problems processing angles. Using this structure as preprocessing, one can eliminate exhaustive searches to find objects with smallest angle.

Handling data in change is a significant concept in Computer Science. One of the design and analysis tools used in the modeling of moving geometric objects is the kinetic data structure (or KDS) framework Kinetic Data Structure is a framework for maintaining a certain attribute of a set of objects while moving in a continuous manner.

In this paper, we use the notion of kinetic data structure to model the dynamic case of the Polar Diagram, i.e we maintain the the Polar Diagram of a set of continuously moving objects in the scene. We show that our proposed structure meets the main criteria of a good KDS.

1 Introduction

Although most of the Geometric problems have optimal solutions, most of them are only optimum in the worst case. If the size of result is small or we have to answer many instances, these solutions may not be suitable for us. For these reasons, algorithms that preprocess the scene and then answer to each query with a better performance are widely used in this field.

C. I. Grima et al. [4, 5] introduced the concept of the Polar Diagram. The Polar Diagram of the scene consisting of n objects is a partition of plane to *polar regions*. Each object creates a polar region representing the locus of points with common angular characteristics in a starting direction. If point p lies in the polar region of object o, we know that o is the first object found after performing an angular scanning from the horizontal line crossing p in counterclockwise direction. The computation of the Polar Diagram can be done using the Divide and Conquer or the Incremental methods, both working in $\Theta(n \log n)$, which is optimum. By using this tessellation as preprocessing, we can avoid other angular sweeps by locating a point into a polar region in logarithmic time [4].

Kinetic Data Structure is a framework for maintaining a certain attribute of a set of objects while moving in a continuous manner. For example, KDS has been used for maintaining the convex hull of moving objects, or the closest distance among moving objects. A KDS is mainly consists of two parts: a description of the attribute with some certificates such that as long as these certificates do not change, the attribute does not change. It is assumed that we can compute the failure time of each of these certificates. In such events that a certificate fails, the KDS must be updated. Until the next event, the KDS remains valid. See the survey by Guibas [3] for more background on KDSs and their analysis.

In this paper, we first propose an improved algorithm for computing the Polar Diagram of a set of line-segments or polygons. Then we use the notion of kinetic data structure to model the dynamic case of the Polar Diagram, i.e we maintain the Polar Diagram of a set of continuously moving objects in the scene. We Show that our proposed structure meets the main criteria of a good KDS.

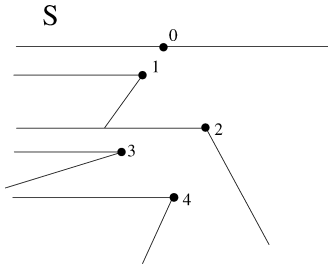


Figure 1: The Polar Diagram of a set of points in plane.

The rest of this paper is organized as follows: In section 2 we define our kinetic configuration for the Polar Diagram, and in section 2.2 we see what happens when the objects move in the plane.

*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

2 Kinetic Configuration

In this section we present a model for kinetic behavior of the Polar Diagram for different situations. Given a set of points moving continuously, we are interested in knowing at all times the Polar Diagram of the scene.

2.1 Proof Scheme

For simplicity of discussions, we assume that our objects are points in 2D. We state that each edge of the Polar Diagram is called a *polar edge*. We also define a *pivot* of an object to be the second object that lies on the polar edge passing through it, e.g., if Figure 1 the pivot of s_4 is s_2 and the pivot of s_2 is s_0.

We claim that if we have the sorted list of objects according to their y-coordinates, and the pivot of each object, we will have a unique Polar Diagram.

Suppose there are n points in the scene. For our proof scheme, we maintain two kinds of information about the scene: we maintain the vertically sorted list of objects, and for each object its current pivot. As we will show shortly, these data is sufficient for the uniqueness of our polar data, i.e. only if one of these conditions change, the polar structure of the scene will change.

So we will have two kinds of certificates: $n-1$ certificates will indicate the sorted list of objects. For instance, if the sorted list of objects is $s_{i_0}, s_{i_1}, \ldots, s_{i_{n-1}}$, we need the certificates 1.

$$\begin{aligned} s_{i_0} &< s_{i_1} \\ s_{i_1} &< s_{i_2} \\ &\ldots \\ s_{i_{n-2}} &< s_{i_{n-1}} \end{aligned} \tag{1}$$

For stating the pivot of each object, we need n more certificates, each indicating a object and its pivot in the Polar Diagram. In total, our proof scheme consists of $2n-1$ certificates.

2.2 Events and Event Handling

Once we have a proof system, we can animate it over time as follows. As stated before, each condition in the proof is called a certificate. A certificate fails if the corresponding function flips its sign. It is also called an event happens if a certificate fails. All the events are placed in a priority queue, sorted by the time they occur. When an event happens, we examine the proof and update it. An event may or may not change the structure. Those events that cause a change to the structure are called *exterior events* and those not *interior events*. When the motion of an object changes, we need to reevaluate the failure time of the certificates that involve that object (this is also called *rescheduling.*).

As there are two kinds of certificates in our proof scheme, it is obvious that there must be two kinds of event:

- **pivot event**, when three objects, which one of them is pivot of another one, become collinear.
- **horizontal event**, when two objects have a same y-coordinate (have a same horizontal level)

In the former case, we must update the certificates relating to sorted sequence of two neighbor points, which is at most three certificates (two, if one of the points is a boundary point, i.e. top most or button most points). In the latter case, one certificate becomes invalid and another certificate (indicating the new pivot of the object) is needed. As we will show, other certificates will remain still.

Lemma 1 *When an event is raised, the objects above the object(s) which raised the event do not change their polar structures.*

Proof: From the incremental method used for the construction of the Polar Diagram of a set of points [4] we know that there is no need to know about the state of objects below a object to determine its pivot object, so when an object change its state, it will not affect the above objects.

We can also say that an angular sweep that starts from the horizontal direction would never intersect any objects below this initial horizontal line (by definition, the top most object has no pivot). □

Pivot event:

First, we consider the simplest case, i.e. when the lowest object is moving. Figures 2 and 3 show these cases, where s_2 is moving. In Figure 2, s_0 is the pivot of s_2. While s_2 is moving left, the line segment s_0s_2 is coincide with the object s_1 (note that there may be other objects between s_0 and s_2, but we are only interested in s_1). At the moment that three objects s_0, s_1, and s_2 become collinear, the s_1 will occlude s_0 from s_2 and it no longer can be its pivot. From now on, s_1 becomes the new pivot of s_2. Similarly, in Figure 3, s_1 is the pivot of moving object s_2. When three objects s_0, s_1, and s_2 become collinear (again, there may be other objects between each pair of these objects, but we are not interested in them), s_2 needs to change its pivot which becomes s_0.

As we assumed that no other object other than s_2 is moving, form lemma 1 we know that there will be no change in other objects, so at this event, only one certificate becomes invalid and it must be replaced by another certificate indicating the new pivot of the moving object. It is clear that upon occurring this event, the processing of the event and changing of proof scheme can be done in $O(1)$ and $O(\log n)$, respectively (we need

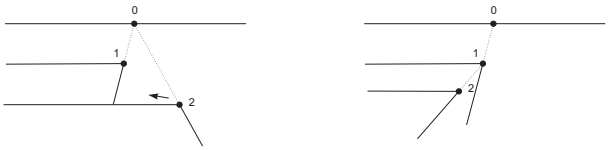


Figure 2: A pivot event. As s_2 moves left, s_0, s_1 and s_2 become collinear.

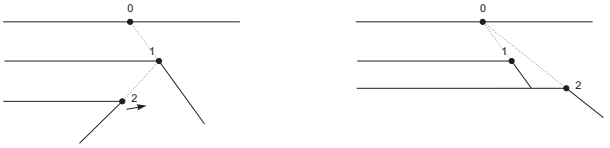


Figure 3: A pivot event. As s_2 moves right, s_0, s_1 and s_2 become collinear.

to find the corresponding certificate in the certificates list).

Now we see what happens to the second lowest object (see Figures 4 and 5, where s_2 is moving right). In Figure 4, s_1 is the pivot of s_2, and also the pivot of the lower object s_3. While moving, there will be a time that s_2 occlude the lower object s_3 from its pivot. In Figure 4 it is when the objects s_1, s_2 and s_3 become collinear. At this time, although there is no change in polar structure of moving object s_2, there is a change in the lower object s_3, and we must update the proof scheme accordingly. If s_2 continues its motion, there will be a pivot event (see Figure 5) that its polar structure is changing.

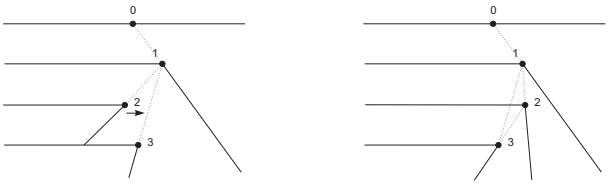


Figure 4: While moving, s_2 can change the pivot of each of its below objects by occluding their initial pivots.

Lemma 2 *The changes in the structure of an object caused by moving an above object, would not cause any other changes in other objects.*

Proof: The Structure of each object is determined by the first object that encountered by an angular sweep. As we assumed that no other objects is moved, this encountered object would not change. □

From above discussions, we can deduce that if an object is moving in the scene and there are k other objects below it, there can be up to k pivot events changing the structure of below objects, and one pivot event changing its own structure. Each of these events can be processed

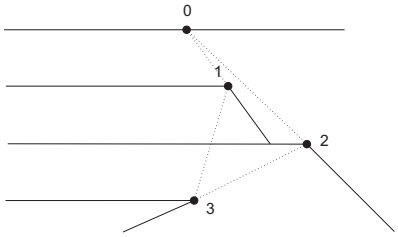


Figure 5: For each moving object, there is one pivot event when its own pivot will change.

in $O(1)$ time and the change in proof scheme can be done in $O(\log n)$.

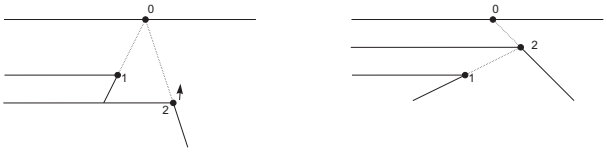


Figure 6: When two objects s_1 and s_2 lay on a same horizontal level, a horizontal event is occurred and the polar structure will change.

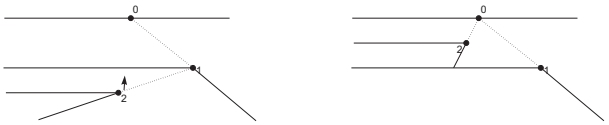


Figure 7: In a horizontal event, only one of the objects will change its pivot.

Horizontal event:

In these events, one of the situations of Figures 6 and 7 will happen. As we can see, only one of the objects will change its pivot (set it to the third object). This change of configuration is equal to changing three or four certificates in proof scheme: one for a change in one of the object's pivot, and three or two for change in vertical order of objects.

Now we show that no more changes is needed. Assume that in a small interval before and after the horizontal event, no other pivot events would occur. From lemma 1 we know that there would be no change in the above objects. What about the below objects? We can see that for a change in the pivot of an object, there must be an occlusion between the objects and its previous pivot, and it means that three objects must lay on a same line, i.e. we need a pivot event (see Figure 8).

Theorem 3 *Each of the events in the kinetic Polar Diagram of a set of points takes $O(\log n)$ time to process and causes has $O(1)$ changes in proof scheme.*

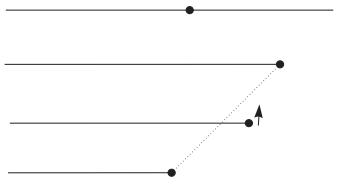


Figure 8: Only upon occurring a pivot event the structure of other objects will change.

Proof: For horizontal events, we need to update at most three certificates, we just need to find these certificates in the proof scheme and replace them with the new ones, which takes $O(\log n)$ time. We also need to update one pivot certificate with the same cost. The same thing is holds for pivot events, which we need to find and update $O(1)$ pivot certificates. □

Theorem 4 *The initial event list can be built in $O(n \log n)$ time, using a suitable event queue.*

Proof: As there are $O(n)$ certificates in our proof scheme, and for each moving object. we can find the first certificate that it will violates by a simple $O(\log n)$ search, the proof is straightforward. □

3 KDS Evaluation

In this section we evaluate our kinetic model according to the criteria of a *good* KDS. Similar to other algorithms, a good KDS should take small space, small initialization cost, and efficient update time. In KDS, an update may happen in two cases. One is when a certificate fails and an event happens. The other is when the motion of an object changes. In first case, we need to update the certificate set, and in the second case we must recompute the failure times for all the certificates that involve that object. These requirements induce the following quality measurements for KDSs [2].

Compactness the size of the proof.
Responsiveness the time to process an event.
Locality the number of certificates that a single object involves in.

Another crucial efficiency factor of a KDS is the number of events processed. This factor determines how many times we need to stop to check our proof and structure. This factor is expressed by efficiency:

Efficiency the number of events processed.

Now, we consider each of the above criteria in our kinetic model.

Compactness The structure clearly takes linear space. As we stated in Section 2.1, for a set of n point objects, the proof scheme consists of $n-1$ certificates for sorted vertical order of objects and n certificates for maintaining the pivots of each object, so in total, our proof scheme have $2n-1$ certificates.

Responsiveness $O(\log n)$ for processing an event as there are $O(1)$ certificates need to reschedule. Each reschedule takes $O(\log n)$ time.

Locality Each object is involved in at most three certificates.

Efficiency All the events are exterior – the ordering changes once a horizontal event happens, or the pivot of an object changes once a pivot event happens. The number of events is bounded by $O(n^2)$ as any two points can exchange their ordering only constant number of times for constant degree algebraic motions, and any point is a potential candidate for being the pivot of another point.

4 Conclusion and Future Work

In this paper we studied the concept of the Polar Diagram, which is a new locus approach for problems processing angles, and KDS, which is a structure that maintains a certain attribute of a set of continuously moving objects among moving objects. We used KDS to model the behavior of a the Polar Diagram when our scene is dynamic, i.e. we maintain the Polar Diagram of a set of continuously moving objects. We showed that our proposed structure meets the main criteria of a good KDS.

Following our defined model for the kinetic Polar Diagram, we can use it in direct applications of the Polar Diagram to maintain the computed attributes. For example, we can use the kinetic Polar Diagram for maintaining the convex hull of a set of moving objects with a very low cost.

References

[1] J. Basch, L. J. Guibas, and J. Hershberger. *Data structures for mobile data.* In Proc. 8th ACM-SIAM Sympos. Discrete Algorithms, pages 747–756, 1997.

[2] J. Basch. *Kinetic data structures.* PhD thesis, Stanford University, Palo Alto, California, 1999.

[3] L. Guibas. *Kinetic data structure: a state of art report.* In Proc. 3rd Workshop Algorithmic Found. Robot., pages 191–209, 1998.

[4] C.I. Grima, A. Márquez, and L. Ortega. *A new 2D tessellation for angle problems: The polar diagram.* In Computational Geometry, Theory and Application, 34(2): 58-74, 2006.

[5] C.I. Grima, A. Márquez, and L. Ortega. *A locus approach to angle problems in computational geometry.* In Proc. of 14th European Workshop in Computational Geometry, Barcelona, 1998.

Isometric Morphing of Triangular Meshes *

Prosenjit Bose† Joseph O'Rourke‡ Chang Shu§ Stefanie Wuhrer¶

Abstract

We present a novel approach to morph between two isometric poses of the same non-rigid object given as triangular meshes. We model the morphs as linear interpolations in a suitable shape space $\mathcal{S}$. For triangulated $3D$ polygons, we prove that interpolating linearly in this shape space corresponds to the most isometric morph in $\mathbb{R}^3$. We extend this shape space to arbitrary triangulations in $3D$ using a heuristic approach.

1 Introduction

Given two isometric poses of the same non-rigid object as triangular meshes $S^{(0)}$ and $S^{(1)}$ with known point-to-point correspondences, we aim to find a smooth isometric deformation between the poses. Interpolating smoothly between two given poses is called *morphing*. We achieve this by finding shortest paths in a shape space similar to the approach by Kilian et al. [5]. We propose a novel shape space.

A deformation of a shape represented by a triangular mesh is isometric if and only if all triangle edge lengths are preserved during the deformation [5]. We call a morph $S^{(t)}, 0 < t < 1$ between two (possibly nonisometric) shapes $S^{(0)}$ and $S^{(1)}$ *most isometric* if it minimizes the sum of the absolute values of the differences between the corresponding edge lengths of two consecutive shapes summed over all shapes $S^{(t)}$, for t in $[0, 1]$[1]. In this paper, we examine isometric morphs of general *triangular manifold meshes* in $3D$ and of *triangulated* $3D$ *polygons*, which are triangular meshes with no interior vertices. We introduce a new shape space $\mathcal{S}$ for triangulated $3D$ polygons that has the property that interpolating linearly in shape space corresponds to the most isometric morph in $\mathbb{R}^3$. We then extend this shape space to arbitrary triangulations in $3D$ using a heuristic approach. Note that self-intersections may occur when morphing.

Computing a smooth morph from one pose of a shape in two or three dimensions to another pose of the same shape has numerous applications. For example in computer graphics and computer animation this problem has received considerable attention [7, 1].

Recently, Kilian et al. [5] used shape space representations to guide morphs and other more general deformations between shapes represented as triangular meshes. Each shape is represented by a point in a high-dimensional shape space and deformations are modeled as geodesics in shape space. The geodesic paths in shape space are found using an energy-minimization approach. Before Kilian et al. [5] presented the use of a shape space for shape deformation and exploration of triangular meshes, shape space representations were developed to deform shapes in different representations. Cheng et al. [2] proposed an approach that deforms shapes given in skin representation, which is a union of spheres that are connected via blending patches of hyperboloids, with the help of a suitable shape space. Furthermore, algorithms for deforming curves with the help of shape space representations were proposed by Younes [10] and Klassen et al. [6]. Eckstein et al. [3] propose a generalized gradient descent method similar to the approach by Kilian et al. that can be applied to deform triangular meshes. All of these approaches depend on solving a highly non-linear optimization problem with many unknown variables using numerical solvers. It is therefore not guaranteed that the globally optimal solution is found.

2 Theory of Shape Space for Triangulated $3D$ Polygons

This section introduces a novel shape space for triangulated $3D$ polygons with the property that interpolating linearly in shape space corresponds to the most isometric morph in $\mathbb{R}^3$. The dimensionality of the shape space is linear in the number of vertices of the deformed polygon.

We start with two triangulated $3D$ polygons $P^{(0)}$ and $P^{(1)}$ corresponding to two almost isometric poses of the same non-rigid object. We assume that the point-to-point correspondence of the vertices $P^{(0)}$ and $P^{(1)}$ are known. Furthermore, we assume that both $P^{(0)}$ and $P^{(1)}$ share the same underlying mesh structure M. Hence, we know the mesh structure M with two sets of ordered vertex coordinates $V^{(0)}$ and $V^{(1)}$ in $\mathbb{R}^3$, where M is an outer-planar graph. We will show that we can represent $P^{(0)}$ and $P^{(1)}$ as points $p^{(0)}$ and $p^{(1)}$ in a shape space $\mathcal{S}$, such that each point $p^{(t)}$ that is a linear interpolation between $p^{(0)}$ and $p^{(1)}$ corresponds to a triangular mesh $P^{(t)}$ isometric to $P^{(0)}$ and $P^{(1)}$ in $\mathbb{R}^3$.

Let M consist of n vertices. As M is a triangulation of a

*Research supported in part by HPCVL and NSERC. We thank Martin Kilian for sharing his insight on the topic of shape spaces with us.

†Carleton University, Ottawa, Canada, *jit@scs.carleton.ca*.

‡Smith College, Northampton, USA, *orourke@cs.smith.edu*.

§National Research Council of Canada, Ottawa, Canada, *chang.shu@nrc-cnrc.gc.ca*.

¶Carleton University, Ottawa, Canada, and National Research Council of Canada, Ottawa, Canada, *swuhrer@scs.carleton.ca*.

[1]In a similar definition, Kilian et al. [5] use the L_2 instead of the L_1 metric to measure most isometric morphs.

$3D$ polygon with n vertices, M has $2n-3$ edges and $n-2$ triangles. We assign an arbitrary but fixed order on the vertices, edges, and faces of M. We eliminate rigid transformations by positioning $P^{(0)}$ and $P^{(1)}$ such that the first vertex v is incident to the origin, the first edge e of M incident to v is aligned along the positive x-axis, and the first triangle containing e lies on the x, y plane. The shape space $\mathcal{S}$ is defined as follows. The first $2n-3$ coordinates are the lengths of the edges in M in order. The final $2(n-2)$ coordinates are the outer normal directions of the triangles in M in spherical coordinates, in order. Hence, the shape space $\mathcal{S}$ has dimension $2n-3+2(n-2) = 4n-7 = \Theta(n)$.

In the following, we prove that interpolating linearly between $P^{(0)}$ and $P^{(1)}$ in shape space yields the most isometric morph. To interpolate linearly in shape space, we interpolate the edge lengths by a simple linear interpolation. That is, $p_k^{(t)} = tp_k^{(0)} + (1-t)p_k^{(1)}$, where $p_k^{(x)}$ is the kth coordinate of $p^{(x)}$. The normal vectors are interpolated using geometric spherical linear interpolation (SLERP) [8]. That is, $p_k^{(t)} = \frac{\sin(1-t)\Theta}{\sin\Theta} p_k^{(0)} + \frac{\sin t\Theta}{\sin\Theta} p_k^{(1)}$, where Θ is the angle between the two directions that are interpolated.

To study interpolation in shape space, we make use of the *dual graph* $D(M)$ of M. The dual graph $D(M)$ has a node for each triangle of M. We denote the dual node corresponding to face f of M by $D(f)$. Two nodes of $D(M)$ are joined by an arc if the two corresponding triangles in M share an edge. We denote the dual arc corresponding to an edge e of M by $D(e)$. Note that because M meshes a $3D$ polygon, it is an outer-planar triangular graph and so the dual graph of M is a binary tree. An example of a mesh M with its dual graph $D(M)$ is shown in Figure 1.

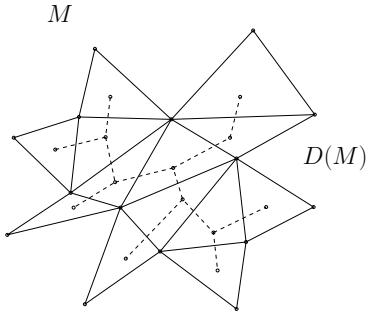


Figure 1: *A mesh M with its dual graph $D(M)$.*

Theorem 1 *Let M be the underlying mesh structure of the triangulated $3D$ polygons $P^{(0)}$ and $P^{(1)}$. The linear interpolation $p^{(t)}$ between $p^{(0)}$ and $p^{(1)}$ in shape space $\mathcal{S}$ for $0 \le t \le 1$ has the following properties:*

1. *The mesh $P^{(t)} \in \mathbb{R}^3$ that corresponds to $p^{(t)} \in \mathcal{S}$ is uniquely defined and has the underlying mesh structure M. We can compute this mesh using a traversal of the binary tree $D(M)$ in $\Theta(n)$ time.*
2. *If $P^{(0)}$ and $P^{(1)}$ are isometric, then $P^{(t)}$ is isometric to $P^{(0)}$ and $P^{(1)}$. If $P^{(0)}$ and $P^{(1)}$ are not isometric, then each edge length of $P^{(t)}$ linearly interpolates between the corresponding edge lengths of $P^{(0)}$ and $P^{(1)}$.*
3. *The coordinates of the vertices of $P^{(t)}$ are a continuous function of t.*

Due to page restrictions, the proof of this theorem is omitted in this abstract, but can be found in [9]. Note that Theorem 1 implies that the most isometric morph is found as all edge lengths are linearly interpolated.

Because $D(M)$ has complexity $\Theta(n)$, we can traverse $D(M)$ in $\Theta(n)$ time. Hence, we can compute intermediate deformation poses in $\Theta(n)$ time each.

Using the proposed algorithm, we deform the polygon shown in Figure 3 (a) to the polygon shown in Figure 3 (i). The morph is illustrated in Figures 3 (b)-(h). All of the intermediate poses are isometric to the start and end poses. The overlayed poses are shown in Figure 2.

Figure 2: *Most isometric morph of a simple polygon. The start polygon is a $3D$ polygon obtained by discretizing the curve $y = \sin(x)$ and by adding thickness to the curve along the z-direction. The end polygon is similarly obtained from $y = -\sin(x)$.*

3 Generalization to Triangular Meshes

This section extends the shape space from the previous section to arbitrary connected triangular meshes. We can no longer guarantee the properties of Theorem 1, because the dual graph of the triangular mesh M is no longer a tree.

Given two triangular meshes $S^{(0)}$ and $S^{(1)}$ corresponding to two almost isometric poses of the same non-rigid object with known point-to-point correspondence, we know one mesh structure M with two sets of ordered vertex coordinates $V^{(0)}$ and $V^{(1)}$ in $\mathbb{R}^3$. As before, we can represent $S^{(0)}$ and $S^{(1)}$ as points $s^{(0)}$ and $s^{(1)}$ in a shape space $\mathcal{S}$ using the same shape space points as in Section 2. Let $s^{(t)}$ be the linear interpolation of $s^{(0)}$ and $s^{(1)}$ in $\mathcal{S}$, where the linear interpolation is computed as outlined in Section 2. The existence of a mesh $S^{(t)} \in \mathbb{R}^3$ that has the underlying mesh structure M and that corresponds to $s^{(t)}$ is no longer guaranteed. This can be seen using the example shown in Figure 4. Figure 4(a) and (b) show two isometric meshes $S^{(0)}$ and $S^{(1)}$. The dual graph $D(M)$ of the mesh structure M is a simple cycle. Note that although the start and the end pose are isometric, we cannot find an intermediate pose that satisfies

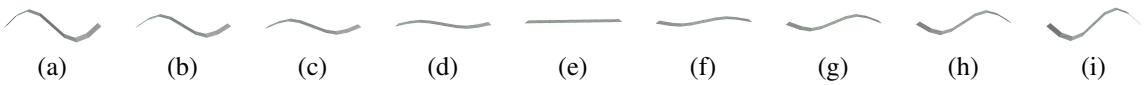


Figure 3: *Most isometric morph of a simple polygon from pose (a) to pose (i) obtained using the polygon algorithm.*

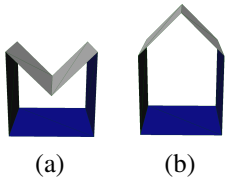


Figure 4: *Example of isometric triangular meshes where most isometric intermediate poses do not exist.*

all of the interpolated normal vectors and that is isometric to $S^{(0)}$ and $S^{(1)}$.

Let M consist of n vertices. As M is a planar graph, M has $\Theta(n)$ edges and $\Theta(n)$ triangles. The shape space $\mathcal{S}$ is defined using the same shape space points as in Section 2. The shape space $\mathcal{S}$ has dimension $\Theta(n)$. As before, we interpolate linearly in shape space by interpolating the edge lengths by a simple linear interpolation. The following observation is illustrated in Figure 4.

Observation 1 *Given a triangular mesh $S^{(t)}$ with underlying mesh structure M, point $s^{(t)}$ in $\mathcal{S}$ is uniquely determined. However, the inverse operation, that is computing a triangular mesh $S^{(t)}$ given a point $s^{(t)} \in \mathcal{S}$, is ill-defined.*

To compute a unique triangular mesh $S^{(t)}$ given a point $s^{(t)} \in \mathcal{S}$ that linearly interpolates between $s^{(0)}$ and $s^{(1)}$, such that $S^{(t)}$ represents the information given in $s^{(t)}$ well, we use the dual graph $D(M)$ of M. Unlike in Section 2, $D(M)$ is not necessarily a tree. Our algorithm therefore operates on a minimum spanning tree $T(M)$ of $D(M)$. The tree $T(M)$ is computed by assigning a weight to each arc e of $D(M)$. The weight of e is equal to the difference in dihedral angle of the supporting planes of the two triangles of M corresponding to the two endpoints of e. That is, we compute the dihedral angle between the two supporting planes of the two triangles of M corresponding to the two endpoints of e for the start pose $S^{(0)}$ and for the end pose $S^{(1)}$. The weight of e is then set as the difference between those two dihedral angles, which corresponds to the change in dihedral angle during the deformation. The weight can therefore be seen as a measure of rigidity. The smaller the weight, the smaller the change in dihedral angle between the two triangles during the deformation, and the more rigidly the two triangles move with respect to each other. As $T(M)$ is a minimum spanning tree, $T(M)$ contains the arcs corresponding to the most rigid components of M.

We compute $S^{(t)}$ by traversing $T(M)$. However, unlike in Section 2, setting the vertex coordinates of a vertex v of $S^{(t)}$ using two paths from the root of $T(M)$ to two triangles containing v can yield two different resulting coordinates for v. An example of this situation is given in Figure 5, where the coordinates of v can be set by starting at $root(T(M))$, and traversing the arcs e_2 and e_3 of $T(M)$ or by traversing the arcs e_1, e_4, and e_5 of M. We call the different coordinates computed for v in $T(M)$ *candidate coordinates* of v. Our algorithm computes the coordinates of each vertex $v \in S^{(t)}$ as the average of all the candidate coordinates of v.

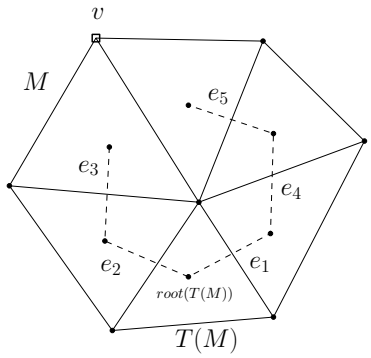
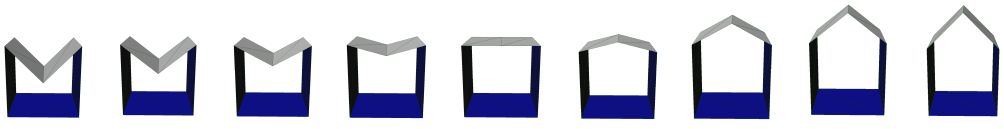


Figure 5: *A mesh M with its dual tree $T(M)$.*

To analyze the maximum number of candidate coordinates that can occur for a vertex in $S^{(t)}$, let e denote an edge of M such that $D(e)$ is in $T(M)$. Let v denote the vertex of $S^{(t)}$ opposite e in the triangle corresponding to an endpoint of $D(e)$, such that the coordinates of v are computed when traversing $D(e)$. This is illustrated in Figure 6. Let d_1 and d_2 denote the total number of candidate coordinates of the two endpoints of e. As we compute $d_1 d_2$ candidate coordinates for v by traversing $D(e)$, we can bound the number of candidate coordinates of v computed using the path through $D(e)$ by $d_1 d_2$. The number of candidate coordinates for the two endpoints of the edge corresponding to the first edge is one. Furthermore, each vertex v can be reached by at most $deg(v)$ paths in $T(M)$, where $deg(v)$ denotes the degree of vertex v in M. As each path in $T(M)$ has length at most $m-1$, where $m = O(n)$ is the number of triangles of M, we can bound the total number of candidate coordinates in $S^{(t)}$ by $\sum_{v \in V} 2^{m-1} deg(v) = 2n2^{m-1}$, where V is the vertex set.

Our algorithm finds a triangular mesh $S^{(t)}$ corresponding to $s^{(t)}$ that is isometric to $S^{(0)}$ and $S^{(1)}$ if such a mesh exists, because all of the candidate coordinates are equal in this case and taking their average yields the desired result. If there is no isometric mesh corresponding to $s^{(t)}$, our algorithm finds the *nearly isometric morph* as a unique mesh that weighs all the evidence given by $T(M)$ equally. By choosing $T(M)$ as a minimum spanning tree based on weights representing rigidity, we allocate rigid parts of the model more



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 7: *Nearly isometric morph of a cycle from pose (a) to pose (i) obtained using the exponential algorithm.*

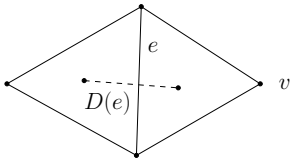


Figure 6: *Illustration of how to bound the number of candidate coordinates of v computed using the path through $D(e)$.*

emphasis than non-rigid parts. The reason for this is that in most morphs, triangles close to non-rigid joints are deformed more than triangles in mainly rigid parts of the model. We conclude with the following.

Proposition 2 *Let $S^{(0)}$ and $S^{(1)}$ denote two isometric connected triangular meshes and let $s^{(0)}$ and $s^{(1)}$ denote the corresponding shape space points, respectively. We can compute a unique triangular mesh $S^{(t)}$ representing the information given in the linear interpolation $s^{(t)}, 0 \leq t \leq 1$ of $s^{(0)}$ and $s^{(1)}$, in exponential time. We find a triangular mesh $S^{(t)}$ corresponding to $s^{(t)}$ that is isometric to $S^{(0)}$ and $S^{(1)}$ if such a mesh exists.*

The algorithm can easily be extended to work for a non-connected triangular mesh M by removing rigid transformations for each connected component of M using local coordinate systems. We can then adapt the algorithm by finding the dual graph $D(M)$ and a minimum spanning tree $T(M)$ for each connected component of M. With this information, we can traverse the graph as described above.

Using the proposed algorithm, we deform the model shown in Figure 7. We aim to smoothly and isometrically deform the pose shown in Figure 7(a) to the pose shown in Figure 7(i). As mentioned previously, there is no isometric deformation between the poses that interpolates the triangle normals. The result of our algorithm is shown in Figures 7(b)-(h). Note that all triangle normals are interpolated and the symmetry of the model is preserved. Furthermore, all edge lengths with the exception of the edges of the four top faces are interpolated.

4 Conclusion

We presented a novel approach to morph efficiently between isometric poses of triangular meshes in a novel shape space. The main advantage of this morphing method is that the most isometric morph is always found in linear time when triangulated $3D$ polygons are considered. For general triangular meshes, the approach cannot be proven to find the optimal solution. However, this paper presents a heuristic approach to find a morph for general triangular meshes. More efficient heuristics are explored in the accompanying report [9].

A direction for future work is to find an efficient way of morphing triangular meshes while guaranteeing that no self-intersections occur. For polygons in two dimensions, this problem was solved using an approach based on energy minimization [4].

References

[1] H. Alt and L. J. Guibas. *Discrete Geometric Shapes: Matching, Interpolation, and Approximation.* In: J. Sack and J. Urrutia (Editors). The handbook of computational geometry, pages 121–153. Elsevier Science, 2000.

[2] H.-L. Cheng, H. Edelsbrunner, and P. Fu. Shape space from deformation. PG, 1998.

[3] I. Eckstein, J.-P. Pons, Y. Tong, C. C. J. Kuo, and M. Desbrun. Generalized surface flows for mesh processing. SGP, 2007.

[4] H. N. Iben, J. F. O'Brien, and E. D. Demaine. Refolding planar polygons. SoCG, 2006.

[5] M. Kilian, N. J. Mitra, and H. Pottmann. Geometric modeling in shape space. *ACM TOG*, 26(3), 2007.

[6] E. Klassen, A. Srivastava, W. Mio, and S. Joshi. Analysis of planar shapes using geodesic paths on shape spaces. *IEEE TPAMI*, 26:372–383, 2004.

[7] F. Lazarus and A. Verroust. Three-dimensional metamorphosis: a survey. *The Visual Computer*, 14:373–389, 1998.

[8] K. Shoemake. Animating rotation with quaternion curves. *ACM TOG*, 19(3):245–254, 1985.

[9] S. Wuhrer, P. Bose, C. Shu, J. O'Rourke and A. Brunton. Morphing of Triangular Meshes in Shape Space. *Technical Report 0805.0162v2, arXiv*, 2008.

[10] L. Younes. Optimal matching between shapes via elastic deformations. *J. Image and Vision Computing*, 17(5,6):381–389, 1999.

Computing the Stretch Factor of Paths, Trees, and Cycles in Weighted Fixed Orientation Metrics

Christian Wulff-Nilsen*

Abstract

Let G be a graph embedded in the L_1-plane. The stretch factor of G is the maximum over all pairs of distinct vertices $\boldsymbol{p}$ and $\boldsymbol{q}$ of G of the ratio $L_1^G(\boldsymbol{p},\boldsymbol{q})/L_1(\boldsymbol{p},\boldsymbol{q})$, where $L_1^G(\boldsymbol{p},\boldsymbol{q})$ is the L_1-distance in G between $\boldsymbol{p}$ and $\boldsymbol{q}$. We show how to compute the stretch factor of an n-vertex path in $O(n\log^2 n)$ worst-case time and $O(n)$ space and we mention generalizations to trees and cycles, to general weighted fixed orientation metrics, and to higher dimensions.

1 Introduction

For $t \geq 1$, a t-spanner for a set of points is a network interconnecting the points such that the distance in the network between any pair of the given points is at most t times longer than the shortest possible distance between them. The smallest t for which the network is a t-spanner is called the stretch factor of the network. Computing networks with small stretch factors is an active area of research. For more on spanners, see e.g. [3, 2, 4].

An interesting dual problem is the following: given a network interconnecting a set of n points, what is the stretch factor of this network?

Fast algorithms for this problem are known only for simple graphs in the Euclidean plane. It has been shown that the stretch factor of a path in the Euclidean plane can be found in $O(n\log n)$ *expected* time and that the stretch factor of a tree and a cycle can be found in $O(n\log^2 n)$ *expected* time [1]. Using parametric search gives (rather complicated) $O(n\text{polylog } n)$ worst-case time algorithms for these types of networks.

In the plane, a *weighted fixed orientation metric* [5, 6] is specified by a fixed set $\mathcal{V}$ of vectors and the distance between a pair of points is defined as the length of a shortest path between them consisting of line segments all with weighted orientations from $\mathcal{V}$.

To our knowledge, the problem of efficiently computing the stretch factor of networks in weighted fixed orientation metrics has not received any attention. Since these metrics may be used to approximate other metrics and due to their applications in VLSI design, we believe this problem to be an important one.

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

In this paper, we give an $O(n\log^2 n)$ worst-case time algorithm for computing the stretch factor of an n-vertex path embedded in the L_1-plane and we mention generalizations to trees and cycles, to arbitrary weighted fixed orientation metrics, and to higher dimensions. Compared to the complicated worst-case time algorithms for the Euclidean metric, our algorithms are relatively simple and should be easy to implement.

The organization of the paper is as follows. In Section 2, we make basic definitions and introduce some notation. In Section 3, we consider the problem of computing the stretch factor of paths in the plane equipped with the L_1-metric and present an algorithm for this problem. In Section 4, we mention generalizations of our algorithm and we make some concluding remarks in Section 5.

2 Basic Definitions

For points $\boldsymbol{p} = (p_x, p_y)$ and $\boldsymbol{q} = (q_x, q_y)$, the L_1-*distance* $L_1(\boldsymbol{p},\boldsymbol{q})$ between $\boldsymbol{p}$ and $\boldsymbol{q}$ is $|q_x - p_x| + |q_y - p_y|$.

Let G be a connected graph embedded in the plane. To distinguish between a vertex of G and its location in the embedding, we write its location in boldface.

For two vertices u and v of G, define $L_1^G(\boldsymbol{u},\boldsymbol{v})$ as the length of a shortest path between u and v in G, where the length of a path is measured as the sum of L_1-lengths of the edges on the path in the embedding.

For distinct vertices u and v in G, the *detour* $\delta_G(\boldsymbol{u},\boldsymbol{v})$ between $\boldsymbol{u}$ and $\boldsymbol{v}$ in G is defined as $L_1^G(\boldsymbol{u},\boldsymbol{v})/L_1(\boldsymbol{u},\boldsymbol{v})$. The *stretch factor* δ_G of G is the maximum detour over all pairs of distinct vertices of G. If $\boldsymbol{u} = \boldsymbol{v}$ for two distinct vertices u and v of G, we define $\delta_G = \infty$.

For $\boldsymbol{p} \in \mathbb{R}^2$ and $r \geq 0$, we let $B_1(\boldsymbol{p}, r)$ denote the closed disc in $(\mathbb{R}^2, L_1)$ with center $\boldsymbol{p}$ and radius r.

3 Stretch Factor of Paths

In the following, let $P = p_1 \to p_2 \to \cdots \to p_n$ be an n-vertex path embedded in $(\mathbb{R}^2, L_1)$. In this section, we show how to compute the stretch factor of P in $O(n\log^2 n)$ time and $O(n)$ space.

We will make the simplifying assumption that all vertices of P have distinct locations. For otherwise, we would have $\delta_P = \infty$ and checking whether all vertices have distinct locations can be done in $O(n\log n)$ time.

In all the following, let $N = \{1, \ldots, n\}$. For $i \in N$ and $\delta > 0$, let $B_i(\delta)$ denote the disc $B_1(\boldsymbol{p_i}, r_i(\delta))$, where radius $r_i(\delta) = L_1^P(\boldsymbol{p_i}, \boldsymbol{p_n})/\delta$. The following lemma relates these discs to the stretch factor of P.

Lemma 1 *The stretch factor of P is $\delta_P = \inf\{\delta > 0 | B_j(\delta) \not\subseteq B_i(\delta)$ for all $i, j \in N, i \neq j\}$.*

Proof. Let $\delta > 0$. For any $i, j \in N$,

$$\frac{L_1^P(\boldsymbol{p_i}, \boldsymbol{p_j})}{\delta} = \frac{|L_1^P(\boldsymbol{p_i}, \boldsymbol{p_n}) - L_1^P(\boldsymbol{p_j}, \boldsymbol{p_n})|}{\delta} = |r_i(\delta) - r_j(\delta)|.$$

Hence,

$$\begin{aligned}\delta_P < \delta &\Leftrightarrow \forall i, j \in N, i \neq j : L_1(\boldsymbol{p_i}, \boldsymbol{p_j}) > |r_i(\delta) - r_j(\delta)| \\ &\Leftrightarrow \forall i, j \in N, i \neq j : L_1(\boldsymbol{p_i}, \boldsymbol{p_j}) > r_i(\delta) - r_j(\delta) \\ &\Leftrightarrow \forall i, j \in N, i \neq j : L_1(\boldsymbol{p_i}, \boldsymbol{p_j}) + r_j(\delta) > r_i(\delta) \\ &\Leftrightarrow \forall i, j \in N, i \neq j : B_j(\delta) \not\subseteq B_i(\delta).\end{aligned}$$

□

The idea of our algorithm is to see how much the size of the above defined discs can be increased before at least one of them includes another disc. By Lemma 1, this will then give us the stretch factor of path P.

For each $i \in N$ and for $w = 1, 2, 3, 4$, define $P_w(i)$ as the set of vertices of $P \setminus \{\boldsymbol{p_i}\}$ belonging to the wth quadrant of $\boldsymbol{p_i}$. Lemma 1 gives

$$\delta_P = \max_{w=1,2,3,4, i \in N} \inf\{\delta > 0 | B_j(\delta) \not\subseteq B_i(\delta) \forall \boldsymbol{p_j} \in P_w(i)\}.$$

Hence, δ_P is the maximum of four δ-values, one for each value of w. By symmetry, we may restrict our attention to computing

$$\max_{i \in N} \inf\{\delta > 0 | B_j(\delta) \not\subseteq B_i(\delta) \forall \boldsymbol{p_j} \in P_1(i)\}. \quad (1)$$

The following lemma gives a useful way of determining whether $B_j(\delta)$ is contained in $B_i(\delta)$ when $\boldsymbol{p_j}$ belongs to the first quadrant of $\boldsymbol{p_i}$.

Lemma 2 *Let $\boldsymbol{p_i}$ be a given vertex and let $\boldsymbol{p_j} \in P_1(i)$. For $\delta > 0$, define $\boldsymbol{r_i}(\delta)$ and $\boldsymbol{r_j}(\delta)$ as the rightmost points in $B_i(\delta)$ and $B_j(\delta)$, respectively. Then*

$$B_j(\delta) \subseteq B_i(\delta) \Leftrightarrow \boldsymbol{r_i}(\delta) \cdot (1, 1) \geq \boldsymbol{r_j}(\delta) \cdot (1, 1).$$

Proof. The point $\boldsymbol{r_j}(\delta)$ is to the right of $\boldsymbol{p_j}$ and belongs to the first quadrant of $\boldsymbol{p_i}$, implying that $L_1(\boldsymbol{p_i}, \boldsymbol{r_j}(\delta)) = L_1(\boldsymbol{p_i}, \boldsymbol{p_j}) + r_j(\delta)$. Since $B_j(\delta) \subseteq B_i(\delta)$ if and only if $L_1(\boldsymbol{p_i}, \boldsymbol{p_j}) + r_j(\delta) \leq r_i(\delta)$, we have

$$\begin{aligned}B_j(\delta) \subseteq B_i(\delta) &\Leftrightarrow L_1(\boldsymbol{p_i}, \boldsymbol{r_j}(\delta)) \leq r_i(\delta) \\ &\Leftrightarrow \boldsymbol{r_j}(\delta) \in B_i(\delta) \\ &\Leftrightarrow (\boldsymbol{r_j}(\delta) - \boldsymbol{r_i}(\delta)) \cdot (1, 1) \leq 0,\end{aligned}$$

since vector $(1, 1)$ is normal to the part of the boundary of $B_i(\delta)$ in the first quadrant of $\boldsymbol{p_i}$. □

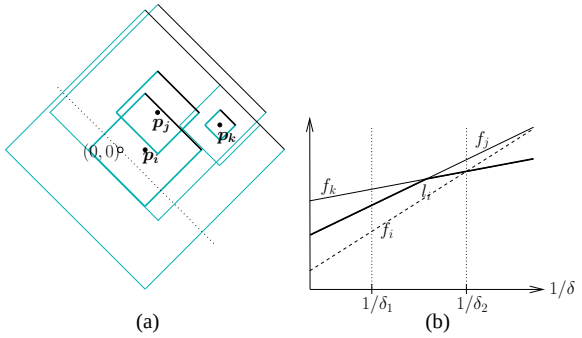


Figure 1: Illustration of Lemma 3. (a): L_1-discs $B_i(\delta)$, $B_j(\delta)$, and $B_k(\delta)$ for two values of δ: δ_1 (bold boundaries) and δ_2. (b): The corresponding functions f_i, f_j, and f_k. Lower envelope l_i is shown in bold. The distances in (a) between the dotted line and the black parts of L_1-discs correspond to values of functions f_i, f_j, and f_k at $1/\delta_1$ and $1/\delta_2$ in (b). Note that $B_k(\delta_2) \subseteq B_i(\delta_2)$. For all $1/\delta < 1/\delta_2$, $B_j(\delta) \not\subseteq B_i(\delta)$ and $B_k(\delta) \not\subseteq B_i(\delta)$.

Recall that, for any $i \in N$, $r_i(\delta) = L_1^P(\boldsymbol{p_i}, \boldsymbol{p_n})/\delta$. Hence, the dot product $\boldsymbol{r_i}(\delta) \cdot (1, 1)$ of Lemma 2 is an affine function of $1/\delta$, i.e. on the form $a(1/\delta) + b$, where a and b are constants. Denote this function by f_i.

Associate with each $\boldsymbol{p_i}$ a *lower envelope function* l_i of $1/\delta$, defined by

$$l_i(1/\delta) = \min\{f_j(1/\delta) | \boldsymbol{p_j} \in P_1(i)\}.$$

Our goal is to compute (1). The following lemma relates this value to the intersection between f_i and l_i.

Lemma 3 *There is at most one intersection point between f_i and l_i on interval $]0, \infty[$. If $1/\delta'$ is such a point then*

$$\delta' = \inf\{\delta > 0 | B_j(\delta) \not\subseteq B_i(\delta) \forall \boldsymbol{p_j} \in P_1(i)\}.$$

If there is no intersection point then for any $\delta > 0$ and any $\boldsymbol{p_j} \in P_1(i)$, $B_j(\delta) \not\subseteq B_i(\delta)$.

Proof. Figure 1 illustrates the lemma.

For any point $\boldsymbol{p}$ in the first quadrant of $\boldsymbol{p_i}$, the value $\boldsymbol{p} \cdot (1, 1)$ is minimized when $\boldsymbol{p} = \boldsymbol{p_i}$. It follows that $f_i(1/\delta) < l_i(1/\delta)$ for all sufficiently small $1/\delta$. Hence, since the graph of l_i on $]0, \infty[$ is a chain of line segments (and one halfline) whose slopes decrease as we move from left to right, there is at most one intersection point $1/\delta'$ between f_i and l_i on interval $]0, \infty[$.

If intersection point $1/\delta'$ exists, the above shows that, on interval $]0, \infty[$, $f_i(1/\delta) < l_i(1/\delta')$ if $1/\delta < 1/\delta'$ and $f_i(1/\delta) > l_i(1/\delta')$ if $1/\delta > 1/\delta'$. And if no intersection point exists then f_i is below l_i on interval $]0, \infty[$.

Lemma 2 shows that $B_j(\delta) \subseteq B_i(\delta)$ if and only if $f_i(1/\delta) \geq f_j(1/\delta)$. Hence, $B_j(\delta) \not\subseteq B_i(\delta)$ for all $\boldsymbol{p_j}$ in the first quadrant $P_1(i)$ of $\boldsymbol{p_i}$ if and only $f_i(1/\delta) < l_i(1/\delta)$. This shows the lemma. □

For each $i \in N$, let $\delta_i = 1/x_i$, where x_i is the intersection point between f_i and l_i. If no such point exists, set $\delta_i = 0$. Lemma 3 shows that (1) equals $\max_{i \in N} \delta_i$.

What remains therefore is the problem of computing the intersection (if any) between f_i and l_i for all i. In the following, we describe an algorithm for this problem.

3.1 The Algorithm

To simplify the description of the algorithm, we will leave out some of the details and return to them in Section 3.2, where we show how to obtain $O(n \log^2 n)$ running time and $O(n)$ space requirement.

The algorithm stores vertices of P in a balanced binary search tree $\mathcal{T}$ of height $\Theta(\log n)$ which is similar to a 1-dimensional range tree. Let V be the set of vertices of P. If V contains exactly one vertex, the root r of $\mathcal{T}$ is a leaf containing this vertex. Otherwise, r contains the median m of x-coordinates of vertices of V (in case of ties, order the vertices on the y-axis) and the subtree rooted at the left resp. right child of r is defined recursively for the set of vertices of V with x-coordinates less or equal to resp. greater than m.

Each node v of $\mathcal{T}$ corresponds to a subset S_v of vertices of P, namely those vertices stored at the leaves of the subtree of $\mathcal{T}$ rooted at v. We refer to these S_v-subsets as *canonical subsets*.

Note that each vertex $\boldsymbol{p_i}$ of P belongs to $\Theta(\log n)$ canonical subsets, namely those corresponding to vertices visited on the path from the root of $\mathcal{T}$ to the leaf containing $\boldsymbol{p_i}$.

In addition to a median, we associate with each node of $\mathcal{T}$ a lower envelope of line segments. This lower envelope is initially empty and will be updated during the course of the algorithm.

After having constructed $\mathcal{T}$, the algorithm makes a pass over the vertices of P in order of descending y-coordinate. In case of ties, vertices are visited from right to left.

The following invariant will be maintained throughout the course of the algorithm: *for each vertex v of $\mathcal{T}$, the lower envelope associated with v is the lower envelope of f_i-functions of vertices in S_v visited so far.*

When a vertex $\boldsymbol{p_i}$ of P is visited, the invariant is maintained by adding f_i to the $\Theta(\log n)$ lower envelopes associated with vertices on the path from the root of $\mathcal{T}$ to the leaf containing $\boldsymbol{p_i}$.

When the algorithm visits a vertex $\boldsymbol{p_i}$, it needs to find the intersection between f_i and l_i. However, explicitly computing l_i is too time-consuming.

Instead, we make use of our invariant which ensures that lower envelopes of visited vertices of all canonical subsets are given. The vertices in the first quadrant of $\boldsymbol{p_i}$ have all been visited and the set $P_1(i)$ of these vertices is therefore the union of visited vertices of the canonical subsets to the right of $\boldsymbol{p_i}$. So the intersection between f_i and l_i is the leftmost of the intersections between f_i and the lower envelopes associated with these canonical subsets.

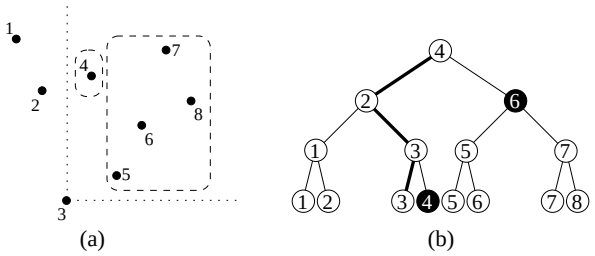


Figure 2: (a) Eight points shown with x-coordinates $1, \ldots, 8$. The set of points with x-coordinate greater than 3 is the union of two canonical subsets. (b) The two canonical subsets are found by picking right children (black) on the path from the root of the range tree to the leaf with median 3.

Since canonical subsets may overlap, not all canonical subsets to the right of $\boldsymbol{p_i}$ are needed. The idea is to pick a small number in order to minimize running time.

The algorithm picks canonical subsets (or more precisely, nodes of $\mathcal{T}$ corresponding to canonical subsets) as follows. Let v_i be the leaf of $\mathcal{T}$ associated with $\boldsymbol{p_i}$. For each vertex v on the path from the root r of $\mathcal{T}$ to v_i, the canonical subset associated with the right child of v is picked unless this child itself is on the path from r to v_i, see Figure 2.

It is easy to see that the visited vertices in the union of the picked canonical subsets are exactly the vertices of $P_1(i)$. Since the height of $\mathcal{T}$ is $\Theta(\log n)$, the number of picked canonical subsets is $O(\log n)$.

A fine point: if there are vertices of P above $\boldsymbol{p_i}$ and with the same x-coordinate as $\boldsymbol{p_i}$, they may not all belong to the picked canonical subsets even though they belong to $P_1(i)$. We may ignore these however, since they will be picked when second quadrants are handled.

Intersections between f_i and each of the lower envelopes of the picked canonical subsets are then computed and the leftmost of these is picked as the intersection between f_i and l_i.

From these intersections, the value (1) is obtained. This is repeated for the other three quadrants, giving the stretch factor of P.

3.2 Running Time and Space Requirement

In the description of the algorithm above, we left out some details. We now focus on them in order to analyze the running time of the algorithm.

It is easy to see that tree $\mathcal{T}$ can be constructed top-down in $O(n \log n)$ time. In the y-descending pass over the vertices of P, maintaining our invariant requires adding each f_i-function to $O(\log n)$ lower envelopes.

Finding these lower envelopes takes $O(\log n)$ time by a traversal from the root to a leaf of $\mathcal{T}$ using the medians at vertices to guide the search. It is easy to see that an f_i-function may be inserted in a lower envelope in $O(\log n)$ amortized time. Thus, the total time spent on maintaining the invariant is $O(n \log^2 n)$.

Next, we need to analyze the time it takes to compute the intersection between f_i and l_i for each i. This involves picking $O(\log n)$ canonical subsets and computing the intersection between f_i and the lower envelopes associated with these subsets.

Clearly, the time it takes to find the canonical subsets is bounded by the height of $\mathcal{T}$ which is $\Theta(\log n)$. Since each lower envelope l is a monotonically increasing function and its graph consists of line segments (and one halfline), computing the intersection between f_i and l can be done in $O(\log n)$ time by using an appropriate data structure. It follows that our algorithm has $O(n \log^2 n)$ running time.

The algorithm described above has $O(n \log n)$ space requirement. To improve space requirement to linear, we modify the algorithm so that, instead of making only one y-descending pass over the vertices of P, it makes $h(\mathcal{T})$ passes (for each of the four quadrants), where $h(\mathcal{T})$ is the height of $\mathcal{T}$. In the kth pass, only lower envelopes at level k-nodes of $\mathcal{T}$ are updated; all other nodes of $\mathcal{T}$ contain empty lower envelopes. And only intersections between f_i-functions and lower envelopes at level k of $\mathcal{T}$ are computed.

This gives the first main result of the paper.

Theorem 4 *The stretch factor of an n-vertex path in $(\mathbb{R}^2, L_1)$ can be computed in $O(n \log^2 n)$ time and $O(n)$ space.*

4 Generalizations

In this section, we present generalizations of our algorithm. Proofs have been omitted due to space constraints.

By using ideas similar to those in [1], we obtain generalizations to trees and cycles. And by using higher dimensional range trees in Section 3.1, it is relatively easy to generalize our algorithm to $(\mathbb{R}^d, L_1)$. This gives the following result.

Theorem 5 *Let G be an n-vertex graph embedded in $(\mathbb{R}^d, L_1)$, $d \geq 2$. The stretch factor of G can be computed in $O(n \log^d n)$ time when G is a path and in $O(n \log^{d+1} n)$ time when G is a tree or cycle. Space requirement is $O(n)$.*

In two dimensions, we have generalized our algorithm to weighted fixed orientation metrics. Basically, we linearly map cones in such metrics to quadrants and then apply our algorithm for the L_1-metric.

Theorem 6 *Let G be an n-vertex graph embedded in the plane equipped with a weighted fixed orientation metric defined by $\lambda \geq 2$ weighted orientations. The stretch factor of G can be computed in $O(\lambda n \log^2 n)$ time when G is a path and in $O(\lambda n \log^3 n)$ time when G is a tree or cycle. Space requirement is $O(n)$.*

Suppose that we are not only interested in computing the stretch factor but also in finding a pair of vertices for which the detour between them equals the stretch factor of the graph. The following theorem shows that this can be done without affecting time and space bounds.

Theorem 7 *Within the same time and space bounds, all algorithms described above can be modified to compute, for every vertex $\boldsymbol{p_i}$, a vertex $\boldsymbol{p_j}$ maximizing the detour between $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ in the graph. In particular, a vertex pair achieving the stretch factor of the graph can be found within these time and space bounds.*

5 Concluding Remarks

Given an n-vertex path in $(\mathbb{R}^2, L_1)$, we showed how to compute its stretch factor in $O(n \log^2 n)$ worst-case time and $O(n)$ space. We mentioned generalizations to cycles and trees, to higher dimensions and to weighted fixed orientation metrics in the plane.

An obvious question is whether our algorithms are optimal with respect to running time. In the Euclidean plane, an $\Omega(n \log n)$ lower bound is known for paths and trees and it is easily seen to hold for the L_1-metric. Thus, there is a gap of $\log n$ for paths and $\log^2 n$ for trees between our time bounds and this lower bound. Is it possible to handle more general types of graphs?

References

[1] P. K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, M. Sharir, and M. Soss. Computing the Detour and Spanning Ratio of Paths, Trees and Cycles in 2D and 3D. Discrete and Computational Geometry, 39 (1): 17–37 (2008).

[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, Amsterdam, 2000.

[3] G. Narasimhan and M. Smid. Geometric Spanner Networks. *Cambridge University Press*, 2007.

[4] 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, Amsterdam, 2000.

[5] K. J. Swanepoel. The Local Steiner Problem in Normed Planes. *Networks*, 36(2):104–113, 2000.

[6] P. Widmayer, Y. F. Wu, and C. K. Wong. On Some Distance Problems in Fixed Orientations. *SIAM Journal on Computing*, 16(4): 728–746, 1987.

The Focus of Attention Problem Revisited

Manjish Pal*

Abstract

The Focus of Attention (FOA) Problem is, given a set of targets and a set of sensors in the plane, to track the targets with 'maximum possible accuracy'. The accuracy is measured in terms of the angle subtended by the sensor pairs at the assigned targets. In this paper, we consider a scenario in which we are given n targets on a line l and $2n$ sensors on a line m such that $l \mid\mid m$ and the objective is to assign sensor-pairs to the targets such that the minimum angle subtended at the targets is maximized. We give a polynomial time algorithm for a restricted version of this problem and also study some properties of the optimal solution. To our knowledge, no deterministic and exact polynomial time algorithm is known for any non-trivial version of the FOA problem when the accuracy is measured in terms of the angles subtended by the sensor pairs.

Keywords: Target tracking, Focus of Attention problem

1 Introduction

The Focus of Attention problem is motivated from the problem of tracking targets using sensor networks which is mainly used for the purpose of surveillance and monitoring tasks [3]. A limitation these sensors have is that one sensor is not capable of tracking a target. The sensors for example can be cameras and they can be used to estimate the position of a target. In practice, at least two such sensors are required to estimate the position of a target and for many cases like pan-tilt-zoom cameras one sensor cannot be used to track more than one target. For range sensors, three are required to localize a target [1]. In this paper, we will only consider the case in which each target can be tracked exactly by 2 cameras. To estimate the quality of tracking a good metric is required. The angle subtended by a camera pair at a target plays a crucial role in tracking the targets [4]. To have minimum uncertainty in the position of the targets the best measure is to assign the cameras to the targets such that the deviation of the subtented angle at each of the targets from 90 degrees is somewhat low. In other words if all the subtended angles are 90 degrees then the position of the targets is estimated very accurately. It follows from this that for a good tracking of a target, the angle subtended at the target should not be very small. A natural way to make sure that the no angle is very small will be to maximize the minimum angle subtended at the targets. In [1] the cameras are assumed to be on a line and the error associated with an assignment of cameras c_i and c_j with target k is Z_k/l_{ij} where l_{ij} is the distance between the cameras and Z_k is the distance of the cameras from the line containing the cameras. The objective is to find an assignment that minimizes the total error. Intuitively, if we fix Z_k and the angle is small then the value of l_{ij} will be small and error will be large. Hence this metric tries to capture the angles via an approximation. So, the natural question that arises is, can we work directly with angles and design efficient algorithms that can output an assignment that minimizes the maximum deviation from 90 degrees? Unfortunately, this problem has been shown to be intractable in [2]. Gfeller et al [2] have recently shown that given a set of $2n$ cameras and n targets in the plane, it is NP-complete to decide whether there exists an assignment of cameras to track targets such that each subtended angle is 90 degrees. They also give approximation algorithms for maximizing the minimum angle and maximizing the sum of angles when cameras are placed on a line. An extensive survey and motivation on this problem is provided in [2].

To our knowledge, no deterministic and exact polynomial time algorithm is known for any non-trivial version of the FOA problem when the accuracy is measured in terms of the angles subtended by the camera pairs. In this paper we achieve such a result for a restricted version of the problem.

2 Problem Definition

Consider n targets present on a line l_1 and $2n$ cameras on a line l_2, where l_1 is parallel to l_2. Two cameras c_i and c_j are said to subtend an angle θ at a target t_k, if $\angle c_i t_k c_j$ is θ. In an **assignment**, a camera c can focus on exactly one target and a target is focussed by exactly two cameras. Moreover, each camera has to be used in an assignment. Consider the following version of the

*Indian Institute of Technology Kanpur, Kanpur-208016, India, manjish@iitk.ac.in. This work was done during the CADMO internship programme-2007, at ETH-Zürich, Switzerland.

focus of attention problem:
We have to find an assignment of cameras to targets such that

- The minimum angle of all possible assignments is maximized (Parallel-MAX-MIN Problem).

Let LHS denote the set of first n cameras and RHS denote the set of last n cameras (ordered from left to right).

3 Parallel-MAX-MIN Problem

We start off by proving a fundamental fact regarding this optimization problem.

Lemma 1 *There always exists an optimal solution $\mathcal{O}$ of the Parallel-MAX-MIN problem such that in $\mathcal{O}$ there are no two camera pairings c_i, c_j and c_k, c_l such that $i < j < k < l$ where i, j, k, l denote the index of the cameras in the sorted order from left.*

Proof Let camera pairs (c_i, c_j) and (k, l) be assigned to t_p and t_q repectively in an optimal solution where $i < j < k < l$. If t_p is to the left of t_q then we can swap the assignment to get a new assignment with pairings (c_i, c_k, t_p) and (c_j, c_l, t_q) and in this new assignment the subtended angles for both the assignments increase, hence the minimum angle in the solution does not decrease. Similarly, if t_p is to the right of t_q then the pairings (c_i, c_k, t_q) and (c_j, c_l, t_p) will give us a solution which is optimal as well.
The following can be easily derived from the previous lemma.

Corollary 1 *There exists an optimal solution in which each camera from the set of first n cameras from left is paired with a camera in the set of last n cameras.*

Consider the following decision version of the problem:

Given an angle θ does there exist an assignment of cameras and targets such that all the subtended angles in the assignment are at least θ

It is clear that if we can solve this problem then we can solve the MAX-MIN problem just by doing a binary search on the n^3 possible values of θ. If we choose a camera pair, c_i from LHS and c_j from RHS then there exists a **validity interval** I such that for every point p in the interval the angle subtented by c_i and c_j at p is at least θ. This interval is defined by the intersection of a ball B with the line containing the cameras such that the boundary of this ball is that circle passing through i and j, for which the angle subtented by the chord ij at its center is 2θ, if θ is acute and $2\pi - 2\theta$, if θ is obtuse. So, the decision problem is equivalent to decide whether there exists an assignment of cameras and targets such that if target t_k is assigned to camera pair (i, j) then it lies inside the validity interval of (i, j).

3.1 Some Properties of the Intervals

We first introduce some notations on which the rest of the paper is based. We label the first n cameras from left to right as $a_1, a_2, \ldots a_n$, the last n cameras from left to right as $b_n, b_{n-1}, \ldots b_1$ and the n targets from left to right as $t_1, t_2, \ldots t_n$. We denote the validity interval corresponding to camera pairs (a_i, b_j) for angle θ by $I_\theta(a_i, b_j)$. The words 'before' and 'after' will corresponding to ordering from left to right. We can now state the following lemmas which are easy to prove:

Lemma 2 *If $I_\theta(a_i, b_j)$ and $I_\theta(a_k, b_l)$ are two intervals with $i \leq k$ and $j \leq l$, then $I_\theta(a_i, b_j)$ is nested within $I_\theta(a_k, b_l)$.*

Corollary 2 *If $I_\theta(a_i, b_j)$ covers a target t then it is also covered by $I_\theta(a_k, b_l)$ for all $k \leq i$ and $l \leq j$.*

3.2 Polynomial time Algorithm for a Restricted Case

In this section we describe the conditions that if imposed on the problem instance can bring some nice structure to the intervals defined by the cameras which can be exploited to get a greedy strategy work for the problem. We call this constraint **Interval Property** which can be stated in the following way:

For every camera $a_i \in LHS$, $I_\theta(a_i, b_n)$ should start before the interval $I_\theta(a_{i+1}, b_1)$ and for every camera $b_j \in RHS$, the interval $I_\theta(a_n, b_j)$ should end after the interval $I_\theta(a_1, b_{j+1})$.

Given an instance of the Parallel-MAX-MIN Problem, let Θ be the set of all angles θ such that there exist $a_i \in LHS, b_j \in RHS$ and target t_p with $\angle a_i t_p b_j = \theta$. We impose the following constraint C in order to make our algorithm work:
C: For each $\theta \in \Theta$ the following should hold
1. All the targets are to the left of the right end-point of the interval $I_\theta(a_n, b_n)$.
2. Interval Property is satisfied for each angle $\theta \in \Theta$.
Next, we derive a geometric constraint under which Interval Property is satisfied for a given value of θ.

3.2.1 The Height Condition

Consider the defining circles C_i and C_i' corresponding to the intervals $I_\theta(a_i, b_n)$ and $I_\theta(a_{i+1}, b_1)$. Let these circles intersect at 2 points, the one which is above the line of the cameras let it be called X. Let the distance of this point from the line containing the cameras be denoted by H_{a_i}. In order to impose the above restriction on the intervals we would like to have the distance between the parallel lines to be less than H_{a_i}. So, we can calculate the critical height for each a_i and b_j and

the height h_c we choose should satisfy the condition $h_c < \min\{\min_{a_i \in LHS} H_{a_i}, \min_{b_j \in RHS} H_{b_j}\}$. The following calculations are done assuming $\theta < 90°$. These can be analogously done for obtuse and right angle as well.

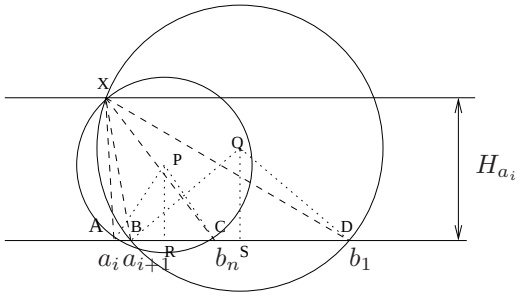


Figure 1: The Height Condition

In figure 1, $\angle AXC = \angle BXD = \theta$. Let the distance between a_i and b_n is $AC = d_1$, the distance between a_{i+1} and b_1 is $BD = d_3$ and $BC = d_2$. We use coordinate geometry to find the value of H_{a_i}. Let the center of C_i be P and that of $C_i^{'}$ be Q. It is clear that the radius of C_i is $r_1 = (d_1 + d_2)\text{cosec}\theta/2$ and that of $C_i^{'}$ is $r_2 = (d_2 + d_3)\text{cosec}\theta/2$. If the relative positions of the points A, B, C, R, S is as shown in the Figure 1 then the distance between the centres of the two circles is $RS = BC - SC - BR$. Now $SC = BC - BS = d_2 - (d_2 + d_3)/2 = (d_2 - d_3)/2$ and $BR = d_2 - (d_1 + d_2)/2 = (d_2 - d_1)/2$. Therefore $RS = (d_1 + d_3)/2$. The equation of C_i is $x^2 + y^2 = r_1^2$ (taking origin of coordinates at P) and that of $C_i^{'}$ is $(x - (d_1 + d_3)/2)^2 + (y - (d_3 - d_1)\text{cot}\theta/2)^2 = r_2^2$. If we solve for the intersection of these circles we get

$$x^2 + y^2 - r_1^2 = (x - \frac{(d_1 + d_3)}{2})^2 + (y - \frac{(d_3 - d_1)\text{cot}\theta}{2})^2 - r_2^2$$

$$r_1^2 - r_2^2 = \frac{(d_1 + d_2)^2}{4} + x(d_1 + d_3) + \frac{(d_3 - d_1)^2\text{cot}^2\theta}{4} + y(d_3 - d_1)\text{cot}\theta$$

Let,

$$A_i(\theta) = \frac{(r_1^2 - r_2^2 - (d_1 + d_3)^2/4 - (d_3 - d_1)^2\text{cot}^2\theta/4)}{(d_1 + d_3)}$$

$$B_i(\theta) = \frac{(d_3 - d_1)\text{cot}\theta}{(d_3 + d_1)}$$

Then we have $x = A_i(\theta) + B_i(\theta)y$. Putting this value in the equation $x^2 + y^2 = r_1^2$ we get a quadratic equation in y whose positive root we denote by y_{pos}. Therefore, $H_{a_i} = y_{pos} + (d_1 + d_2)\text{cot}\theta/2$. So, in order to force the aforementioned structure on the intervals we need to select $h \leq h_c$.

Before going for the algorithm we state the following lemmas which are quite easy to see and hence we omit their proofs.

Lemma 3 *Given that interval property is satisfied, if interval $I_\theta(a_i, b_j)$ is nested by $I_\theta(a_k, b_l)$ then $k \leq i$ and $l \leq j$.*

Lemma 4 *Given that interval property is satified, if the left end-point of $I_\theta(a_i, b_j)$ is to the left of left end-point of $I_\theta(a_k, b_l)$ and right end-point of $I_\theta(a_i, b_j)$ is to the left of right end-point of $I_\theta(a_k, b_l)$ then $i < k$ and $l < j$.*

The arrangment of the intervals satisfying interval property ($n = 3$) for a particular instance is shown in Figure 2

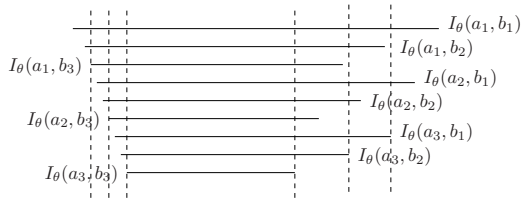


Figure 2: Arrangement of Intervals satisfying Interval Property

3.3 Algorithm

Now the question arises, how can this structure help us in getting a greedy strategy work. We assume a restriction on the where the cameras are placed which is as follows. We suggest a greedy strategy that works in this case. The strategy is as follows:

Consider the targets from left to right. For the i^{th} target from the left, let $I_1, I_2, \ldots I_m$ be the intervals in which it lies, assign it to the interval among these which is ending earliest. If the interval is $I_\theta(a_i, b_j)$ then remove all the intervals with left camera as a_i and right camera as b_j.

It can easily be seen that this algorithm can be implemented in $O(n^3 \log n)$ time.

Proof of Correctness of Algorithm

Proof. Let the given settings of cameras and targets have a solution as an assignment and let a valid assignment be called a **real assignment**. There can be many real assignments possible for an instance of the decesion problem. In this proof we will fix a real assignment (if there is any) and refer to that allthrough. A triple (a_i, b_j, t_p) is said to be a valid **pairing**, if the target t_p lies inside the interval defined by camera pair a_i, b_j. Consider the leftmost target t_1, and let $I_1, I_2, \ldots I_m$ be the intervals in which it lies. Also assume, $I_1 = I_\theta(a_i, b_j)$ be the interval that is finishing earliest among these. Suppose, in the real assignment, t_1 be assigned to the interval $I_2 = (a_k, b_l)$.

By case analysis we show that we can get a solution from the real solution which has the triple (a_i, b_j, t_1) in the assignment. So, we can now remove the triple (a_i, b_j, t_1) from the real assignment and the rest of the $n-1$ targets and $2(n-1)$ cameras will have a solution. The base case ($n = 2$) can be verified easily. The following cases need to be handled:

Case 1. $j \neq n$. If I_1 is one of $I_\theta(a_1, b_1), I_\theta(a_1, b_2), \ldots, I_\theta(a_1, b_{n-1})$, say $I_\theta(a_1, b_j)$, $j \leq n-1$ then only intervals that can cover it are $I_\theta(a_1, b_1), I_\theta(a_1, b_2), \ldots, I_\theta(a_1, b_j)$. So, in the real assignment it must be covered by $I_\theta(a_1, b_1)$ where $k \leq j$. If $k = j$ we are done, else if $I_\theta(a_{i_1}, b_j)$ covers some target t_p then we can swap these to get new pairings as $(a_1, b_j, t_1), (a_{i_1}, b_k, t_p)$ and this will be a valid assignment because of above conditions.

Case 2. $j = n$. The following subcases arise:
(a) $k \leq i$. Let $k < i$, then because of the imposed constraint on the intervals, $I_\theta(a_k, b_l)$ will start before $I_\theta(a_i, b_j)$ and since it is ending after $I_\theta(a_i, b_j)$ it will be nesting the interval $I_\theta(a_i, b_j)$. Hence, from lemma 3 we have $j \leq l$. Now let in the real assignment we have the following 3 assignments $(a_k, b_l, t_1), (a_i, b_{j'}, t_p), (a_{i'}, b_j, t_q)$. We interchange the pairings to get 3 new pairings $(a_i, b_j, t_1), (a_k, b_{j'}, t_p), (a_{i'}, b_l, t_q)$ which are valid as $I_\theta(a_i, b_{j'})$ is nested in $I_\theta(a_k, b_{j'})$ and $I_\theta(a_{i'}, b_j)$ is nested in $I_\theta(a_{i'}, 1)$.

(b) $k > i$. If $k > i$, then starting point of $I_\theta(a_k, b_l)$ will be to the right of that of $I_\theta(a_i, b_j)$ and hence the intervals $I_\theta(a_i, b_j)$ and $I_\theta(a_k, b_l)$ satisfy the conditions of lemma 4, which implies $l < j = n$. Now assume that the real assignment has the triples $(a_k, b_l, t_1), (a_i, b_{j'}, t_p), (a_{i'}, b_j, t_q)$. Now, the new pairings we would like to propose are $(a_i, b_j, t_1), (a_k, b_{j'}, t_p), (a_{i'}, b_l, t_q)$. In this case the only problematic pair is $(a_i, b_{j'}, t_p), (a_k, b_{j'}, t_p)$ because $I_\theta(a_k, b_{j'})$ is nested in $I_\theta(a_i, b_{j'})$. If t_p lies in $I_\theta(a_k, b_{j'})$ then we are done. If not, because of the constraint on the position of cameras t_p can lie only to the left of $I_\theta(a_k, b_{j'})$. Since $j^{'} < n = j$, $I_\theta(a_i, b_j)$ will be nested in $I_\theta(a_i, b_{j'})$. Also l cannot be greater than $j^{'}$ because if it is the case $I_\theta(a_k, b_l)$ will be nested in $I_\theta(a_k, b_{j'})$ and since t_1 is the leftmost camera t_p will lie to the right of it and hence will be covered by $I_\theta(a_k, b_l)$ as well, implying that it will be covered by $I_\theta(a_i, b_{j'})$ as well which is contrary to our assumption. Therefore, $l < j^{'}$. Figure 3 shows the relative position of the intervals in this case. Hence in we can use the pairings $(a_i, b_j, t_1), (a_k, b_l, t_p), (a_{i'}, b_{j'}, t_q)$ to get a valid new solution. □

Therefore we can state the following theorem

Theorem 5 *Under the constraint C, the Parallel-MAX-MIN problem can be solved in $O(n^3 \log^2 n)$ time.*

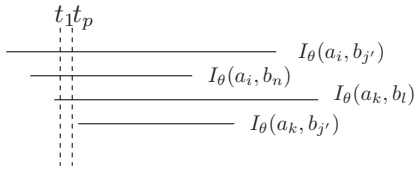


Figure 3: Case 3

Lemma 6 *If all the targets are placed to the left of the right end point of the interval $I_\theta(a_n, b_n)$ and the instance has a solution then there exists a valid assignment in which the leftmost camera is assigned to the leftmost target.*

Proof. In long version of the paper □

Using the above lemma we can prove a result for the case when targets can be placed arbitrarily on the line. For the following lemma assume that the given instance of the decision problem has a valid assignment.

Theorem 7 *When the cameras are positioned such that the interval property is satisfied then there exists a valid assignment in which the leftmost camera is assigned to the leftmost target or the rightmost camera is assigned to the rightmost target.*

Proof. In long version of the paper □

4 Conclusion

We study the Parallel-MAX-MIN version of the Focus Of Attention Problem and give a polynomial time algorithm assuming a constraint on the positioning of the cameras and the targets.

References

[1] Volkan Isler, Sanjeev Khanna, John Spletzer and Camillo J. Taylor, *Target tracking with distributed sensors: The focus of attention problem*, Computer Vision and Image Understanding 100, pp. 225-247, 2005.

[2] Beat Gfeller,Matúš Mihal'ák, Subhash Suri, Elias Vicari and Peter Widmayer *Angle Optimization in Target Tracking*, SWAT 2008, To appear

[3] A. Yilmaz, O. Javed and M. Shah *Object tracking: A survey*, ACM Computing Surveys 38, 2006

[4] R.I. Hartley and A. Zisserman *Multiple View Geometry in Computer Vision*, Second edition, Cambridge University Press, ISBN: 0521540518, 2004

On distinct distances among points in general position and other related problems

Adrian Dumitrescu*

Abstract

A set of points in the plane is said to be in *general position* if no three of them are collinear and no four of them are cocircular. If a point set determines only distinct vectors, it is called *parallelogram free*. We show that there exist n-element point sets in the plane in general position, and parallelogram free, that determine only $O(n^2/\sqrt{\log n})$ distinct distances. This answers a question of Erdős, Hickerson and Pach. We then revisit an old problem of Erdős : given any n points in the plane (or in d dimensions), how many of them can one select so that the distances which are determined are all distinct? — and provide (make explicit) some new bounds in one and two dimensions.

1 Introduction

In 1946, in his classical paper [7] published in the *American Mathematical Monthly*, Erdős raised the following question: What is the minimum number of distinct distances determined by n points in the plane? Denoting this number by $g(n)$, he proved that $g(n) = \Omega(\sqrt{n})$, and showed that $g(n) = O(n/\sqrt{\log n})$ by estimating the number of distinct distances in a $\sqrt{n} \times \sqrt{n}$ piece of the integer grid. He also went further to conjecture that the upper bound is best possible, in other words $g(n) = \Omega(n/\sqrt{\log n})$. The lower bound estimates have been successively raised by Moser (1952), Chung (1984), Beck (1983), Clarkson, Edelsbrunner, Guibas, Sharir and Welzl (1990), Chung, Szemerédi and Trotter (1992), Székely (1997), Solymosi and C. Tóth (2001), Tardos (2003), Katz and Tardos (2004), with the current best lower bound standing at $g(n) \approx \Omega(n^{0.8641})$. This question has lead to many other variants, some of which we discuss here.

Throughout this paper, we say that a set S of points in the plane is in *general position* if no three of them are collinear and no four of them are on a circle.[1] In our bounds, we denote by c possibly different absolute constants.

Erdős asked in 1985 whether there exist n points in general position that determine only $o(n^2)$ distinct distances [9]. Erdős, Hickerson and Pach [13] constructed such point sets with $O(n^{\log 3/\log 2})$ distinct distances, a bound that was later improved by Erdős, Füredi, Pach and Ruzsa [11] to $n2^{c\sqrt{\log n}}$ (for some constant c); see also [25]. It is still an open question whether this number can be linear in n. These constructions use many duplicate vectors, which motivates the further restriction of *no parallelogram*—equivalently, that no two vectors determined by the point set are the same. Erdős, Hickerson and Pach [13] raised the following question: Does there exist a set S of n points in the plane in general position, such that S does not contain all four vertices of a parallelogram, but $g(S)$, the number of distinct distances determined by S, is $o(n^2)$? The question also appears in a paper [10] from 1988, as well as in the recent collection of open problems [5] (Problem 3, Section 5.5, pp. 215). Here we give a positive answer and thus show that the above (three) conditions are not enough to force a quadratic number of distinct distances.

For a prime p, and $x \in \mathbb{Z}$, let $\widehat{x} := x \mod p$ (we view $\widehat{x}$ as an element of $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$). An old construction of Erdős described below (see also [3, pp. 28–29], and [5, pp. 417]) has proved to be instrumental in answering several different questions in combinatorial geometry. Let n be a prime and consider the n-element point set $E_n = \{(i, \widehat{i^2}) \mid i = 0, 1, \ldots, n-1\}$. E_n is a subset of $G_n = \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, n-1\}\}$. An old (still unsolved) question asks how many points can be selected from the $n \times n$ grid G_n so that no three are collinear. Erdős has shown that E_n has no three collinear points, so this gives a first large set with n points; more complicated constructions approach $2n$ from below (obviously $2n$ is an upper bound). The set E_n gives also a first partial answer in the old Heilbronn problem: What is the smallest $a(n)$ such that any set of n points in the unit square determines a triangle whose area is at most $a(n)$? Using the fact that the minimum nonzero area of a triangle in G_n is $1/2$, after suitable scaling E_n to the unit square, one gets the estimate $a(n) \geq \frac{1}{2(n-1)^2}$, i.e., half of the area of a scaled lattice square; see [3, 5]. A more complicated construction due to Komlós, Pintz and Szemerédi [20] yields the current

*Department of Computer Science, University of Wisconsin-Milwaukee, WI 53201-0784, USA. E-mail: ad@cs.uwm.edu. Supported in part by NSF CAREER grant CCF-0444188.

[1] It is not uncommon to find this qualification attached to sets satisfying only one of the two restrictions. Alternatively, one can use the term *strong general position* for sets satisfying both restrictions.

best lower bound $a(n) = \Omega(\frac{\log n}{n^2})$.

Here we use the Erdős construction yet one more time, and show that a suitably large subset $S_n \subset E_n$ is in general position and parallelogram-free ($|S_n| = (n-1)/4$). The fact that S_n determines only $o(n^2)$ distinct distances comes out easily from the old Erdős upper bound of $g(n) = O(n/\sqrt{\log n})$. Let $v(n) = \min g(S)$, where the minimum is taken over all n-element point sets in the plane in general position, and parallelogram free.

Theorem 1 *For every natural number n, $v(n) = O(n^2/\sqrt{\log n})$.*

In the second part we discuss another old problem of Erdős on distinct distances [8, 12, 14]: What is the largest number $h(n)$ so that any set of n points in the plane (or in d dimensions) has an $h(n)$-element subset in which all $\binom{h(n)}{2}$ distances are distinct? Denote this number by $h_d(n)$, and also write $h(n)$ for $h_2(n)$.

The problem on the line ($d = 1$) turns out to be related to the classical *Sidon sequences* [28]. Erdős conjectured that $h_1(n) = (1 + o(1))n^{1/2}$ [14], and observed that the upper bound follows from his 1941 result with Turán [16] on Sidon sequences. By combining various number theoretical results (some of them quite old and possibly forgotten), in particular a powerful result of Komlós et al. [21], one can show:

Theorem 2 *Given a set P of n points in the line, one can select a subset $X \subseteq S$ of size $|X| = \Omega(n^{1/2})$ in which all pairwise distances are distinct. This bound is best possible apart from a constant factor. Thus $h_1(n) = \Theta(n^{1/2})$; more precisely: $(0.0805+o(1))\cdot n^{1/2} \le h_1(n) \le (1 + o(1)) \cdot n^{1/2}$.*

For the planar variant, a $\sqrt{n} \times \sqrt{n}$ section of the integer grid yields $h(n) = O(n^{1/2}(\log n)^{-1/4})$; see also [5]. From the results in [2], it follows that $h(n) = \Omega(n^{1/5})$. The lower bound has been subsequently raised by Lefmann and Thiele [22] to $h(n) = \Omega(n^{1/4})$. By using their method in combination with recent results of Pach and Tardos [26] on the maximum number of isosceles triangles determined by a planar point set, a better bound can be derived. We have included a short outline of the argument in Section 3. Letting $\alpha = \frac{234-68e}{110-32e}$, where e is the base of the natural logarithm ($\alpha < 2.136$), Pach and Tardos proved that the number of isosceles triangles determined by a planar set of n points is $O(n^{\alpha+\varepsilon}) = O(n^{2.136})$, for any $\varepsilon > 0$; see [5, 19]. Put now $\beta = 1 - \frac{\alpha}{3} > 0.288$. The improved lower bound on $h(n)$ is:

Theorem 3 *For any $\varepsilon > 0$, out of any set P of n points in the plane, one can select a subset $X \subseteq S$ of size $|X| = \Omega(n^{\beta-\varepsilon}) = \Omega(n^{0.288})$ in which all pairwise distances are distinct. Thus $h(n) = \Omega(n^{\beta-\varepsilon}) = \Omega(n^{0.288})$.*

For $d \ge 3$, the lower bound $h_d(n) = \Omega(n^{1/(3d-2)})$ is known, cf. [22, 30]; see also [5].

2 Proof of Theorem 1

Assume first that n is a prime. Let $S_n = \{(i, \widehat{i^2}) \mid i = 0, 1, \ldots, (n-1)/4\}$. Recall that a $\sqrt{n} \times \sqrt{n}$ piece of the integer lattice determines $O(n/\sqrt{\log n})$ distances (cf. Erdős, this leads to the upper bound $g(n) = O(n/\sqrt{\log n})$). Therefore G_n determines $O(n^2/\sqrt{\log n})$ distinct distances; obviously this upper bound also holds for any subset of G_n, E_n or S_n in particular. Clearly S_n has no three collinear points (as a subset of E_n). Moreover, Thiele has shown that S_n has no four points cocircular [31]. His result was in the context of finding large subsets of the $n \times n$ grid without four cocircular points in response to a problem raised by Erdős and Purdy; see [5, pp. 418].

To conclude our result in Theorem 1 it remains to be shown that S_n determines no parallelogram. Then the result follows from standard facts about the distribution of primes [18], e.g., that there is a prime between k and $2k$ for any integer $k \ge 1$.

Lemma 4 *S_n determines no parallelogram.*

Proof. Assume (for contradiction) that $ABCD$ is a parallelogram whose vertices are in S_n. Let $A = (a, \widehat{a^2})$, $B = (b, \widehat{b^2})$, $C = (c, \widehat{c^2})$, $D = (d, \widehat{d^2})$. We may assume that $0 \le a < b < c < d \le (n-1)/4$. Observe that AD must be a diagonal, henceforth BC is the other diagonal. Denote by $x(p)$ and $y(p)$ the x- and y-coordinates of a point p. Since the midpoints of the diagonals coincide, we have

$$\frac{x(A) + x(D)}{2} = \frac{x(B) + x(C)}{2}, \text{ and}$$

$$\frac{y(A) + y(D)}{2} = \frac{y(B) + y(C)}{2}.$$

The first equality yields $a + d = b + c$, or equivalently $b - a = d - c$, and note that $b - a$ is a nonzero invertible element of $\mathbb{Z}_n$ (since n is prime). The second equality yields $\widehat{a^2} + \widehat{d^2} = \widehat{b^2} + \widehat{c^2}$, or equivalently $\widehat{b^2} - \widehat{a^2} = \widehat{d^2} - \widehat{c^2}$. Taking this equality modulo n we get

$$(b - a)(b + a) \equiv (d - c)(d + c) \pmod{n}.$$

After simplifying by $b - a$, we obtain $a + b = c + d$, obviously a contradiction, since $1 \le a + b < 2b < 2c < c + d < (n-1)/2$. □

3 Further connections and related problems

A *Sidon sequence* of integers $1 \le a_1 < a_2 < \cdots < a_s \le n$ is one in which the sums of all pairs, $a_i + a_j$, for $i \le j$, are

all different [15, 17, 27]. Suppose that we were to find a large subset $A \subseteq \{1, 2, \ldots, n\}$ in which all distances are distinct. It is easy to see that this amounts to finding a large Sidon sequence in $\{1, 2, \ldots, n\}$: that is, A is a Sidon sequence if and only if the differences (distances) between any two elements are distinct. Indeed, if $a_i < a_j \leq a_k < a_l$, then $a_j - a_i = a_l - a_k$ if and only if $a_i + a_l = a_j + a_k$ (the case of overlapping intervals is similar).

Denote by $s = s(n)$ the maximum number of elements in a Sidon sequence with elements not greater than n. By a packing argument, it follows that $s < 2n^{1/2}$, see [15], and this implies the same upper bound on $h_1(n)$: therefore $h_1(n) = O(n^{1/2})$ (for the moment, we do not insist on the constant). Sharper bounds (with a better constant) have been obtained by Erdős and Turán [16] and Lindström [23]: $s(n) \leq n^{1/2} + n^{1/4} + 1$. From the other direction, the existence of perfect difference sets [29] shows that $s(n) \geq (1 - \varepsilon)n^{1/2}$ [12]. The reader can find more details in [15]. Let $S : 1 \leq a_1 < a_2 < \cdots < a_n$ be any sequence of n integers. By extending the previous lower bound (in a very broad setting), Komlós et al. [21] have proved that S always contains a Sidon subsequence of size $\Omega(n^{1/2})$. From their very general theorem, the resulting constant factor is quite small, about 2^{-15}, but this has been later raised by Abbott [1] to about $0.0805 \gtrapprox 2/25$. Therefore, out of a set of n integer points, one can always find a subset of size $\Omega(n^{1/2})$, with all distinct distances, and this is best possible.

3.1 All distinct distances on the line: proof of Theorem 2

It only remains to show that given n points on the line, one can select a subset of size $\Omega(n^{1/2})$, with all distinct distances. Let $A = \{a_1 < \ldots < a_n\}$ be a set of n points on the line. Using simultaneous approximation [18] (see other applications in [6, 24]), construct a set of n rational points $A' = \{a'_1 < \ldots < a'_n\}$, and then a set of integer points $A'' = \{a''_1 < \ldots < a''_n\}$, so that $a''_j - a''_i = a''_l - a''_k$ holds whenever $a_j - a_i = a_l - a_k$ holds. For any positive integer m, there exist n rational points $A' = \{a'_1 < \ldots < a'_n\} = \{r_1/m, \ldots, r_n/m\}$, where $r_i, m \in \mathbb{N}$, and

$$\left| a_i - \frac{r_i}{m} \right| \leq \frac{1}{m^{1+1/n}}, \quad 1 \leq i \leq n.$$

One can also ensure that the order of the points in A' is $r_1/m < \ldots < r_n/m$ (i.e., $a'_i = r_i/m$, for all i). It can be shown there exists an m large enough such that the structure of distinct distances is preserved, and in particular we have $g(A'') = g(A') = g(A)$.

Now select a large Sidon subsequence from A'' (cf. with the above result of [21], of size $\Omega(n^{1/2})$), and construct a subset $B \subset A$ by including all corresponding points from A into B. By the properties of A'', all pairwise distances in B are distinct, as desired. Taking now into account the best constants available, one has $(0.0805 + o(1)) \cdot n^{1/2} \leq h_1(n) \leq (1 + o(1)) \cdot n^{1/2}$.

3.2 All distinct distances in the plane: proof of Theorem 3

The method of proof is due to Lefmann and Thiele [22]; see also [25]. Denote by $\mathcal{I}(P)$ the number of isosceles triangles spanned by triples of P, where each equilateral triangle is counted three times (this is the same as the number of weighted incidences between perpendicular bisectors determined by P and points of P, where the weight of a bisector is the number of pairs of points for which it is common).

Lemma 5 (Lefmann and Thiele [22]). *Let P be a set of n points in the plane, which determine t distinct distances $d_1, d_2, \ldots, d_t$, where d_i occurs with multiplicity m_i for $i = 1, 2, \ldots, t$. Then*

$$\sum_{i=1}^{t} m_i^2 \leq \frac{n}{2} \left(\mathcal{I}(P) + \binom{n}{2} \right).$$

Using the upper bound $\mathcal{I}(P) = O(n^{\alpha+\varepsilon}) = O(n^{2.136})$ from [26], one obtains from Lemma 5:

$$\sum_{i=1}^{t} m_i^2 = O(n^{1+\alpha+\varepsilon}) = O(n^{3.136}). \tag{1}$$

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane, which determine t distinct distances $d_1, d_2, \ldots, d_t$, where d_i occurs with multiplicity m_i for $i = 1, 2, \ldots, t$. Now define a hypergraph $\mathcal{H} = (P, \mathcal{E}_3 \cup \mathcal{E}_4)$ as follows:

Let $\{p_i, p_j, p_k\} \in \mathcal{E}_3 \subseteq [P]^3$ if and only if $|p_i - p_j| = |p_i - p_k|$, that is, $\Delta p_i p_j p_k$ is an isosceles triangle. Let $\{p_i, p_j, p_k, p_l\} \in \mathcal{E}_4 \subseteq [P]^4$ if and only if $|p_i - p_j| = |p_k - p_l|$, that is, the two segments have the same length. Let $\varepsilon > 0$ be arbitrary small (but fixed). Clearly, $|\mathcal{E}_3| \leq c_3 \cdot n^{\alpha+\varepsilon}$, and $|\mathcal{E}_4| \leq \sum_{i=1}^{t} \binom{m_i}{2} \leq c_4 \cdot n^{1+\alpha+\varepsilon}$, for some constants $c_3, c_4 > 0$.

To conclude the proof, one makes an experiment consisting of two steps: random sampling followed by the deletion method. In the first step, a random subset $X \subset P$ is chosen by selecting points independently with probability $p = c \cdot n^{-\alpha/3-\varepsilon/3}$, for some constant $c > 0$ to be specified later. In the second step, one point from each "surviving" edge in $\mathcal{E}'_3 = [X]^3 \cap \mathcal{E}_3$ and $\mathcal{E}'_4 = [X]^4 \cap \mathcal{E}_4$ is deleted. It results in an independent set Y of $\mathcal{H}$ with average size

$$E[|Y|] \geq (c - c^4 c_4) n^{1-\alpha/3-\varepsilon/3} - c^3 c_3,$$

see [22] for details. By choosing c sufficiently small, and after relabeling ε, one gets $E[|Y|] = \Omega(n^{\beta-\varepsilon})$, where $\beta = 1 - \frac{\alpha}{3}$, as claimed.

References

[1] H. L. Abbott: Sidon sets, *Canadian Mathematical Bulletin*, **33(3)** (1990), 335–341.

[2] D. Avis, P. Erdős, and J. Pach: Distinct distances determined by subsets of a point set in space, *Computational Geometry: Theory and Applications*, **1** (1991), 1–11.

[3] N. Alon and J. Spencer: *The Probabilistic Method*, second edition, Wiley, New York, 2000.

[4] F. Behrend: On sets of integers which contain no three in arithmetic progressions, *Proceedings of the National Academy of Sciences, USA*, **23** (1946), 331–332.

[5] P. Braß, W. Moser, and J. Pach: *Research Problems in Discrete Geometry*, Springer, New York, 2005.

[6] A. Dumitrescu: On distinct distances and λ-free point sets, *Discrete Mathematics* (2007), doi:10.1016/j.disc.2007.11.046.

[7] P. Erdős: On sets of distances of n points, *American Mathematical Monthly*, **53** (1946), 248–250.

[8] P. Erdős: Nehany geometriai problémáról (in Hungarian), *Matematikai Lapok*, **8** (1957), 86–92; *Mathematical Reviews*, **20** (1959), pp. 6056.

[9] P. Erdős: Some of my old and new problems in elementary number theory and geometry, *Proceedings of the Sundance Conference on Combinatorics and Related Topics*, Sundance, Utah, 1985; *Congressus Numerantium* **50** (1985), 97–106.

[10] P. Erdős: Some old and new problems in combinatorial geometry, *Applications of Discrete Mathematics* (Clemson, SC, 1986), pp. 32–37, SIAM, Philadelphia, PA, 1988.

[11] P. Erdős, Z. Füredi, J. Pach and I. Z. Ruzsa: The grid revisited, *Discrete Mathematics*, **111** (1993), 189–196.

[12] P. Erdős and R. Guy: Distinct distances between lattice points, *Elemente der Mathematik*, **25** (1970), 121–123.

[13] P. Erdős, D. Hickerson and J. Pach: A problem of Leo Moser about repeated distances on the sphere, *American Mathematical Monthly*, **96** (1989), 569–575.

[14] P. Erdős and G. Purdy: Extremal problems in combinatorial geometry; in *Handbook of Combinatorics*, Vol. I; (R. L. Graham, M. Grötschel, and L. Lovász, editors), Elsevier, Amsterdam, 1995, pp. 809–874.

[15] P. Erdős and J. Surányi: *Topics in the Theory of Numbers*, second edition, Springer, New York, 2003.

[16] P. Erdős and P. Turán: On a problem of Sidon in additive number theory, and on some related problems, *Journal of London Mathematical Society*, **16** (1941), 212–215.

[17] R. K. Guy: *Unsolved Problems in Number Theory*, third edition, Springer, New York, 2004.

[18] G. H. Hardy and E. M. Wright: *An Introduction to the Theory of Numbers*, fifth edition, Oxford University Press, 1979.

[19] N. Katz and G. Tardos: A new entropy inequality for the Erdős distance problem, in *Towards a Theory of Geometric Graphs*, János Pach (editor), Contemporary Mathematics series of AMS, 2004, pages 119–126.

[20] J. Komlós, J. Pintz and E. Szemerédi: A lower bound for Heilbronn's problem, *Journal of London Mathematical Society*, **25** (1982), 13–24.

[21] J. Komlós, M. Sulyok, and E. Szemerédi: Linear problems in combinatorial number theory, *Acta Mathematica Academiae Scientiarum Hungaricae*, **26(1-2)** (1975), 113–121.

[22] H. Lefmann and T. Thiele: Point sets with distinct distances, *Combinatorica*, **15(3)** (1995), 379–408.

[23] B. Lindström: An inequality for B_2-sequences, *Journal of Combinatorial Theory*, **6** (1969), 211–212.

[24] J. Pach: Midpoints of segments induced by a point set, *Geombinatorics*, **13** (2003), 98–105.

[25] J. Pach and P. K. Agarwal: *Combinatorial Geometry*, Wiley-Interscience, New York, 1995.

[26] J. Pach and G. Tardos: Isosceles triangles determined by a planar point set, *Graphs and Combinatorics*, **18** (2002), 769–779.

[27] C. Pomerance and A. Sárközy: Combinatorial Number Theory; in *Handbook of Combinatorics*, Vol. I; (R. L. Graham, M. Grötschel, and L. Lovász, editors), Elsevier, Amsterdam, 1995, pp. 967–1018.

[28] S. Sidon: Ein Satz über trigonometrische Polynome und seine Anwendung in der Theorie der Fourier-Reihen, *Mathematische Annalen*, **106** (1932), 536–539.

[29] J. Singer: A theorem in finite projective geometry and some applications to number theory, *Transactions of American Mathematical Society*, **43** (1938), 377–385.

[30] T. Thiele: *Geometric Selection Problems and Hypergraphs*, Dissertation, Freie Universität Berlin, 1995.

[31] T. Thiele: The no-four-on-circle problem, *Journal of Combinatorial Theory Ser. A*, **71** (1995), 332–334.

Monochromatic simplices of any volume

Adrian Dumitrescu* Minghui Jiang†

Abstract

We give a very short proof of the following result of Graham from 1980: For any finite coloring of $\mathbb{R}^d$, $d \geq 2$, and for any $\alpha > 0$, there is a monochromatic $(d+1)$-tuple that spans a simplex of volume α. Our proof also yields new estimates on the number $A = A(r)$ defined as the minimum positive value A such that, in any r-coloring of the grid points $\mathbb{Z}^2$ of the plane, there is a monochromatic triangle of area exactly A.

1 Introduction

The classical theorem of Van der Waerden states that if the set of integers $\mathbb{Z}$ is partitioned into two classes then at least one of the classes must contain an arbitrarily long arithmetic progression [20]. The result holds for any fixed number of classes [17]. Let $W(k, r)$ denote the Van der Waerden numbers: $W = W(k, r)$ is the least integer such that for any r-coloring of $[1, W]$, there is a monochromatic arithmetic progression of length k. The following generalization of Van der Waerden's theorem to two dimensions is given by Gallai's theorem [17]: If the grid points $\mathbb{Z}^2$ of the plane are finitely colored, then for any $t \in \mathbb{N}$, there exist $x_0, y_0, h \in \mathbb{Z}$ such that the t^2 points $\{(x_0 + ih, y_0 + jh) \mid 0 \leq i, j \leq t-1\}$ are of the same color.

Many extensions of these Ramsey type problems to the Euclidean space have been investigated in a series of papers by Erdős et al. [9, 10, 11] in the early 1970s, and by Graham [12, 13, 14]. See also Ch. 6.3 in the problem collection by Braß, Moser and Pach [4], and the recent survey articles by Braß and Pach [3] and by Graham [15, 16]. For a related coloring problem on the integer grid, see [6].

In 1980, answering a question of Gurevich, Graham [12] proved that for any finite coloring of the plane, and for any $\alpha > 0$, there is a monochromatic triangle of area α. In their survey article, Braß and Pach [3] observed that for any 2-coloring of the plane there is a monochromatic triple that spans a triangle of unit area, and asked whether this holds for any finite coloring, apparently unaware of Graham's solution [12]. This also brought the problem to our attention. Graham's proof was quite involved, and was later simplified by Adhikari [1] using the same main idea. Adhikari and Rath [2] have subsequently obtained a similar result for trapezoids. See also [8] for discussions on this and other related problems. Here we present a very short proof of Graham's result [12] in the following theorem, which gives new insight into the problem and also has quantitative implications (see Theorem 4).

Theorem 1 (Graham [12]). *For any finite coloring of the plane, and for any $\alpha > 0$, there is a monochromatic triangle of area α.*

As a corollary of the planar result, one obtains a similar result concerning simplices in d-space for all $d \geq 2$. This was pointed out by Graham [12] without giving details. For completeness, we include our short proof of the following theorem.

Theorem 2 (Graham [12]). *Let $d \geq 2$. For any finite coloring of $\mathbb{R}^d$, and for any $\alpha > 0$, there is a monochromatic $(d+1)$-tuple that spans a simplex of volume α.*

Using a general "product" theorem for Ramsey sets [12, Theorem 3], Graham extended Theorem 2 to the following much stronger result that accommodates all values of α in the same color class. Theorem 3 below can also be obtained using the same "product" theorem in conjunction with our short proof of Theorem 2.

Theorem 3 (Graham [12]). *Let $d \geq 2$. For any finite coloring of $\mathbb{R}^d$, some color class has the property that, for any $\alpha > 0$, it contains a monochromatic $(d+1)$-tuple that spans a simplex of volume α.*

Let $r \geq 2$. Graham [12] defined the number $T = T(r)$ as the minimum value $T > 0$ such that, in any r-coloring of the grid points $\mathbb{Z}^2$ of the plane, there is a monochromatic *right* triangle of area exactly T. We now define $A(r)$ for *arbitrary* triangles. Let $A = A(r)$ be the minimum value $A > 0$ such that, in any r-coloring of the grid points $\mathbb{Z}^2$ of the plane, there is a monochromatic grid triangle of area exactly A. Graham's proof of Theorem 1 [12] shows that $T(r)$ exists, which obviously implies the existence of $A(r)$. We clearly have $A(r) \leq T(r)$.

*Department of Computer Science, University of Wisconsin–Milwaukee, Milwaukee, WI 53201-0784, USA. Email: ad@cs.uwm.edu. Supported in part by NSF CAREER grant CCF-0444188.

†Department of Computer Science, Utah State University, Logan, UT 84322-4205, USA. Email: mjiang@cc.usu.edu. Supported in part by NSF grant DBI-0743670.

Graham [12] obtained an upper bound $T(r) \le \widehat{T}(r) = S_1 \cdot S_2 \cdots S_r$, where

$$S_1 = 1, \quad S_{i+1} = (S_i + 1)! \cdot W(2(S_i + 1)! + 1, i + 1)!\,.$$

In Theorem 4 below, we derive an upper bound $A(r) \le \widehat{A}(r)$, and show that $\widehat{A}(r) \ll \widehat{T}(r)$. While Graham [12] finds a *right* monochromatic grid triangle of area exactly $\widehat{T}(r)$, we find an *arbitrary* monochromatic grid triangle of area exactly $\widehat{A}(r)$. However, as far as we are concerned in answering the original question of Gurevich, or the question of Braß and Pach [3], this aspect is irrelevant.

For the lower bound, we clearly have $A(r) \ge 1/2$ because the triangles are spanned by grid points. Let l.c.m. denote the least common multiple of a set of numbers. Graham [12] notes the following lower bound for $T(r)$ based on cyclic colorings of $\mathbb{Z}^2$ (without giving details):

$$T(r) \ge \frac{1}{2} \times \text{l.c.m.}\,(2, 3, \ldots, r) = e^{(1+o(1))r}.$$

We will show that the same lower bound holds for $A(r)$ as well.

Theorem 4 *Let $A = A(r)$ be the minimum value $A > 0$ such that, in any r-coloring of the grid points $\mathbb{Z}^2$ of the plane, there is a monochromatic triangle of area exactly A. Let*

$$H = \left\lfloor \frac{W(r! + 1, 2^r - 1) - 1}{r!} \right\rfloor, \text{ and } \widehat{A}(r) = H! \cdot r!.$$

Then $\frac{1}{2} \times \text{l.c.m.}\,(2, 3, \ldots, r) \le A(r) \le \widehat{A}(r)$, where $\widehat{A}(r) \ll \widehat{T}(r)$ for sufficiently large r.

It is worth noting the connection between the problems we discussed here and the following old and probably difficult problem of Erdős [7, 8]: Does there exist an absolute constant B such that any measurable plane set E of area B contains the vertices of a unit-area triangle? The answer is known only in certain special cases: if E has infinite area, or even if E has positive area but is unbounded, then E has the desired property; see [5, Problem G13, pp. 182] and [19]. It follows that if in a finite coloring of the plane each color class is measurable, then the largest color class, say E, has infinite area, and hence there is a monochromatic triple that spans a triangle of unit area. But of course, this case is already covered by Theorem 1.

2 Proof of Theorem 1

Let $R = \{1, 2, \ldots, r\}$ be the set of colors. Pick a Cartesian coordinate system (x, y). Consider the finite coloring of the lines induced by the coloring of the points: each line is colored (labeled) by the subset of colors $R' \subseteq R$ used in coloring its points. Note that this is a $(2^r - 1)$-coloring of the lines.

Set $N = W(r!+1, 2^r-1)$. By Van der Waerden's theorem, any (2^r-1)-coloring of the N horizontal lines $y = i$, $i = 0, 1, \ldots, N-1$, contains a monochromatic arithmetic progression of length $r! + 1$: $y_0, y_0 + k, \ldots, y_0 + r!k$. Let $\mathcal{L} = \{\ell_i \mid 0 \le i \le r!\}$, where $\ell_i : y = y_0 + ik$ for some integers $y_0 \ge 0$, $k \ge 1$. Each of these lines is colored by the same set of colors, say $R' \subseteq R$.

Set $x = 2\alpha/r!k$. Consider the $r + 1$ points of ℓ_0 with x-coordinates $0, x, \ldots, rx$. By the pigeon-hole principle, two of these points, say a and b, share the same color, and their distance is jx for some $j \in R$. Pick any point c of the same color on the line $\ell_{r!/j}$ (note that $r!/j$ is a valid integer index, and this is possible by construction!). The three points a, b, c span a monochromatic triangle Δabc of area

$$\frac{1}{2} \cdot jx \cdot \frac{r!k}{j} = \frac{1}{2} \cdot \frac{2j\alpha}{r!k} \cdot \frac{r!k}{j} = \alpha,$$

as required.

3 Proof of Theorem 2

We proceed by induction on d. The basis $d = 2$ is verified in Theorem 1. Let now $d \ge 3$. Assume that the statement holds for dimension $d-1$, and we prove it for dimension d.

Consider the finite coloring of the hyperplanes induced by the coloring of the points by a set R of r colors: each hyperplane is colored (labeled) by the subset of colors $R' \subseteq R$ used in coloring its points. We thus get a $(2^r - 1)$-coloring of the hyperplanes. Pick a Cartesian coordinate system $(x_1, \ldots, x_d)$, and consider the set of parallel hyperplanes $x_d = i$, $i \in \mathbb{N}$. Let π_1 and π_2 be two parallel hyperplanes colored by the same set of colors, say $R' \subseteq R$. Let h be the distance between π_1 and π_2. By induction, π_1 has a monochromatic d-tuple that spans a simplex of volume $\alpha d/h$. Pick a point of the same color in π_2, and note that together they form a $(d + 1)$-tuple that spans a simplex of volume

$$\frac{1}{d} \cdot \frac{\alpha d}{h} \cdot h = \alpha,$$

as required.

4 Proof of Theorem 4

We note that our short proof of Theorem 1 does not imply the existence of $A(r)$, since the triangle found there is not necessarily a *grid* triangle. We proceed as in the proof of Theorem 1, but with different settings for the parameters. Set $\alpha = \widehat{A}(r)$. We will show that there is a grid triangle of area exactly α.

Let $R = \{1, 2, \ldots, r\}$ be the set of colors. Pick a Cartesian coordinate system (x, y). Consider the finite coloring of the lines induced by the coloring of the *grid* points on the lines: each line is colored (labeled) by the subset of colors $R' \subseteq R$ used in coloring its grid points. Note that this is a $(2^r - 1)$-coloring of the lines.

Set $N = W(r! + 1, 2^r - 1)$. By Van der Waerden's theorem, any $(2^r - 1)$-coloring of the N horizontal grid lines $y = i$, $i = 0, 1, \ldots, N - 1$, contains a monochromatic arithmetic progression of length $r! + 1$: $y_0, y_0 + k, \ldots, y_0 + r!k$. Let $\mathcal{L} = \{\ell_i \mid 0 \le i \le r!\}$, where $\ell_i : y = y_0 + ik$ for some integers $y_0 \ge 0$, $k \ge 1$. Each of these grid lines is colored by the same set of colors, say $R' \subseteq R$. The common difference of this arithmetic progression is

$$k \le \left\lfloor \frac{W(r! + 1, 2^r - 1) - 1}{r!} \right\rfloor = H.$$

Set $x = 2\alpha / r!k$. Since $\alpha = \widehat{A}(r) = H! \cdot r!$, we have $x = 2H!/k \in \mathbb{N}$. Consider the $r + 1$ grid points on ℓ_0 with x-coordinates $0, x, \ldots, rx$. By the pigeon-hole principle, two of these points, say a and b, share the same color, and their distance is jx for some $j \in R$. Pick any grid point c of the same color on the line $\ell_{r!/j}$ (note that $r!/j$ is a valid integer index, and this is possible by construction!). The three grid points a, b, c span a monochromatic triangle Δabc of area

$$\frac{1}{2} \cdot jx \cdot \frac{r!k}{j} = \frac{1}{2} \cdot \frac{2j\alpha}{r!k} \cdot \frac{r!k}{j} = \alpha,$$

as required. This completes the proof of the existence of $A(r)$ and the upper bound $A(r) \le \widehat{A}(r)$.

We next show the lower bound for $A(r)$. Consider (independently) the following $r - 1$ colorings λ_j, $j = 2, \ldots, r$. The coloring λ_j colors grid point (x, y) with color $(y \mod j)$. Observe that the area of a triangle with vertices (x_i, y_i), $i = 1, 2, 3$, is

$$\frac{|x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + x_3y_1 - x_1y_3|}{2}.$$

Let Δ be a monochromatic grid triangle of area $A(r)$ in this coloring. By symmetry, there is a congruent triangle Δ_0 of color 0, whose y-coordinates satisfy $y_1 \equiv y_2 \equiv y_3 \equiv 0 \pmod{j}$. Hence $2A(r)$ is a nonzero multiple of j. By repeating this argument for each j, we get that $2A(r)$ is a nonzero multiple of all numbers $2, \ldots, r$, hence also of l.c.m. $(2, 3, \ldots, r)$. This completes the proof of the lower bound.

We now prove that $\widehat{A}(r) \ll \widehat{T}(r)$. Although our estimates $\widehat{A}(r)$ also depend on the Van der Waerden numbers $W(k, r)$, the dependence shows a much more modest growth rate for $\widehat{A}(r)$ than for $\widehat{T}(r)$. For instance, since $W(3, 3) = 27$, we have $\widehat{A}(2) \le 13! \cdot 2! \approx 10^{10}$, while $\widehat{T}(2) \le 2W(5, 2)! = 2 \cdot 178! \approx 10^{325}$. We have only very imprecise estimates on Van der Waerden numbers available. The current best upper bound, due to Gowers [18], gives

$$W(k, r) \le 2^{2^{f(k,r)}}, \quad \text{where } f(k, r) = r^{2^{2^{k+9}}}.$$

In particular,

$$W(7, 7) \le 2^{2^{7^{2^{2^{16}}}}}, \text{ and } \widehat{A}(3) = \left\lfloor \frac{W(7, 7) - 1}{6} \right\rfloor! \cdot 6\,.$$

On the other hand, Graham's estimate

$$\widehat{T}(3) = 2 \cdot 178!(2 \cdot 178! + 1) \cdot W(2 \cdot (2 \cdot 178! + 1)! + 1, 3)!$$

appears to be much larger.

The ratio between the two estimates amplifies even more for larger values of r. Let now $r \ge 4$. We have

$$\begin{aligned} \widehat{A}(r) = H! \cdot r! &= \left\lfloor \frac{W(r! + 1, 2^r - 1) - 1}{r!} \right\rfloor! \cdot r! \\ &\le W(r! + 1, 2^r - 1)!\,. \end{aligned}$$

Write $\log^{(i)} x$ for the ith iterated binary logarithm of x. Using again very rough approximations such as

$$r! + 10 \le 2^{2^{r-1}} \quad \text{and} \quad 2^{2^{2^{2^{r-1}}}} \cdot r \le 2^{2^{2^{2^r}}},$$

we obtain

$$\begin{aligned} &\log^{(2)} \widehat{A}(r) \le W(r! + 1, 2^r - 1), \\ &\log^{(2)} W(r! + 1, 2^r - 1) \le f(r! + 1, 2^r - 1), \\ &\log^{(1)} f(r! + 1, 2^r - 1) \le 2^{2^{r!+10}} \log(2^r - 1) \le 2^{2^{2^{2^r}}}, \\ &\log^{(4)} 2^{2^{2^{2^r}}} = r. \end{aligned}$$

It follows that

$$\log^{(9)} \widehat{A}(r) \le r. \tag{1}$$

On the other hand, even if we ignore the predominant factor $W(2(S_i + 1)! + 1, i + 1)!$ in the expression of S_{i+1} when estimating $\widehat{T}(r)$, the inequality $S_{i+1} \ge (S_i + 1)! \ge 2^{S_i}$ still implies that

$$\widehat{T}(r) \ge 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}, \quad \text{a tower of } r \text{ 2s.} \tag{2}$$

By comparing the two inequalities (1) and (2), we conclude that $\widehat{A}(r) \le \widehat{T}(r)$ for $r \ge 12$, and that $\widehat{A}(r) \ll \widehat{T}(r)$ for sufficiently large r. This gives a partial answer to Graham's question[1] raised in the conclusion of his paper [12], and completes the proof of Theorem 4.

Finally, observe that we can replace $H!$ and $r!$ with the smaller numbers l.c.m. $(2, 3, \ldots, H)$ and l.c.m. $(2, 3, \ldots, r)$, respectively, and thereby obtain:

[1] The proof by Adhikari [1] gives an alternative upper bound $T(r) \le \widehat{T'}(r)$. The reader can check that the same tower of 2s expression in (2) is also a lower bound on his estimate $\widehat{T'}(r)$ for $r \ge 2$.

Corollary 5 *Let*

$$H' = \left\lfloor \frac{W(\text{l.c.m.}\,(2,3,\ldots,r)+1, 2^r-1)-1}{\text{l.c.m.}\,(2,3,\ldots,r)} \right\rfloor.$$

Then

$$\begin{aligned} A(r) &\geq \frac{1}{2} \times \text{l.c.m.}\,(2,3,\ldots,r) = e^{(1+o(1))r}, \text{ and} \\ A(r) &\leq \text{l.c.m.}\,(2,3,\ldots,H') \times \text{l.c.m.}\,(2,3,\ldots,r). \end{aligned}$$

It is an easy exercise to show that the above lower bound is tight for $r = 2$, that is, $A(2) = 1$. Consider two cases:

1. If the 2-coloring of $\mathbb{Z}^2$ follows a chess-board pattern, say point (x, y) is colored $(x+y) \mod 2$, then clearly there is a monochromatic triangle of area 1, for example the triangle with vertices $(0,0)$, $(1,1)$, and $(0,2)$.

2. Otherwise, there are two adjacent points of the same color, say $(0,0)$ and $(1,0)$ of color 0. Suppose there is no monochromatic triangle of area 1. Then $(0,2)$ and $(2,2)$ would have color 1. Then $(0,1)$ and $(2,1)$ would have color 0. Then the triangle with vertices $(0,0)$, $(0,1)$, and $(2,1)$ would have color 0 and area 1, a contradiction.

References

[1] S. D. Adhikari: On a theorem of Graham, *Expositiones Mathematicae* **19** (2001), 365–367.

[2] S. D. Adhikari and P. Rath: Monochromatic configurations for finite colourings of the plane, *Note di Matematica* **22** (2003), 59–63.

[3] P. Braß and J. Pach: Problems and results on geometric patterns, in *Graph Theory and Combinatorial Optimization* (D. Avis et al., editors), Springer, 2005, pp. 17–36.

[4] P. Braß, W. Moser, and J. Pach: *Research Problems in Discrete Geometry*, Springer, New York, 2005.

[5] H. T. Croft, K. J. Falconer, and R. K. Guy: *Unsolved Problems in Geometry*, Springer, New York, 1991.

[6] A. Dumitrescu and R. Radoičić: On a coloring problem for the integer grid, in *Towards a Theory of Geometric Graphs*, (J. Pach, editor), Contemporary Mathematics Series of AMS, 2004, pp. 67–74.

[7] P. Erdős: Set-theoretic, measure-theoretic, combinatorial, and number-theoretic problems concerning point sets in Euclidean space, *Real Anal. Exchange*, **4(2)** (1978/79), 113–138.

[8] P. Erdős: Some combinatorial, geometric and set theoretic problems in measure theory, in *Measure Theory*, Oberwolfach, 1983. Lecture Notes in Math. **108**, Springer, Berlin, 1984, pp. 321–327.

[9] P. Erdős, R. Graham, P. Montgomery, B. Rothschild, J. Spencer, and E. Straus: Euclidean Ramsey theorems, I, *J. Combinatorial Theory Ser. A* **14** (1973), 341–363.

[10] P. Erdős, R. Graham, P. Montgomery, B. Rothschild, J. Spencer, and E. Straus: Euclidean Ramsey theorems, II, in *Infinite and Finite Sets*, vol. I; Colloq. Math. Soc. János Bolyai, vol. 10, North-Holland, Amsterdam, 1975, pp. 529–557.

[11] P. Erdős, R. Graham, P. Montgomery, B. Rothschild, J. Spencer, and E. Straus: Euclidean Ramsey theorems, III, in *Infinite and Finite Sets*, vol. I; Colloq. Math. Soc. János Bolyai, vol. 10, North-Holland, Amsterdam, 1975, pp. 559–583.

[12] R. Graham: On Partitions of $\mathbb{E}^n$, *J. Combinatorial Theory Ser. A*, **28** (1980), 89–97.

[13] R. Graham: Topics in Euclidean Ramsey Theory, in *Mathematics of Ramsey Theory* (J. Nesetril and V. Rodl, editors), Springer-Verlag, New York, (1990), pp. 200–213.

[14] R. Graham: Recent trends in Euclidean Ramsey Theory, *Discrete Mathematics*, **136** (1994), 119–127.

[15] R. Graham: Some of my favorite problems in Ramsey Theory, *Integers: Electronic Journal of Combinatorial Number Theory*, **7(2)** (2007), #A15.

[16] R. Graham: Old and new problems and results in Ramsey theory, preprint, 2007, available at `http://www.math.ucsd.edu/~fan/ron/papers/papers.html`.

[17] R. Graham, B. Rothschield, and J. Spencer: *Ramsey Theory*, second edition, John Wiley, New York, 1990.

[18] B. M. Landman and A. Robertson: *Ramsey Theory on the Integers*, American Mathematical Society, 2004.

[19] D. Mauldin: Some problems in set theory, geometry and analysis, in *Paul Erdős and his Mathematics I* (G. Halasz, L. Lovasz, M. Simonovits and V. T. Sos, editors), Bolyai Society Mathematical Studies, vol. I, 2002, pp. 493–505.

[20] B. van der Waerden: Beweis einer Baudetschen Vermutung, *Nieuw Arch. Wiskunde*, **15** (1927), 212–216.

Empty Monochromatic Triangles*

Oswin Aichholzer† Ruy Fabila-Monroy‡ David Flores-Peñaloza§ Thomas Hackl¶ Clemens Huemer‖
Jorge Urrutia**

Abstract

We consider a variation of a problem stated by Erdös and Guy in 1973 about the number of convex k-gons determined by any set S of n points in the plane. In our setting the points of S are colored and we say that a spanned polygon is monochromatic if all its points are colored with the same color.

As a main result we show that any bi-colored set of n points in $\mathcal{R}^2$ in general position determines a super-linear number of empty monochromatic triangles, namely $\Omega(n^{5/4})$.

1 Introduction

Erdös and Guy [6] asked the following question. "What is the least number of convex k-gons determined by any set of n points[1] in the plane?" The trivial solution for the case $k = 3$ is $\binom{n}{3}$. In addition, if we require the triangles to be empty then Katchalski and Meir [8] showed that for all $n \geq 3$ a lower bound is given by $\binom{n-1}{2}$ and that there exists a constant $c > 0$ such that cn^2 is an upper bound. Around the same time Bárány and Füredi [1] showed that any set of n points has at least $n^2 - O(n \log n)$ empty triangles and they also gave an upper bound of $2n^2$ if n is a power of 2.

Valtr [12] described a configuration of n points related to Horton sets [7] with fewer than $1.8n^2$ empty triangles and also provided upper bounds on the number of empty k-gons, e.g. $2.42n^2$ empty quadrilaterals. Later Dumitrescu [5] improved the upper bound for triangles to $\approx 1.68n^2$, which then consequently was further improved by Bárány and Valtr [2] to $\approx 1.62n^2$, the currently best bound. It is still unknown whether the constant could be smaller than 1, that is, whether there exists a family of n-element sets with fewer than n^2 empty triangles.

We consider a related problem, where the points of the given set S are colored. A polygon spanned by points in S is called monochromatic if all its points are colored with the same color. In contrast to the above described race for the best constant for the uncolored case, we are interested in the asymptotic behavior of the number of empty monochromatic triangles for bi-colored point sets.

A result in this direction was obtained by Devillers et al. [4]. They proved that any bi-colored point set in the plane exhibits at least $\lceil \frac{n}{4} \rceil - 2$ interior disjoint empty monochromatic triangles. In a generalization Urrutia [11] showed that in any 4-colored point set in $\mathcal{R}^3$ there is at least a linear number of empty monochromatic tetrahedra.

One might be also interested in the minimum number of colors so that we can color any given set S of n points in a way such that S does not determine an empty monochromatic triangle (or in general an empty monochromatic convex k-gon). In [4] (Theorem 3.3) this question has been settled by showing that already for three colors there are sets not spanning any empty monochromatic triangle.

The remaining question is to determine the asymptotic behavior of the number of empty monochromatic triangles for bi-colored sets. We show that any bi-colored set of n points in $\mathcal{R}^2$ in general position determines $\Omega(n^{5/4})$ empty monochromatic triangles. To the best of our knowledge no non-trivial bounds have been known before.

2 Lower Bound Construction

We start with a technical lemma which shows that for point sets with a triangular convex hull there exists a triangulation such that a sufficient fraction of its triangles are incident to vertices of the convex hull.

Lemma 1 *Let S be a set of n points in general position in the plane with 3 extreme points, that is, with a triangular convex hull and $m = n - 3$ interior points. Then*

*Research supported by the Austrian FWF Joint Research Project 'Industrial Geometry' S9205-N12. Part of the work was done while the second, third and fifth author were visiting Graz.

†Institute for Software Technology, Graz University of Technology, oaich@ist.tugraz.at

‡Instituto de Matemáticas, Universidad Nacional Autónoma de México, ruy@ciencias.unam.mx

§Instituto de Matemáticas, Universidad Nacional Autónoma de México, dflores@math.unam.mx

¶Institute for Software Technology, Graz University of Technology, thackl@ist.tugraz.at

‖Dept. Matemática Aplicada II. Universitat Politècnica de Catalunya, clemens.huemer@upc.edu

**Instituto de Matemáticas, Universidad Nacional Autónoma de México. Research supported in part by MTM2006-03909 (Spain) and CONCYT of México, Proyecto SEP-2004-Co1-45876, urrutia@matem.unam.mx

[1]Throughout, all considered point sets are in general position, that is, they do not contain three collinear points.

S can be triangulated such that at least $m+\sqrt{m}+1$ triangles have (at least) as one of their vertices an extreme point of S.

Proof. Let Δ be the convex hull of S, $E(\Delta)$ the edges of Δ, and $M = S \setminus \Delta = \{q_1, ..., q_m\}$ the interior points of S, $|M| = m = n - 3$.

We first define a partial order $\leq_e$ on the elements of M. Two points $p_1, p_2 \in M$ are comparable with respect to an edge $e \in E(\Delta)$ if the open triangle formed by e and p_1 is contained in the closed triangle formed by e and p_2 ($p_1 \leq_e p_2$) or vice versa ($p_2 \leq_e p_1$), see Figure 1 for an example.

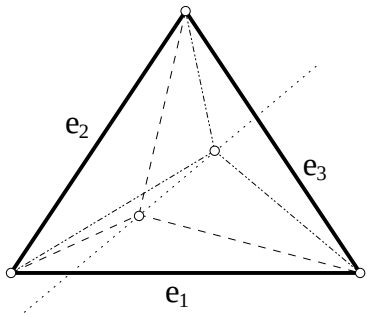


Figure 1: Two points are comparable w.r.t. e_1 and e_3, but incomparable w.r.t. e_2.

Observe that two fixed points $p_1, p_2 \in M$ are comparable w.r.t. exactly 2 out of 3 edges of Δ. This can be seen by considering the supporting line of the edge p_1, p_2, see Figure 1. Two points are comparable w.r.t. an edge e of Δ if and only if this supporting line intersects e. This implies that if two points are not comparable w.r.t. e then they are comparable w.r.t. both other edges.

A chain is an ordered set of (pairwise) comparable points of M and an anti-chain is a set of pairwise incomparable points of M. By Dilworth's Theorem [3] there exists a chain or an anti-chain in M w.r.t. a given edge e of Δ of size $\sqrt{m}$. Because an anti-chain for e is a chain for the other two edges of Δ, we may assume w.l.o.g that there exists a chain $q_{i_1} \leq_e \cdots \leq_e q_{i_{\sqrt{m}}}$ w.r.t. e.

We obtain a triangulation of $\Delta \cup \{q_{i_1}, \ldots, q_{i_{\sqrt{m}}}\}$ by joining each q_{i_j}, $1 \leq j < \sqrt{m}$, to $q_{i_{j+1}}$ and to the end-points of e, and $q_{i_{\sqrt{m}}}$ to the vertices of Δ, see Figure 2, left. There are $2\sqrt{m}+1$ triangles in this triangulation and all of them have at least one vertex on the convex hull. We now extend the triangulation to cover the remaining points. For each point q_i not in the chain there is precisely one end-point p of e visible to q_i and we add the edge joining q_i and p.

We have, so far, a collection of pairwise non-crossing edges, and we complete this to a triangulation of $\Delta \cup \{q_1, \ldots q_m\}$, see Figure 2, right. There are $2\sqrt{m}+m-\sqrt{m}+1 = m+\sqrt{m}+1$ triangles in this triangulation with at least one of its vertices on the convex hull. □

We now generalize the above result to sets with larger convex hulls. Let $CH(S)$ denote the set of vertices of the convex hull of S and $|CH(S)|$ its cardinality, that is, the number of extreme points of S.

Lemma 2 *(Order Lemma) Let S be a set of n points in general position in the plane with $h = |CH(S)|$ extreme points. Then S can be triangulated such that at least $n+\sqrt{n-h}-2$ triangles have (at least) as one of their vertices an extreme point of S.*

Proof. Consider an arbitrary triangulation of the h convex hull points of S (ignoring inner points). Let $\tau_1, \ldots, \tau_{h-2}$ be the obtained triangles and let s_i be the number of points of S interior to τ_i. By Lemma 1 each triangle τ_i can be triangulated such that at least $s_i + \sqrt{s_i} + 1$ triangles have one of its vertices on the convex hull of τ_i and therefore on the convex hull of S. Taking the sum over all τ_i we have: $\sum_{i=1}^{h-2}(s_i+\sqrt{s_i}+1) = \sum_{i=1}^{h-2} s_i + \sum_{i=1}^{h-2}\sqrt{s_i} + \sum_{i=1}^{h-2} 1 = (n-h) + \sum_{i=1}^{h-2}\sqrt{s_i} + (h-2) \geq n + \sqrt{\sum_{i=1}^{h-2} s_i} - 2 = n + \sqrt{n-h} - 2$. □

For the next result we consider bi-colored sets. We will show that if the cardinality of the two color classes differs significantly then this implies the existence of a large number of empty monochromatic triangles.

Lemma 3 *(Discrepancy Lemma) Let S be a set of n points in general position in the plane, partitioned in a red set R and a blue set B with $|R| = |B| + \alpha$, $\alpha \geq 2$. Then S determines at least $\frac{(\alpha-2)}{6}(n+\alpha)$ empty monochromatic triangles.*

Proof. Consider a red point $r \in R$ and the star connecting r to all vertices $R \setminus r$. Completing this star to a triangulation of R gives at least $|R| - 2$ triangles having r as a vertex. At least $\alpha - 2$ of these triangles are empty of points from B, as $|B| = |R| - \alpha$. Repeating this process for all points in R we obtain at least $\frac{(\alpha-2)}{3}|R| = \frac{(\alpha-2)}{3}\frac{n+\alpha}{2} = \frac{(\alpha-2)}{6}(n+\alpha)$ empty red triangles, since we over-count a triangle at most 3 times. □

Note that for the monochromatic case the Discrepancy Lemma implies the $\Omega(n^2)$ bound on the number of empty triangles given in [8], although the constants are slightly worse.

We are now ready to prove our main result.

Theorem 4 *Any bi-colored set of n points in the plane in general position determines $\Omega(n^{5/4})$ empty monochromatic triangles.*

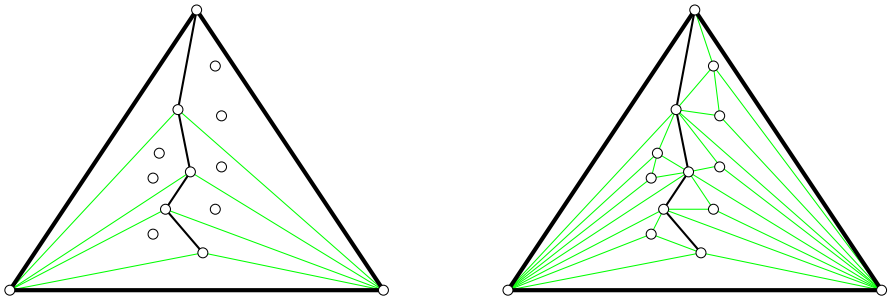


Figure 2: A triangulation for a chain and its extended triangulation

Proof. The general idea behind the proof is to iteratively peel a monochromatic convex layer of the point set. For each layer we use the Order Lemma to obtain roughly $\sqrt{n}$ empty monochromatic triangles. If at any moment the difference of the cardinality of the two color classes is too large we utilize the Discrepancy Lemma and terminate the process. Otherwise we stop after at most $\frac{1}{3}n^{3/4}$ steps.

Let S_1 be the given bi-colored set of n points, with R_1 the set of red and B_1 the set of blue points. Let $\tilde{n} = \frac{n}{6}$. For each iteration step we construct smaller sets $S_{i+1} \subset S_i$, $R_{i+1} \subseteq R_i$, and $B_{i+1} \subseteq B_i$, respectively, with $S_{i+1} = R_{i+1} \cup B_{i+1}$. As an invariant we will have that in any step $|S_i| \geq 2\tilde{n}$ holds. The iteration stops either if at some step the discrepancy between the two sets is larger than $\tilde{n}^{1/4}$ or after at most $\frac{1}{3}n^{3/4}$ steps.

Consider the i-th step of the iteration and w.l.o.g. let $|R_i| \geq |B_i|$. There are two possible cases.

(a) If $|R_i| - |B_i| \geq \tilde{n}^{1/4}$ we apply the Discrepancy Lemma and get at least $\frac{(\tilde{n}^{1/4}-2)}{6}(2 * \tilde{n} + \tilde{n}^{1/4}) = \Omega(n^{5/4})$ empty monochromatic triangles.

(b) Otherwise build the convex hull of the red points and let $B'_i \subseteq B_i$ be the blue points outside of this convex hull. We denote by $r_i = |R_i|$ and $b_i = |B_i \setminus B'_i|$. We have $r_i \geq \tilde{n}$ by our invariant assumption, $r_i \geq b_i$, and $r_i \leq b_i + \tilde{n}^{1/4}$, as otherwise we apply the Discrepancy Lemma to $R_i \cup B_i \setminus B'_i$ and terminate the iteration with $\Omega(n^{5/4})$ empty monochromatic triangles as above. Note that the latter inequality implies that $|B'_i| < \tilde{n}^{1/4}$.

We apply the Order Lemma to R_i and get at least $r_i + \sqrt{r_i - |CH(R_i)|} - 2$ monochromatic (red) triangles which are by construction a subset of a triangulation of R_i incident to $CH(R_i)$. At most b_i of these triangles may contain a blue point, so we get at least $r_i - b_i + \sqrt{r_i - |CH(R_i)|} - 2 \geq \sqrt{r_i - |CH(R_i)|} - 2$ empty monochromatic triangles.

Now we show that $|CH(R_i)| < 2\tilde{n}^{1/4}$. Assume to the contrary that $|CH(R_i)| \geq 2\tilde{n}^{1/4}$ and consider the set $(R_i \setminus CH(R_i)) \cup (B_i \setminus B'_i)$. This set has at most $r_i - 2\tilde{n}^{1/4}$ red points and $b_i \geq r_i - \tilde{n}^{1/4}$ blue points, so the difference is at least $\tilde{n}^{1/4}$ and as above we apply the Discrepancy Lemma and terminate.

Thus, if we don't terminate, we get at least $\sqrt{\tilde{n} - 2\tilde{n}^{1/4}} - 2 \geq \frac{\sqrt{\tilde{n}}}{2}$ empty monochromatic triangles in step i. Note that the last inequality holds for sufficiently large $\tilde{n}$.

For the next iteration step let $R_{i+1} = R_i \setminus CH(R_i)$, $B_{i+1} = B_i \setminus B'_i$, and $S_{i+1} = R_{i+1} \cup B_{i+1}$. Note that all the empty monochromatic triangles we have constructed in step i had at least one vertex in $CH(R_i)$, that is, we will not use these vertices for the next iterations, and therefore we do not overcount.

The process ends either by applying the Discrepancy Lemma or after $\frac{1}{3}n^{3/4}$ steps. As in each step we obtain at least $\frac{\sqrt{\tilde{n}}}{2}$ empty monochromatic triangles, we get in both cases a total of $\Omega(n^{5/4})$ empty monochromatic triangles.

It remains to show that the invariant $|S_i| \geq 2\tilde{n}$ holds. In step i we remove $|B'_i| + |CH(R_i)| < \tilde{n}^{1/4} + 2\tilde{n}^{1/4} = 3\tilde{n}^{1/4}$ points. Thus after $\frac{1}{3}n^{3/4}$ steps we have at least $n - \frac{1}{3}n^{3/4} \cdot 3\tilde{n}^{1/4} \geq 2\tilde{n}$ points left. □

Note that the constants in the above proof could be improved, but it is easy to see that the asymptotic behavior is tight within this approach.

3 Conclusions and Open Problems

We have not been able to construct a point set with $o(n^2)$ empty monochromatic triangles. Usually Horton sets are a good candidate to provide minimal examples with respect to determining empty convex polygons. But it turns out that every two-coloring of a Horton set has $\Omega(n^2)$ empty monochromatic triangles. A brief sketch of that fact looks as follows. Take any bi-coloring of the Horton set and note that the upper and lower part must have a linear number of red and blue points, as otherwise by the Discrepancy Lemma there would be

a quadratic number of empty monochromatic triangles. Now take any triangle of three consecutive points in the upper part which form a cap. Any edge of this triangle, where at least one is monochromatic, say red, and any point from the lower part spans an empty triangle. Thus, together with the $\Theta(n)$ red points from below, it forms a linear number of empty red triangles. Since there is a linear number of such caps we get a quadratic number of empty monochromatic triangles for the Horton set.

Other interesting sets with $O(n^2)$ empty triangles, which are not based on Horton sets, can be found in the constructions of Katchalski and Meir [8].

Considering results in [2] and [10] one can see that in the uncolored case and for sufficiently large n there is always a quadratic number of empty triangles and empty convex quadrilaterals, there is at least a linear number of pentagons but the correct bound seems to be quadratic. The status for convex empty hexagons was a long-standing open problem and it has recently been shown that at least one (and thus a linear number) exists, but the best upper bound is again quadratic. Finally no empty convex 7-gons may exist. So it seems that either none, or a quadratic number of empty k-gons exists, and we believe that somehow this translates to colored point sets. We therefore state the following conjecture.

Conjecture 1 *Any bi-colored set of n points in $\mathcal{R}^2$ in general position determines a quadratic number of empty monochromatic triangles.*

In fact we did not obtain a single family of sets where the asymptotics of the number of empty triangles and empty monochromatic triangles differ. What we have been able to construct are sets which have 5 times fewer empty monochromatic triangles than empty triangles. The idea behind the construction is to start with a set S of n points with $t(S)$ empty triangles. W.l.o.g. S has no two points on a horizontal line. We then add a copy of S which is shifted horizontally to the right by some sufficiently small ε and color the points of S red and their duplicates blue. For each ε-near pair we get $2n-2$ empty bi-chromatic triangles. For each empty triangle in S we get 3 new bi-chromatic triangles (not using an ε-near pair of points), but only one empty monochromatic triangle. Thus the ratio of empty triangles to monochromatic ones is $4+\frac{2n^2-2n}{t(S)}$. Taking the sets constructed by Bárány and Valtr [2] with $t(S) = 1.62n^2$ empty triangles gives a factor of ≈ 5.23.

Another interesting question is to consider empty monochromatic convex k-gons for $k > 3$. Devillers et al. [4] (Theorem 3.4) showed that for $k \geq 5$ and any n there are bi-colored sets where no empty monochromatic convex k-gon exists. So the remaining case are empty monochromatic convex quadrilaterals. For example in [4] they showed that for $n \geq 64$ any bi-colored Horton set contains empty monochromatic convex quadrilaterals. This leads to Conjecture 3.1 in [4] which states that for sufficiently large n any bi-colored set contains at least one monochromatic convex quadrilateral.

We recently learned that based on our approach the lower bound on the number of empty monochromatic triangles can be slightly improved [9].

Let us finally mention that we have been able to prove an analogous lower bound on the number of empty monochromatic simplices in R^d. We leave the details of this extension for the full version of this paper.

4 Acknowledgment

We would like to thank Wolfgang Aigner, Sergio Cabello, Bernhard Kornberger and Birgit Vogtenhuber for helpful discussions.

References

[1] I. Bárány, Z. Füredi, Empty simplices in Euclidean space. *Canad. Math. Bull.*, 30:436–445, 1987.

[2] I. Bárány, P. Valtr, Planar point sets with a small number of empty convex polygons. *Studia Scientiarum Mathematicarum Hungarica*, 41(2):243-269, 2004.

[3] R.P. Dilworth, A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[4] O. Devillers, F. Hurtado, G. Károlyi, C. Seara, Chromatic variants of the Erdös-Szekeres Theorem. *Computational Geometry, Theory and Applications*, 26(3):193–208, 2003.

[5] A. Dumitrescu, Planar sets with few empty convex polygons. *Studia Scientiarum Mathematicarum Hungarica*, 36(1-2):93-109, 2000.

[6] P. Erdös, R.K. Guy, Crossing number problems. *Amer. Math. Monthly*, 88:52–58, 1973.

[7] J.D. Horton, Sets with no empty convex 7-gons. *Canad. Math. Bull.*, 26:482-484, 1983.

[8] M. Katchalski, A. Meir, On empty triangles determined by points in the plane. *Acta Math. Hung.*, 51(3-4):323–328, 1988.

[9] J. Pach, G. Toth, personal communication 2008.

[10] R. Pinchasi, R. Radoicic, M. Sharir, On empty convex polygons in a planar point set. *J. Comb. Theory, Ser. A*, 113(3):385–419, 2006.

[11] J. Urrutia, Coloraciones, tetraedralizaciones, y tetraedros vacios en coloraciones de conjuntos de puntos en R^3. *Proc. X Encuentros de geometria Computacional*, Sevilla, pp 95–100, 2003.

[12] P. Valtr, On the minimum number of empty polygons in planar point sets. *Studia. Sci. Math. Hungar.* 30:155–163, 1995.

Draining a Polygon –or– Rolling a Ball out of a Polygon

Greg Aloupis* Jean Cardinal* Sébastien Collette†* Ferran Hurtado‡ Stefan Langerman§*
Joseph O'Rourke¶

Abstract

We introduce the problem of draining water (or balls representing water drops) out of a punctured polygon (or a polyhedron) by rotating the shape. For 2D polygons, we obtain combinatorial bounds on the number of holes needed, both for arbitrary polygons and for special classes of polygons. We detail an $O(n^2 \log n)$ algorithm that finds the minimum number of holes needed for a given polygon, and argue that the complexity remains polynomial for polyhedra in 3D. We make a start at characterizing the *1-drainable* shapes, those that only need one hole.

1 Introduction

Imagine a closed polyhedral container P partially filled with water. How many surface point-holes are needed to entirely drain it under the action of gentle rotations of P? It may seem that one hole suffices, but we will show that in fact sometimes $\Omega(n)$ holes are needed for a polyhedron of n vertices. Our focus is on variants of this problem in 2D, with a brief foray in Sec. 5 into 3D. We address the relationship between our problem and injection-filling of polyhedral molds [BvKT98] in Sec. 4.

A second physical model aids the intuition. Let P be a 2D polygon containing a single small ball. Again the question is: How many holes are needed to ensure that the ball, regardless of its initial placement, will escape to the exterior under gentle rotation of P? Here the ball is akin to a single drop of water. We will favor the ball analogy, without forgetting the water analogy.

Models. We consider two models, the (gentle) *Rotation* and the *Tilt* models. In the first, P lies in a vertical xy-plane, and gravity points in the $-y$ direction. The ball B sits initially at some convex vertex v_i; vertices are labeled counterclockwise (ccw). Let us assume that v_i is a local minimum with respect to y, i.e., both v_{i-1} and v_{i+1} are above v_i. Now we are permitted to rotate P in the vertical plane (or equivalently, alter the gravity vector). In the Rotation model, B does not move from v_i until one of the two adjacent edges, say $e_i = v_i v_{i+1}$, turns infinitesimally beyond the horizontal, at which time B rolls down e_i and falls under the influence of gravity until it settles at some other convex vertex v_j. For example, in Fig. 1, B at v_4 rolls ccw

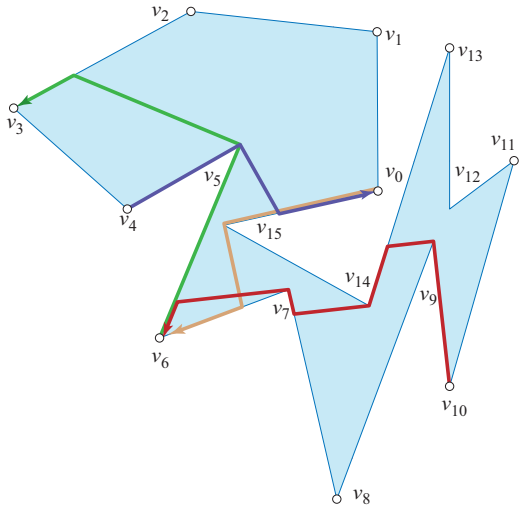


Figure 1: Polygon with several ball paths.

when $v_4 v_5$ is horizontal, falls to edge $v_{15} v_0$, and comes to rest at v_0. Similarly, B at v_{10} rolls clockwise (cw) to v_6 after three falls. Note that all falls are parallel, and (arbitrarily close to) orthogonal to the initiating edge (in the Rotation model). After B falls to an edge, it rolls to the endpoint on the obtuse side of its fall path.

The only difference in the Tilt model is that any gravity vector may be selected. Only vectors between e_{i-1} and e_i will initiate a departure of B from v_i, i.e., the entire wedge is available rather than just the two incident edges. For example, in Fig. 1, B at v_4 rolls to $\{v_0, v_3\}$ in the Rotation model, but can roll to $\{v_0, v_1, v_2, v_3\}$ in the Tilt model. The Rotation model more accurately represents physical reality, for rain drops or for balls. The Tilt model mimics various ball-rolling games

*[greg.aloupis,jcardin,secollet,slanger]@ulb.ac.be Université Libre de Bruxelles (ULB), CP212, Bld. du Triomphe, 1050 Brussels, Belgium.
†Chargé de Recherches du FRS-FNRS.
‡ferran.hurtado@upc.edu Universitat Politècnica de Catalunya, Jordi Girona 1–3, E-08034 Barcelona, Spain.
§Chercheur Qualifié du FRS-FNRS.
¶orourke@cs.smith.edu Smith College, Northampton, MA 01063, USA.

(e.g., Labyrinth) that permit quickly "tilting" the polygon/maze from the horizontal so that any departure vector from v_i can be achieved. We emphasize that, aside from this departure difference, the models are identical. In particular, inertia is ignored, and rotation while the ball is "in-flight" is forbidden (otherwise we could direct B along any path).

There are two "degenerate" situations that can occur. If B falls exactly orthogonal to an edge e, we arbitrarily say it rolls to the cw endpoint of e. If B falls directly on a vertex, both of whose edges angle down with respect to gravity, we stipulate that it rolls to the cw side.

Questions. Given P, what is the minimum number of point-holes needed to guarantee that any ball, regardless of starting position, may eventually escape from P under some sequence of rotations/tilts? Our main result is that this number can be determined in $O(n^2 \log n)$ time. In terms of combinatorial bounds, we show that some polygons require $\lfloor n/6 \rfloor$ and $\lfloor n/7 \rfloor$ holes (in the Rotation/Tilt models respectively), but $\lceil n/4 \rceil$ holes always suffice. We make a start at characterizing the *1-drainable* polygons, those that only need one hole Finally we argue that the minimum number of holes can be computed for a 3D polyhedron in polynomial time. (Omitted proofs may be found in the full version.)

2 Traps

We start by exhibiting polygons that need $\Omega(n)$ holes to drain. The basic idea is shown in Fig. 2(a) for the Rotation model. We create *traps* with 5 vertices forming

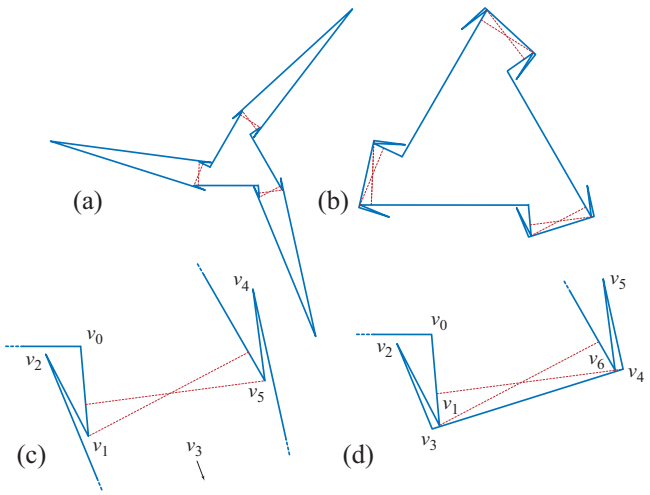


Figure 2: (a) Trap for Rotation model. (b) Trap for Tilt model. (c,d) Details of traps.

an "arrow" shape, connected together around a convex polygonal core so that 6 vertices are needed per trap. A ball in v_4 rolls to fall on edge v_0v_1, but because of the slightly obtuse angle of incidence, rolls to v_2; and symmetrically, v_2 leads to v_4. So there is a cycle (defined precisely in Sec. 3) that "traps" ball between $\{v_2, v_3, v_4\}$ and isolates it from the other two traps. Therefore three holes are required to drain this polygon. In the Tilt model, Fig. 2(a) only needs one hole, because B could roll directly from v_3 through the $v_1 - v_5$ "gap." However, the polygon in Fig. 2(b) requires 3 holes. Here the range of effective gravity tilt vectors from v_4 is so narrow that the previous analysis holds. These examples establish the necessity half of this theorem:

Theorem 1 (Combinatorial Bounds) *In the Rotation (resp. Tilt) model, $\lfloor n/6 \rfloor$ (resp. $\lfloor n/7 \rfloor$) holes are sometimes necessary to drain an n-vertex polygon. $\lceil n/4 \rceil$ holes suffice to drain any polygon.*

Although we believe that at least two reflex vertices are needed in every cycle, we were unable to show that they could not be shared between traps. We nevertheless conjecture that $\lceil n/5 \rceil$ holes suffice. (A variation on Fig. 2(a) permits the formation of two traps with $n = 11$.)

Proposition 2 *$\lfloor n/28 \rfloor$ holes are sometimes necessary to drain an n-vertex orthogonal polygon, and $\lceil n/8 \rceil$ holes suffice.*

3 The Pin-Ball Graph

Let G be a directed graph whose nodes are the convex vertices of P, with v_i connected to v_j if B can roll in one "move" from v_i to v_j. Here, a move is a complete path to the local y-minimum v_j, for some fixed orientation of P. We conceptually label the arcs of G with the sequence of vertices and edges along the path $\rho(v_i, v_j)$. Thus, the (v_{10}, v_6) arc in Fig. 1 is labeled $(v_9, e_{13}, v_{14}, e_7, v_7, e_5)$. We use G_R and G_T to distinguish the graphs for the Rotation and Tilt models respectively, and G when the distinction is irrelevant.

We gather together a number of basic properties of G in the following lemma.

Lemma 3 (G Properties)

1. *Every node of G_R has out-degree 2; a node of G_T has out-degree at least 2 and at most $O(n)$.*
2. *Both G_R and G_T have $O(n)$ nodes (one per convex vertex). G_R has at most $2n$ arcs, while G_T has $O(n^2)$ arcs, and sometimes $\Omega(n^2)$ arcs.*
3. *Each path label has length $O(n)$ (in either model).*
4. *The total number of path labels on the arcs of G_R is $O(n^2)$, and sometimes $\Omega(n^2)$.*
5. *The total number of labels in G_T is $O(n^3)$, and sometimes $\Omega(n^3)$.*

We will see below that G_T can be constructed more efficiently than what the cubic total label size in Lemma 3(5) might indicate.

Noncrossing Paths. A ball path corresponding to one arc of G is a polygonal curve, monotone with respect to gravity $\overrightarrow{g}$. The path is composed of subsegments of polygon edges, as well as *fall* segments, each of which is parallel to $\overrightarrow{g}$ and incident to a reflex vertex. A directed path ρ naturally divides P into a "left half" $L = L(\rho)$ of points left of the traveling direction, and a "right half" $R = R(\rho)$, where L and R are disjoint, and $L \cup R \cup \rho = P$. Two ball paths ρ_1 and ρ_2 (properly) *cross* if ρ_2 contains points in both $L(\rho_1)$ and $R(\rho_1)$. For example, in Fig. 1, $\rho(v_0, v_6)$ crosses $\rho(v_{10}, v_6)$. Let $\overline{L} = L(\rho) \cup \rho$ be the closure of $L(\rho)$, and similarly define $\overline{R}$.

Two paths can only cross at a reflex vertex (as do $\rho(v_4, v_0)$ and $\rho(v_6, v_3)$ in Fig. 1) or on fall segments of each (as do $\rho(v_0, v_6)$ and $\rho(v_{10}, v_6)$).

Lemma 4 (Noncrossing) *Two paths ρ_1 and ρ_2 from the same source vertex v_0 never properly cross (in either model).* See Fig. 3.

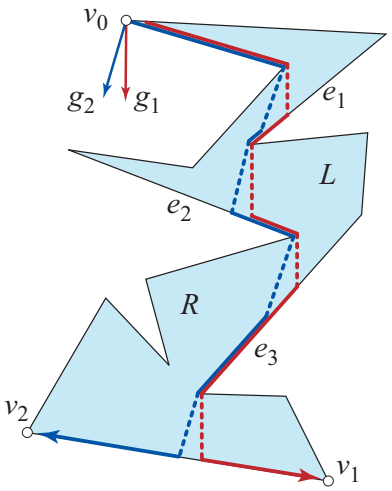


Figure 3: Paths from the same source v_0 do not cross. Fall segments are dashed. L and R indicate polygon "halves" left and right of the directed paths.

Lemma 5 (Label Intervals) *In the Tilt model, a particular label λ appears on the arcs of G_T originating at one particular v_i within an interval $[\overrightarrow{g_1}, \overrightarrow{g_2}]$ of gravity directions.*

Cycles and Strongly Connected Components. The directed path from any vertex v_i of G leads to a cycle in G, because every node has at least two outgoing edges by Lemma 3(1). Any maximal cycle in G has length at least 3. Anything less would involve a pair (v_i, v_j) connecting only to each other, which would contradict Lemma 3(1). Note that any pair of convex vertices adjacent on ∂P form a non-maximal cycle of length 2.

A cycle is a particular instance of a *strongly connected component* (SCC) of G, a maximal subset $C \subset G$ in which each node has a directed path to all others.

Define a graph G^* as follows. Let $C_1, C_2, \ldots$ be the SCC's of G. Contract each C_k to a node c_k of G^*, while otherwise maintaining the connectivity of G. Then G^* is a DAG (because all cycles have been contracted).

Lemma 6 (Sinks) *The minimum number m of holes needed to drain P is the number of sinks of G^*.*

Lemma 7 *The locations of the minimum number m of holes needed to drain P can be found in linear time in the size $|G|$ of G, once G has been constructed.*

Construction of G. Our goal is to construct the unlabeled G. Labels merely represent the paths that realize each arc of G. An example given in the long version seems to require $\Omega(n^2)$ ray-shooting queries in the Rotation model, and as we do not know how to avoid this, our goal becomes an $O(n^2 \log n)$ algorithm. This is straightforward for G_R, so we focus on G_T, which by Lemma 3(5) is potentially cubic.

We first preprocess P for efficient ray-shooting queries, using fractional cascading to support ray shooting in a polygonal chain. This takes $O(n \log n)$ preprocessing time and supports $O(\log n)$ time per query ray [CEG+94]. Next we construct the visibility polygon from each vertex of in overall $O(n^2)$ time [JS87]. From these visibility polygons, for each v_i we construct a *gravity diagram* D_i. This partitions all gravity vectors $\overrightarrow{g}$ into angular intervals labeled with the next vertex that B will roll to from v_i with tilt $\overrightarrow{g}$. For example, Fig. 4(a) shows the gravity diagram for v_4 in Fig. 1. Note that

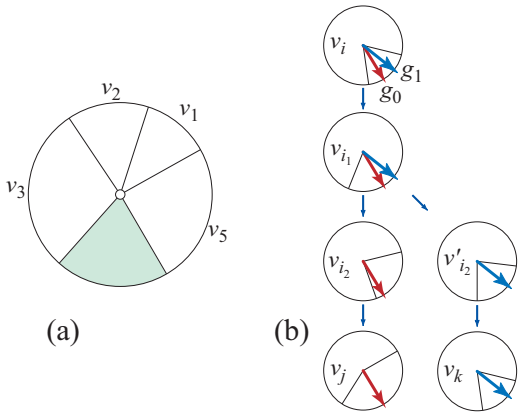


Figure 4: (a) Gravity diagram for v_4 in Fig. 1. (b) Gravity diagrams for paths ρ_0 and ρ_1.

D_i only records the next vertex encountered, not the ultimate destination. We maintain each diagram in a structure that permits any $\overrightarrow{g}$ to be located in $O(\log n)$ time.

We now argue that we can construct all paths with source v_i, and therefore all arcs of G_T leaving v_i, in

$O(n \log n)$ time. Compute the path ρ_0 for the cw extreme gravity vector $\overrightarrow{g_0}$ that leaves v_i (perpendicular to $v_i v_{i+1}$). This uses $O(n)$ ray-shooting queries for the fall segments of ρ_0, totaling $O(n \log n)$ time. Let $\rho_0 = (v_i, v_{i_1}, v_{i_2}, \ldots, v_j)$. During its construction, we locate $\overrightarrow{g_0}$ within each diagram D_{i_k}. Now we find the minimum angle between $\overrightarrow{g_0}$ and the next ccw event over all diagrams. This can be done in $O(\log n)$ time using a priority queue. Call the next event $\overrightarrow{g_1}$, and suppose it occurs at v_{i_k} in diagram D_{i_k}. We now construct the path ρ_1 from v_{i_k} onward, until it terminates at a new vertex, or rejoins ρ_0 (recall from Fig. 3 that paths might rejoin, i.e., the suffixes from v_{i_k} are not necessarily disjoint). In our "update" from ρ_0 to ρ_1, let V_0 be the set of vertices lost from ρ_0, and V_1 those gained in ρ_1. The priority queue of minima is updated by deleting those for V_0 and inserting those for V_1. The angular sweep about v_i continues in the same manner until the full gravity vector range about v_i is exhausted. Fig. 4(b) illustrates one step of this process, where v_{i_1} determines the transition event between g_0 and g_1, at which point the path changes from $\rho_0 = (v_i, v_{i_1}, v_{i_2}, v_j)$ to $\rho_1 = (v_i, v_{i_1}, v'_{i_2}, v_k)$.

By Lemma 5, each diagram abandoned in this sweep is never revisited. Thus the number of invocations of the minimum operation to find the next event is $O(n)$, or $O(n \log n)$ overall. Repeating for each v_i we obtain:

Lemma 8 *G can be constructed in $O(n^2 \log n)$ time.*

Theorem 9 *The locations of the minimum number of holes needed to drain P can be found in $O(n^2 \log n)$.*

We leave it as a claim that G_R can be constructed (and the holes located) in $O(n \log n)$ time for orthogonal polygons.

4 1-Drainable Shapes

Define a *k-drainable* polygon as one that can be drained with k holes but not with $k-1$ holes. For example, Fig. 1 is 1-drainable with a hole at v_6. We make a start here at exploring the 1-drainable shapes under each model. Note that these shapes do depend on the model: Fig. 2(a) is 1-drainable in the Tilt model but 3-drainable in the Rotation model.

Our definition of k-drainable polygons is inspired by the k-fillable polygons of [BvKT98], those mold shapes that can be filled with liquid metal poured into k holes. Despite the apparent inverse relationship between filling and draining, the two concepts are rather different. In particular, there are star-shaped polygons k-drainable in the rotation model (Proposition 13 below), but Theorem 7.2 of [BvKT98] shows that these are all "2-fillable with re-orientation." Also, there are 1-drainable polygons that are k-fillable (with or without reorientation).

Proposition 10 *Monotone polygons are 1-drainable.*

Let the *ccw roll* from v_i be the roll toward v_{i+1} in the Rotation model, or equivalently, the tilt according to $\overrightarrow{g}$ perpendicular to $v_i v_{i+1}$ in the Tilt model.

Lemma 11 (Kernel) *Let P be star-shaped with kernel K. Then for each arc $(v_i, v_j) \in G$ corresponding to the ccw roll path ρ from v_i, K is in $\overline{L(\rho)}$, i.e., K is on or to the left of ρ.*

A *fan* is a star-shaped polygon whose kernel includes a convex vertex.

Proposition 12 *Fans are 1-drainable.*

Proposition 12 cannot be extended to star-shaped polygons in the Rotation model:

Proposition 13 *For any $k > 1$, there is a k-drainable star-shaped n-gon in the Rotation model, with $k = \Omega(n)$.*

5 3D

Define the Tilt model for 3D polyhedra to permit departure from a vertex v at a direction vector lying in any of the faces of P incident to v. We do not see how to mimic the efficient construction of G previously described, so we content ourselves with showing (in the full version) that it can be accomplished in polynomial time: $O(n^7 \log n)$.

6 Open Problems

1. Can the upper bound of $\lceil n/4 \rceil$ in Theorem 1 be improved?
2. Are star-shaped polygons 1-drainable in the Tilt model? More generally, characterize 1-drainable polygons.
3. Suppose m balls are present in P at the start, and P is k-drainable. What is the computational complexity of finding an optimal schedule of rotations, say, in terms of the total absolute angle turn, or in terms of the number of angular reversals?

References

[BvKT98] P. Bose, M. van Kreveld, and Godfried T. Toussaint. Filling polyhedral molds. *Comput. Aided Design*, 30(4):245–254, April 1998.

[CEG+94] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[JS87] B. Joe and R. B. Simpson. Correction to Lee's visibility polygon algorithm. *BIT*, 27:458–473, 1987.

Partial Matching of Planar Polygons Under Translation and Rotation

Eric C. McCreath*

Abstract

Curve matching is an important computational task for domains such as: reconstruction of archaeological fragments, forensics investigation, measuring melodic similarity, and model-based object recognition. There are a variety of measures and algorithmic approaches used to address the curve matching problem including: shape signature strings with substring matching, geometric hashing, and Hausdorff distance approaches. In this paper we propose an approach that uses a turning function representation of the shape and also uses a L_2 measure for comparing matches. The novel algorithm presented finds the best match along a fixed length portion of two polygon's perimeters where the polygons may be arbitrarily translated and rotated. The algorithm's time complexity is $O(mn(n+m))$ where n and m are the numbers of vertices in the perimeters being matched. The utility of the algorithm is demonstrated in the reconstruction of a small jigsaw puzzle.

1 Introduction

Reconstruction of a broken object is an important yet time consuming task for a number of disciplines including forensics and archeology. Digitally automating or semi-automating this process is beneficial. Jigsaw puzzles, which are a simplistic form of this reconstruction problem, have been investigated by a number of researchers over the last 50 years. Freeman and Garder[6] produced what is generally considered the first of these investigations. They matched portions of a piece by comparing features extracted from the shape of those portions. There has since been a variety of other approaches in solving this problem including: curve matching combinatorial optimization[11], use of critical points[10, 8, 7], shape and image matching[12], and even an attempt to reconstruct the puzzle via a robot [4]. In many respects the jigsaw puzzle problem is a much simpler problem than the more general reconstruction of a broken fragment due to the well defined constraints on the shape of jigsaw puzzles, though it is not a simple problem to solve.

This paper proposes a novel algorithm which takes two polygons and matches a fixed length portion of the polygons perimeter. The polygons may be arbitrarily translated and rotated, however they are not scaled. This algorithm finds the match which minimizes the L_2 distance of the turning functions of the two portions of the polygons. The novelty of the approach taken in this paper is how a fixed length portion of two unscaled polygons can be matched. Such matching is useful for reconstruction when fragments are matched against the complement of other fragments.

Arkin et al. [3] proposed using the L_2 distance between turning functions of polygons to compare two shapes. Their algorithm works in $O(mn \log mn)$ time where n and m are the numbers of vertices in the polygons. The Arkin et al. approach is different to the contribution made in this paper as they find matches between two entire polygons which have both been rescaled to have a perimeter length of 1.

Cohen and Guibas [5] developed an algorithm that matches a polyline by translation, rotation, and scaling to a part of another polyline. Their algorithm works in $O(m^2n^2)$ time where m and n are the numbers of edges within the polylines. The approach taken in this paper is different to the Cohen and Guibas approach, as Cohen and Guibas finds the shifting and stretching parameters that minimize a combination of L_2 distance of the turning functions and match length. Whereas, in this paper, the approach presented finds the two shifting parameters which determine where the matching portions of the polygons will start.

Aloupis et al. [1, 2] developed an approach that finds the minimum area between two given orthogonal melodies with periods of 2π. Their approach runs in $O(n^2 \log n)$ time and can be used for matching short patterns in a database of music. This is the same problem of matching polygons when a turning function representation is used. The problem Aloupis et al. address is different to the problem this paper addresses as Aloupis et al. use an L_1 distance and they focus on comparing either two cyclic melodies (parallels with Arkin et al.) or a melody which matches a portion of another melody. Whereas, this paper uses the L_2 distance and focuses on fixed length portions as these portions could occur anywhere along the x-axis of the two turning functions. If the approach presented in this paper was applied to the music domain, then the approach could be used to find common melodies of fixed length which occur anywhere within two items of music.

*Department of Computer Science, The Australian National University, ACT 0200 Australia ericm@cs.anu.edu.au

2 Problem Setup

Let A and B denote two planar polygons with n and m vertices respectively. The vertices of A are points in the xy-plane denoted $\{\dot{a}_0, \dot{a}_1, \ldots, \dot{a}_{n-1}\}$. Vertex $\dot{a}_i$ is connected to vertex $\dot{a}_{i+1}$ by an edge. Also vertex $\dot{a}_{n-1}$ is connected to vertex $\dot{a}_0$ by an edge. To simplify the wrapping of the polygon we also define $\dot{a}_{i+n} = \dot{a}_i,\ \forall i \geq 0$. Let $\bar{a}_i$ denote the vector from $\dot{a}_i$ to $\dot{a}_{i+1}$, this vector provides the direction and length of edge i and may be calculated by $\bar{a}_i = \dot{a}_{i+1} - \dot{a}_i$. So $|\bar{a}_i|$ is the length of edge i. Let a_i denote the distance from $\dot{a}_0$ to $\dot{a}_i$ following the perimeter of the polygon. More formally $a_i = \sum_{j=0}^{i-1} |\bar{a}_j|$. Note that $a_0 = 0$ and a_n is the length of the perimeter. Let the turning function $t_A(d)$ be the accumulative turning angle at distance d around the perimeter of A from $\dot{a}_0$. $\dot{b}_i$, $\bar{b}_i$, b_i, and $t_B(d)$ are defined for polygon B in a similar way to that of polygon A.

Figure 1 shows an example of a simple polygon and its turning function representation. We wish to determine how well portions of one polygon will fit together with that of another. The turning function provides an efficient way of determining if polylines closely follow each other[3]. This efficiency is due to the translation invariant nature of the representation. Also, finding the best rotation of one polygon onto the other can be analytically determined without explicitly searching this dimension.

An L_2 distance is used over a fixed length l of the perimeter to determine the error in matching a particular configuration. Given this fixed perimeter length we must find the minimum error over a 3 dimensional space, where the dimensions are: the starting location s_A of the matching on the perimeter of polygon A; the starting location s_B on polygon B; and the angle of rotation θ. The start location s_A (and s_B) is the distance around the perimeter from a_0 (and b_0). Thus the error we wish to minimize over s_A, s_B, and θ is:

$$\text{error}(s_A, s_B, \theta) = \int_0^l (t_A(s_A + x) - t_B(s_B + x) + \theta)^2 dx$$

3 Searching

To search for the values that minimize the error, we first show how to calculate the θ that minimizes the error for a given s_A and s_B. We denote this minimum angle with the function $\theta^*(s_A, s_B)$. This calculation is done in the same way as [3]. We set the partial derivative of $\text{error}(s_A, s_B, \theta)$ to zero finding the only critical point at:

$$\theta = \frac{-\int_0^l t_A(s_A + x) - t_B(s_B + x)dx}{l}$$

A second derivative test reveals that this is the minimum. Thus we set:

$$\theta^*(s_A, s_B) = \frac{-\int_0^l t_A(s_A + x) - t_B(s_B + x)dx}{l}$$

This enables us to reduce the degrees of freedom for the search down to 2 as:

$$\min\{\text{error}(s_A, s_B, \theta)\} = \min\{\text{error}(s_A, s_B, \theta^*(s_A, s_B))\}$$

Let:

$$\begin{aligned}
\text{error}^*(s_A, s_B) &= \text{error}(s_A, s_B, \theta^*(s_A, s_B)) \\
&= \int_0^l (t_A(s_A + x) - t_B(s_B + x) \\
&\quad - \tfrac{\int_0^l t_A(s_A+x) - t_B(s_B+x)dx}{l})^2 dx \\
&= \int_0^l (t_A(s_A + x) - t_B(s_B + x))^2 dx - \\
&\quad \tfrac{(\int_0^l t_A(s_A+x) - t_B(s_B+x)dx)^2}{l} \\
&= II(s_A, s_B) - \tfrac{1}{l} I(s_A, s_B)^2
\end{aligned}$$

where

$$I(s_A, s_B) = \int_0^l t_A(s_A + x) - t_B(s_B + x)dx$$

and

$$II(s_A, s_B) = \int_0^l (t_A(s_A + x) - t_B(s_B + x))^2 dx$$

Since both s_A and s_B are continuous values it is impossible to explicitly search all possibilities. However, this search space may be partitioned by lines into a number of regions. The lines are either when vertices of the two polygons line up or when the start or end of the matching region lines up with a vertice on a polygon. The minimum of the error function over each region can be found at the crossing points on the border of the region. Hence, the minimum over the entire search space can be found by considering all the points at which these lines intersect.

As the functions t_A and t_B are both piecewise constant, both $t_A(s_A + x) - t_B(s_B + x)$ and $(t_A(s_A + x) - t_B(s_B + x))^2$ will also be piecewise constant functions in the variable x. Hence to calculate $I(s_A, s_B)$ and $II(s_A, a_B)$ one can simply sum the length of the contribution of each of the constant sections multiplied by the value for that section. We let $X_{ij}(s_A, s_B)$ be the length of the contribution made by the polygon sides $\bar{a}_i$ and $\bar{b}_j$. This can be calculated via:

$$\begin{aligned}
X_{ij}(s_A, s_B) = \ & |(\max\{0, a_i - s_A, b_j - s_B\}, \\
& \min\{l, a_{i+1} - s_A, b_{j+1} - s_B\})|
\end{aligned}$$

where $|(x, y)| = \max\{y - x, 0\}$. We also let $\theta_{ij} = t_A(a_i) - t_B(b_j)$. So we now have:

$$I(s_A, s_B) = \sum_{ij} X_{ij}(s_A, s_B)\theta_{ij}$$

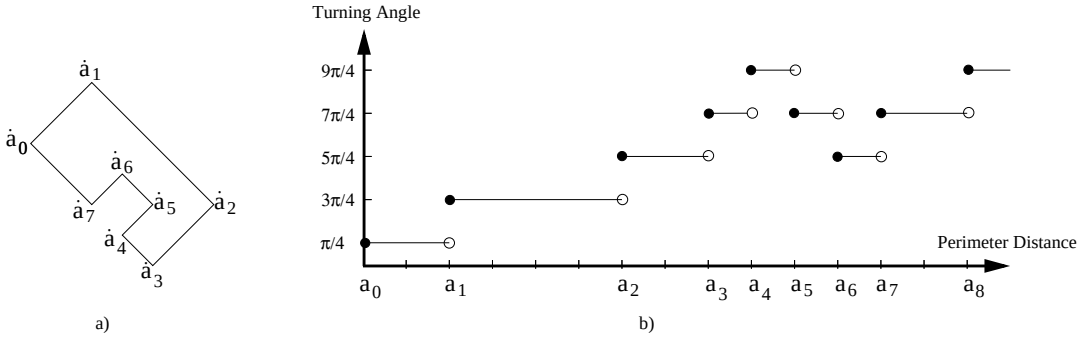


Figure 1: a) A simple polygon. b) The turning function for the polygon shown in a).

and

$$II(s_A, s_B) = \sum_{ij} X_{ij}(s_A, s_B)(\theta_{ij}^2)$$

The X_{ij} can be rewritten as a linear function in s_A and s_B for 10 different regions of the s_A, s_B plane. This can be done by considering: the three different possible maximums with the three possible minimums; along with the no overlapping possibility.

$$X_{ij}(s_A, s_B) = \begin{cases} l & \text{if } 0 \geq a_i - s_A \wedge \\ & 0 \geq b_j - s_B \wedge \\ & l \leq a_{i+1} - s_A \wedge \\ & l \leq b_{j+1} - s_B \\ a_{i+1} - s_A & \text{if } 0 \geq a_i - s_A \wedge \\ & 0 \geq b_j - s_B \wedge \\ & a_{i+1} - s_A \leq b_{j+1} - s_B \wedge \\ & a_{i+1} - s_A \leq l \wedge \\ & a_{i+1} - s_A \geq 0 \\ \vdots & \vdots \\ 0 & \text{otherwise} \end{cases}$$

The s_A,s_B plane can be divide up into regions via the following lines: $0 = a_i - s_A$, $0 = b_j - s_B$, $a_i - s_A = b_j - s_B$, $l = a_i - s_A$, and $l = b_j - s_B$. Within each of these regions X_{ij} will be linear with respect to s_A and s_B. Moreover, within each of these regions both $I(s_A, s_B)$ and $II(s_A, s_B)$ will be linear with respect to s_A and s_B. For each region r, let c_1, c_2, c_3, c_4, c_5, and c_6, be the constants such that:

$$I_r(s_A, s_B) = c_1 s_A + c_2 s_B + c_3$$

and

$$II_r(s_A, s_B) = c_4 s_A + c_5 s_B + c_6$$

then

$$\begin{aligned} \text{error}_r^*(s_A, s_B) &= II_r(s_A, s_B) - \tfrac{1}{l} I_r(s_A, s_B)^2 \\ &= -\tfrac{c_1^2}{l} s_A^2 - \tfrac{c_2^2}{l} s_B^2 - \tfrac{2c_1c_2}{l} s_A s_B + \\ &\quad (c_4 - \tfrac{2c_1c_3}{l}) s_A + (c_5 - \tfrac{2c_2c_3}{l}) s_B + \\ &\quad c_6 - \tfrac{c_3^2}{l} \end{aligned}$$

It is simple to confirm that the minimum of this function will be at one of the vertices of the region. Therefore, to find the minimum of this function over the entire s_A, s_B plane, one may simply find the minimum over all the points at which the lines cross. Fortunately, we do not need to recalculate I and II for every point, as we can move from one crossing point to a neighboring crossing point and evaluate the new error from information from the previous point in a constant amount of time.

4 The Algorithm

The algorithm works by calculating the error on each of the crossing points between the lines over the entire plane. There are at most[1] nm sloping lines $a_i - s_A = b_j - s_B$. These are all parallel with each other and have a gradient of -1. Also, there are at most $2n$ horizontal lines $0 = a_i - s_A$ or $l = a_i - s_A$. Finally, there are at most $2m$ vertical lines $0 = b_j - s_B$ or $l = b_j - s_B$. The sloping lines will intersect with both the vertical and horizontal lines. To calculate the minimum over all these intersecting points we consider each of the sloping lines in turn. We begin at any point on a line and calculate $I(s_A, s_B)$ and $II(s_A, s_B)$. This may be accomplished in $n+m$ steps by moving across the turning functions of A and B summing contributions to the integrals. Once this is calculated it is possible to slide along this sloping line to the next point where a vertical or horizontal line intersects with it. We denote this new point (s'_A, s'_B). $I(s'_A, s'_B)$ and $II(s'_A, s'_B)$ can be calculated using $I(s_A, s_B)$ and $II(s_A, s_B)$ and subtracting the contributions that no longer overlap and adding the new overlapping contributions. This may be achieved in constant time. Therefore, finding the minimum over all points which intersect with the sloping lines is $O(mn(m+n))$. The other points that must be considered occur when horizontal and vertical lines intersect. There are at most $4nm$ of these. The error for

[1] There could be fewer lines if a number of combinations of i and j produce the same line.

each of these can be calculated separately in at most $n + m$ steps. Hence, the complexity of calculating the points of intersection for both the horizontal and vertical lines is $O(mn(n+m))$. Moreover, the time complexity of calculating the configuration that minimizes error is $O(mn(n + m))$. Note that the space required for this algorithm is only $O(n + m)$.

5 Discussion

The matching algorithm was implemented and used within a puzzle solving program to demonstrate the utility of the matching algorithm. The data sets consisted of a simple 20 piece puzzle. The puzzle solving program used a greedy approach. The minimum error is found between each fragment and the complement of another fragment using the algorithm presented in this paper. The fragments with the minimum error are removed from the set of fragments, then the fragments are joined forming a new fragment which is then incorporated back into the set of fragments. This process is repeated until all the fragments are joined into a single fragment. Clearly, this greedy approach is not guaranteed to produce either an optimal or correct solution. However, in the puzzle tested the greedy approach produced a correct result. Note that, the fixed matching length was manually tune for this particular puzzle.

A larger class of shapes can be more compactly and accurately represented by including circular arcs as edges. In such cases the turning function is piecewise linear. Arkin et al. [3] considered this for matching shapes. In a similar way the algorithm presented in this paper could be extended to include circular arcs. Such a representation would clearly perform well for shapes like the puzzle fragments.

In terms of improving the performance of the algorithm it would be possible to use an approach similar to that of Latecki et al. [9] where polygons undergo a curve evolution to approximate a polygon with fewer edges. This approximation would be within some known error of the turning function. This could be used to prune large sections of the search, as bounds could be found for particular regions of the search space. The optimal configuration could then be found on this restricted search space. Hence, the overall algorithm would produce the optimal result more quickly. In general it is unlikely that such a modification would improve the worst case complexity of the algorithm, however, it could improve the expected running time of the algorithm.

References

[1] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, Y. Nuñez, D. Rappaport, and G. Toussaint. Algorithms for computing geometric measures of melodic similarity. *Comput. Music J.*, 30(3):67–76, 2006.

[2] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, D. Rappaport, and G. Toussaint. Computing the similarity of two melodies. In *Proceedings of the 15th Canadian Conference on Computational Geometry (CCCG'03)*, pages 81–84, 2003.

[3] E. M. Arkin, P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–216, March 1991.

[4] G. Burdea and H. Wolfson. Solving jigsaw puzzles by a robot. *IEEE Transactions on Robotics and Automation*, 5(6):752–764, Dec 1989.

[5] S. D. Cohen and L. J. Guibas. Partial matching of planar polylines under similarity transformations. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[6] H. Freeman and L. Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-13(2):118–127, April 1964.

[7] D. Goldberg, C. Malon, and M. Bern. A global approach to automatic solution of jigsaw puzzles. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 82–87, New York, NY, USA, 2002. ACM Press.

[8] D. Kosiba, P. Devaux, S. Balasubramanian, T. Gandhi, and K. Kasturi. An automatic jigsaw puzzle solver. *Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, 1:616–618 vol.1, 9-13 Oct 1994.

[9] L. J. Latecki and R. Lakamper. Shape similarity measures based on correspondence of visual parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10), 2000.

[10] R. Webster, P. LaFollette, and R. Stafford. Isthmus critical points for solving jigsaw puzzles in computer vision. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1271–1278, Sep/Oct 1991.

[11] H. Wolfson, E. Schonberg, A. Kalvin, and Y. Lamdan. Solving jigsaw puzzles by computer. *Ann. Oper. Res.*, 12(1-4):51–64, 1988.

[12] F.-H. Yao and G.-F. Shao. A shape and image merging technique to solve jigsaw puzzles. *Pattern Recognition Letters*, 24:1819–1835, 2003.

Recognition of Largest Empty Orthoconvex Polygon in a Point Set

Subhas C. Nandy* Krishnendu Mukhopadhyaya* Bhargab B. Bhattacharya*

Abstract

An algorithm for computing the maximum area empty isothetic orthoconvex polygon among a set of n points in a rectangular region, is presented. The worst case time and space complexities of the proposed algorithm are $O(n^3)$ and $O(n^2)$ respectively.

1 Introduction

The problem of finding an empty convex k-gon of maximum area or perimeter amidst a point set [4] has several applications. A survey paper [2] has been published very recently, which elaborates several optimization issues related to this problem. In VLSI layout design, document image processing and shape description, isothetic polygons play a major role. A polygon is said to be *isothetic* if its sides are parallel to coordinate axes. The problem of identifying the largest empty isothetic rectangle among a set of points has been studied extensively. The best known algorithm for this problem runs in $O(n \log^2 n)$ time [1]. The same time complexity holds if the obstacles are arbitrary polygons [6, 9]. Recently it is shown that the largest isothetic rectangle inside a simple polygon can be obtained in $O(n \log n)$ time [5].

In isothetic domain, the generalization of this problem is recognizing the largest empty orthoconvex polygon. An isothetic polygon is said to be *orthoconvex* if the intersection of the polygon with a horizontal or a vertical line is a single line segment. Orthoconvexity has importance in robotic visibility, and also its discrete variant appears in other areas like digital geometry [3] and discrete tomography [8]. Datta and Ramkumar [7] proposed algorithms for recognizing largest empty orthoconvex polygon of some specified shapes amidst a 2D point set. These include (i) L-shape, (ii) cross shape, (iii) point visible, and (iv) edge visible polygons. The time complexity of these algorithms are all $O(n^2)$. Another variation in this class of problems is recognizing the largest empty staircase polygon among point and isothetic polygonal obstacles, which can also be solved in $O(n^2)$ time and space [10]. But, to the best of our knowledge, the problem of recognizing an empty orthoconvex polygon of arbitrary shape maximizing area/perimeter is not studied yet. In this paper, we propose an algorithm of recognizing an empty orthoconvex polygon of maximum area, that runs in $O(n^3)$ time using $O(n^2)$ space.

*Indian Statistical Institute, Kolkata - 700 108, India

2 Priliminaries

Let $\mathcal{R}$ be a rectangular region containing a set of n points $P = \{p_1, p_2, \ldots, p_n\}$. We will assume a coordinate system with bottom and left boundaries of $\mathcal{R}$ as the x- and y-axes respectively. The coordinates of a point α are denoted as $(x(\alpha), y(\alpha))$. We assume that the points in P are in general positions, i.e., for every two points p_i and p_j , $x(p_i) \neq x(p_j)$ and $y(p_i) \neq y(p_j)$. Henceforth, we shall use H_i and V_i to denote a horizontal and a vertical line passing through the point p_i.

Definition 1 *An isothetic curve is a rectilinear path consisting of alternately horizontal and vertical line segments. An isothetic curve is a monotonically rising staircase (R-stair) if for every pair of points α and β on the curve, $x(\alpha) \leq x(\beta)$ implies $y(\alpha) \leq y(\beta)$. Similarly, for every pair of points α and β on a monotonically falling staircase (F-stair), we have $x(\alpha) \leq x(\beta)$ implies $y(\alpha) \geq y(\beta)$. An isothetic polygon is a region bounded by a closed isothetic curve.*

Definition 2 *An isothetic polygon Π is said to be orthoconvex if for any horizontal or vertical line ℓ, the intersection of Π with ℓ is a line segment of length greater than or equal to 0. In other words, ℓ either intersects no edge or exactly two edges of Π.*

An orthoconvex polygon is *empty* if it does not contain any member of P in its interior. Our objective is to identify the largest empty orthoconvex polygon in $\mathcal{R}$.

Definition 3 *An empty orthoconvex polygon Π is said to be maximal empty orthoconvex polygon ($MEOP$) if there exists no other empty orthoconvex polygon Π' that properly encloses Π.*

It is easy to observe that an $MEOP$ is bounded by two R-stairs $R_{t\ell}$ and R_{br}, and two F-stairs F_{tr} and $F_{b\ell}$, where $R_{t\ell}$ spans from the left boundary to the top boundary, R_{br} spans from the bottom boundary to the right boundary, F_{tr} spans from the top boundary to the right boundary and $F_{b\ell}$ spans from the left boundary to the bottom boundary of $\mathcal{R}$, and each concave vertex of these stairs must coincide with a member in P (see Figure 1). It may be observed that an R-stair (or an F-stair) may degenerate to a corner point of $\mathcal{R}$.

The number of maximal empty staircase polygons amidst a point set of size n may be exponential in n; however, the maximum-area empty staircase polygon can be computed in $O(n^2)$ time [10]. Since a maximal empty staircase polygon is an $MEOP$ as well, the same

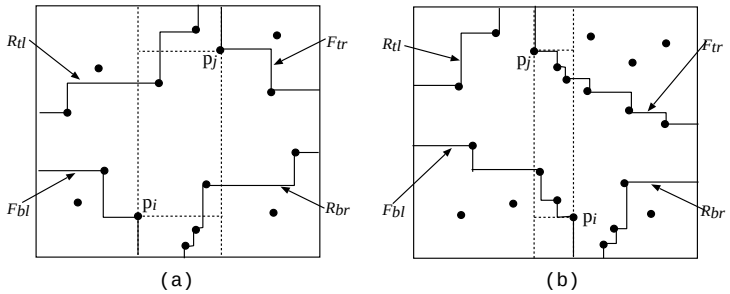


Figure 1: Orthoconvex polygons

combinatorial explosion holds for $MEOP$ also. We now present a polynomial time algorithm for computing the maximum-area $MEOP$.

3 Algorithm

We shall consider all possible pairs of points $p_i, p_j \in P$, and identify the maximum area $MEOP$ with $p_i \in F_{b\ell}$ as the closest point of the bottom boundary of $\mathcal{R}$, and $p_j \in F_{tr}$ as the closest point of the top boundary of $\mathcal{R}$. The points p_i and p_j are said to be the bottom-pivot and top-pivot respectively, and the corresponding $MEOP$ is denoted by $MEOP(p_i, p_j)$. We will use S to denote the vertical slab bounded by V_i and V_j. The projections of a point $p_k \in S$ on V_i, V_j, H_i and H_j are denoted by q_k, q'_k, r_k and r'_k respectively. We now separately consider two cases: (i) $x(p_i) < x(p_j)$, and (ii) $x(p_i) > x(p_j)$.

In Case (i), the lines V_i and V_j split the point set P into three parts, P_1, P_2 and P_3, where P_1 and P_3 are the set of points to the left of V_i and to the right of V_j respectively, and the points in P_2 lie inside the vertical slab S. If V_i hits the top and bottom boundaries of $\mathcal{R}$ at t_1 and b_1 respectively, and V_j hits the top and bottom boundaries of $\mathcal{R}$ at t_2 and b_2 respectively. Now, the portion of the $MEOP$ inside the vertical slab S, denoted by $M_2(b_1, t_2)$, is an empty staircase polygon with diagonally opposite corners b_1 and t_2 among the points in P_2. The two stairs of $M_2(b_1, t_2)$ are parts of the rising stairs R_{br} and R_{tl} respectively. If R_{tl} hits V_i at q_α, then the portion of the $MEOP$ to the left of V_i, denoted by $M_1(q_\alpha)$, is an empty edge-visible polygon with base $[p_i, q_\alpha]$ among the points in P_1 such that every point inside the polygon is visible from its base $[p_i, q_\alpha]$. Similarly, if R_{br} hits V_j at q'_β then $M_3(q'_\beta)$ is an empty edge-visible polygon with base $[p_j, q'_\beta]$ among the points in P_3.

In Case (ii), V_j is to the left of V_i. Here M_2 is an empty staircase polygon from p_i to p_j, and these are the parts of F_{bl} and F_{tr} of the $MEOP$ respectively. If F_{bl} (resp. F_{tr}) hits V_j (resp. V_i) at q'_α (resp. q_β), then $M_1(q'_\alpha)$ (portion to the left of V_j) is an edge-visible polygon with base $[q'_\alpha, p_j]$, and $M_3(q_\beta)$ (portion to the right of V_i) is an edge-visible polygon with base $[q_\beta, p_i]$. After fixing p_i and p_j as the bottom-pivot and top-pivot respectively, we need to choose M_1, M_2 and M_3 such that the sum of areas of these three polygons is maximum among all such polygons. We shall describe our algorithm for Case (i) only. Case (ii) can easily be handled using a similar method. For Case (i), we explain the method of computing the desired M_1 and M_2. The computation of M_3 is the same as that of M_1.

3.1 Computation of M_1

Let us consider a point $p_i \in P$. Let $P_1 = \{p_k | x(p_k) < x(p_i)\}$ and $Q = \{p_k | x(p_k) > x(p_i)$ and $y(p_k) > y(p_i)\}$. Q includes the top-right corner of $\mathcal{R}$, and $|Q| = m + 1$. Let $q_0, q_1, q_2, \ldots, q_m$ denote the projections of the points in Q on the vertical line V_i in decreasing order of their y-coordinates. We create an array $EVL(p_i)$ whose elements are the maximum area empty edge-visible polygon $M_1(q_k)$ with $[p_i, q_k]$ as the *base* for all $k = 0, 1, 2, \ldots, m$.

We use vertical line sweep among the points in P_1 starting from the position of V_i to create a height-balanced binary tree $\mathcal{T}$. Its each node v is represented as a 5-tuple $(I, x\_val, y\_val, \Delta, \delta)$. I is the base of the edge-visible polygons attached to node v. (x_val, y_val) is the point where the node v is generated, and Δ contains the area of the largest edge visible polygon rooted at that node. The Δ parameters are computed in two passes. In the forward pass during the sweep, the Δ parameter of a node contains the area of the edge visible polygon that is computed so far at that node. At the end of the sweep, a backward pass is executed from the leaf level of $\mathcal{T}$ up to its root, and Δ value of each node is properly set. The δ value of all nodes are 0 at the time of creation of $\mathcal{T}$; it will be set and used during the computation of $M_1(q_k)$ for different q_k.

Creation of $\mathcal{T}$

The root r of $\mathcal{T}$ corresponds to the entire interval $I = [p_i, q_0]$; its x_val and Δ parameters are set to $x(p_i)$ and 0 respectively. A vertical line sweep is performed from $x = x(p_i)$ towards left. When a point $p = (x(p), y(p)) \in P_1$ is faced by the sweep line, the leaf nodes in $\mathcal{T}$ are searched. If $y(p)$ lies in the inter-

val $[\alpha, \beta]$ of a node $v = ([\alpha, \beta], \mu, \nu, \Delta, \delta)$, we compute $\Delta^* = \Delta + (\mu - x(p)) \times (\beta - \alpha)$. Next, we create two children of v, namely $v_a = ([\alpha, y(p)], x(p), y(p), \Delta^*, 0)$ and $v_b = ([y(p), \beta], x(p), y(p), \Delta^*, 0)$. Finally, the backward pass is executed from leaves towards the root in post-order manner to set the Δ values as described above.

Computation of $M_1(q_k)$

$M_1(q_0) = \Delta$ attached to the root node r. While processing q_k, we assume that q_{k-1} is already processed. We start scaning from the root of $\mathcal{T}$. At a particular node $v = ([\alpha, \beta], \mu, \nu, \Delta(v), \delta(v))$ on the search path, one of the following two situations may happen: (i) $\nu \geq y(q_k)$ and (ii) $\nu < y(q_k)$.

In Case (i), we compute $A = (x(p_i) - \mu) \times (y(q_k) - y(q_{k-1}))$. The area A is to be subtracted from all the edge-visible polygons stored in the the right-child v_b of the node v. We subtract A from $\Delta(v_b)$. Without entirely traversing the subtree rooted at v_b, we add A in $\delta(v_b)$. The motivation is that, while processing some other q_ℓ, if the subtree rooted at v_b is traversed, $\delta(v_b)$ will be subtracted from the Δ value of those nodes. The search proceeds towards the left child of the node v. At each move from a node v to its children v', $\delta(v)$ is subtracted from $\Delta(v')$, and added to $\delta(v')$, and then $\delta(v)$ is set to 0.

In Case (ii), the edge visible polygon in the left child v_a of the node v does not exist; so we delete the subtree rooted at v_a and the node v also; the search proceeds towards the right child of v. The propagation of δ is to be performed at each step, but the computation of excess area A is to be performed when Case (i) arises. Next, a backward pass is needed to set the Δ field of all the nodes in the updated $\mathcal{T}$. Finally, $M_1(q_k)$ is set with the Δ value of the root of the updated $\mathcal{T}$.

Lemma 1 *The computation of $M_1(q_k)$ for all $k = 0, 1, \ldots, m$ needs $O(n^2)$ time.*

Proof. The creation of $\mathcal{T}$ needs $O(n \log n)$ time. The lemma follows from the fact that the combinatorial complexity of $M_1(q_k)$ is $O(n)$ for all $k = 0, 1, \ldots, m$. □

Similarly, with each point $p_i \in P$, an array $EVR(p_i)$ is attached. If the projections of the points $\{p_k | x(p_k) < x(p_i)$ & $y(p_k) < y(p_i)\}$ are denoted by $q'_0, q'_1, q'_2, \ldots, q'_{m'}$, then $|EVR(p_i)| = m' + 1$, and the content of its k-th element is the largest empty edge-visible polygon $M_3(q'_k)$ with base $[p_i, q'_k]$ among the points to the right of V_i.

Lemma 1 says that $EVL(p_i)$ and $EVR(p_i)$ for all $i = 1, 2, \ldots, n$ can be created in $O(n^3)$ time.

3.2 Computation of M_2

Consider the processing of a pair of points $p_i, p_j \in P$ satisfying $x(p_i) < x(p_j)$. Let R_{ij} be the rectangle with p_i and p_j at its diagonally opposite corner, and P_2^* be the set of points in P that lie in R_{ij}. Here M_2 can be split into three parts: the L_polygons inside the slab S below H_i and above H_j, and the empty staircase polygon $MESP(p_i, p_j)$ from p_i to p_j inside R_{ij}. The objective is to choose the staircase polygon such that the sum of its area along with the area of the corresponding L_polygons in S and the edge-visible polygons M_1 and M_3 on two sides of S is maximum.

Computation of L-polygons

Let $r_1, r_2, \ldots, r_m$ be the projections of the points in R_{ij} on H_i in the increasing order of their x-coordinates, and r_{m+1} is the intersection of H_i and V_j. We execute a horizontal line sweep among the points in S from the floor of $\mathcal{R}$ up to H_i to compute the area of the maximal empty L-polygons $LB(r_k)$ for $k = 1, 2, \ldots, m + 1$. The upper stair of $LB(r_k)$ is an L-path with p_i at its corner, and the lower stair is a staircase path from b_1 to r_k. The L-polygons $LA(r_k), k = 0, 1, 2, \ldots, m$ above H_j are computed in an exactly similar manner; here r_0 is the intersection point of V_i and H_j. This needs $O(n)$ time in the worst case.

Computation of the staircase polygon

We now describe the last step of our algorithm for computing the maximal empty staircase polygons $MESP(p_i, p_j)$ considering the area of the coresponding L-polygons and edge-visible polygons such that the total area of $MEOP(p_i, p_j)$ is maximum. Let G be a directed graph with vertices corresponding to the points in R_{ij} and edges $\{e_{k\ell} = (p_k, p_\ell) | p_k, p_\ell \in P_2^*$ with $x(p_k) < x(p_\ell)$ and $y(p_k) < y(p_\ell)\}$. Any path from p_i to p_j in G corresponds to the lower stair of an $MESP(p_i, p_j)$; but there are different choices of the upper stairs corresponding to the same lower stair. The problem of computing the maximum area $MESP(p_i, p_j)$ can be formulated as finding the maximum weighted path in an weighted directed graph, called the staircase graph [10],.

Definition 4 *[10] Let (p_a, p_b) be an edge of G. The point $(x(p_a), y(p_b))$, where the vertical line V_a abuts the horizontal line H_b, is called the footprint of p_b contributed by p_a, and is denoted by b_a. The footprint of the point p_i is p_i itself. We use $FP(p)$ to denote the set of footprints of the point $p \in P_2^*$.*

Definition 5 *[10] The staircase graph $SG = (V, E)$ is a weighted digraph with nodes $V = \cup_{p_a \in P_2^*} FP(p_a)$. A footprint $b_a \in FP(p_b)$ has a directed edge to a footprint $d_c \in FP(p_d)$ if (p_b, p_d) is an edge in G, and the upper stair of the L_polygon$[b_a, p_d]$ meets the horizontal line H_d at the footprint d_c. The weight of the edge (b_a, d_c), denoted by $w(b_a, d_c)$ is equal to the area of the L_polygon$[b_a, p_d]$.*

A path in SG corresponds to a unique staircase polygon from p_i to p_j. Assuming $|P_2^*| = m$, the worst-case number of vertices and edges in SG are $O(m^2)$ and $O(m^3)$ respectively, and the maximum weighted path in SG can be found in $O(m^3)$ time.

In our problem, computing the maximum area empty staircase polygon among the points in P_2 will not suffice. Suppose $MEOP(p_i, p_j)$ consists of a staircase polygon $MESP(p_i, p_j)$, that has edges (p_i, q_α) along V_i, $(p_j, q_{\alpha'})$ along V_j, (p_i, r_β) along H_i and $(p_j, r_{\beta'})$ along H_j, then it includes (i) an edge-visible polygon with base (p_i, q) to the left of V_i, (ii) an edge visible polygon with base (p_j, q') to the right of V_j, (ii) an L-polygon with base (p_i, r) and (iv) an L-polygon with base (p_j, r'). Let q, q', r and r' correspond to $p_\alpha, p_{\alpha'}, p_\beta, p_{\beta'} \in P_2$ respectively. Thus, in order to compute the $MEOP$ of maximum area, we need to modify the weight of some edges of the graph SG as follows, and then compute the maximum weighted path in the graph SG.

- For each α such that $p_\alpha \in P_2$, change the weight of its each outgoing edge e to $w(e) + area(M_1(p_i, q_\alpha))$.
- For each edge $e' = (p_i, \beta_{k'})$, change the weight of e' to $w(e') + area(LB(p_i, r_\beta))$.
- For each edge α' such that $p_{\alpha'} \in P_2$, if there exists an edge e^{**} from a footprint of α' to a footprint of p_j, then change its weight to $w(e^{**}) + area(M_3(p_j, q'_{\alpha'}))$.
- For each incoming edge e' on $j_{\beta'}$), change the weight of e' to $w(e') + area(LA(p_j, r_{\beta'}))$.

Theorem 2 *The largest $MEOP$ among a set of n points can be computed in $O(n^5)$ time and $O(n^2)$ space.*

In [10], it is also shown that the geometric properties of the problem can be exploited to design an algorithm for computing the empty staircase polygon of maximum area in $O(|P_2|^2)$ time and space. Here the results of procesing a point p_j for computing the $MESP(o, p_j)$ are used to compute $MESP(o, p_k)$, where $x(p_k) > x(p_j)$ and $y(p_k) > y(p_j)$; The point o is the bottom left corner of the rectangular floor. We will use the same principle to reduce the time complexity of the problem of computing the maximum area $MEOP$ to $O(n^3)$.

4 Further improvement

We will fix a point $p_i \in P$, and consider all $p_j \in P$ with $x(p_j) > x(p_i)$ and $y(p_j) > y(p_i)$. We also use the the notion of complete processing of a point p_j [10]. A point p_j is said to be *completely processed* if all the edges incident to p_j in the graph G are processed. When a point p_j is completely processed, the weights of different paths in the staircase graph (SG) with bottom-pivot and p_i and p_j respectively, are available at the footprints of p_j. Each of these polygons has included the corresponding $M_1(p_i, q_\alpha)$ and $LB(p_i, r_\beta)$ for some appropriate $p_\alpha, p_\beta \in S$. In order to get the maximum area $MEOP$ with a $MESP(p_i, p_j)$, we need to add the area of the appropriate $M_3(p_j, q_{\alpha'})$ and $LA(p_j, r_{\beta'})$, where $p_{\alpha'}, p_{\beta'} \in S$. We can compute the array L containing $LA(p_j, r_k)$ for all the points $p_k \in S$ above H_j in $O(n)$ time. The values of $area(M_3(p_j, q_k))$ are all available in the array $EVR(p_j)$. Now we can use $EVR(p_j)$, L, and the area attached to the different footprints of p_j to compute the largest $MEOP(p_i, p_j)$ in $O(n)$ time.

We process the points above H_i in S by sweeping a horizontal line upwards. After complete processing of p_j, we compute $MEOP(p_i, p_j)$ as described above, and then process the outgoing edges of p_j in G. Thus, we have the following theorem:

Theorem 3 *The largest $MEOP$ among a set of n points can be computed in $O(n^3)$ time and $O(n^2)$ space.*

Proof. For a fixed p_i, the generation of footprints for all $p_j \in P$ with $x(p_j) > x(p_i)$ and $y(p_j) > y(p_i)$ needs $O(n^2)$ time [10]. The additional time required to process each $p_j \in P$ is $O(n)$. Since we need to fix each p_i, the time complexity result follows.

The space complexity result follows from the fact that the preprocessed arrays $M_1(p_i)$ and $M_3(p_i)$ are of size $O(n)$ in the worst case. Moreover while processing a point p_i, the number of footprints generated is $O(n^2)$ in the worst case. These need to be stored during the processing of p_i. □

References

[1] A. Aggarwal and S. Suri, *Fast algorithms for computing the largest empty rectangle*, in Proc. 3rd. Annual Symp. Comp. Geom., pp. 278-290, 1987.

[2] C. Audet, P. Hansen and F. Messine, *Extremal problems for convex polygons*, J. of Global Optimization, vol. 38, pp. 163-179, 2007.

[3] A. Biswas, P. Bhowmick and B. B. Bhattacharya, *Finding the orthogonal hull of a digital object*, Proc. IWCIA, LNCS-4958, pp. 124-135, 2008.

[4] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. Guibas, *Finding extremal polygons*, Proc. 14th Annual ACM Symp. on Theory of comput., pp. 282 - 289, 1982.

[5] R. P. Boland and J. Urrutia, *Finding the largest axis aligned rectangle in a polygon in $o(n \log n)$ time*, Proc. Canad. Conf. in Comp. Geom., pp. 41-44, 2001.

[6] K. Daniels, V. Milenkovic and D. Roth, *Finding the largest area axis-parallel rectangle in a polygon*, Computational Geometry: Theory and Applications, vol. 7, pp. 125-148, 1997.

[7] A. Datta and G. D. S. Ramkumar, *On some largest empty orthoconvex polygons in a point set*, Proc. FSTTCS, LNCS 472, pp. 270-285, 1990.

[8] G. T. Herman and A. Kuba (eds.), Advances in Discrete Tomography and its Applications, Birkhauser, 2007.

[9] S. C. Nandy, A. Sinha and B. B. Bhattacharya, *Location of the largest empty rectangle among arbitrary obstacles*, Proc. FSTTCS, LNCS 880, pp. 159-170, 1994.

[10] S. C. Nandy and B. B. Bhattacharya, *On finding an empty staircase polygon of largest area (width) in a planar point-set* Computational Geometry: Theory and Applications, vol. 26, pp. 143-171, 2003.

Minimum blocking sets of circles for a set of lines in the plane

Natasa Jovanovic* Jan Korst†

Augustus J.E.M. Janssen‡

Abstract

A circle C is occluded by a set of circles $C_1, \ldots, C_n$ if every line that intersects C also intersects at least one of the C_i, $i = 1, \ldots, n$. In this paper, we focus on determining the minimum number of circles that occlude a given circle assuming that all circles have radius 1 and their mutual distance is at least d. As main contribution of this paper, we present upper and lower bounds on this minimal number of circles for $2 \leq d \leq 4$, as well as the algorithms we used to derive them.

1 Introduction

A set of circles $C_1, \ldots, C_n$ occludes a circle C if every line that intersects circle C also intersects at least one of these circles. Such a set of circles is called a blocking set of circles. In this paper, we discuss the case where all the circles have radius 1 and the distance between the circles' centers is at least d. We present the algorithms we used to determine the upper and lower bounds on the minimum cardinality of the blocking sets of circles for different values of d, as well as the results we obtained.

The problem of an occluded convex shape in a two-dimensional plane arises in the process of detecting objects using light beams from arbitrary directions. Here, we limit ourselves to circular objects of equal size in the plane.

In the literature there is related work on blocking sets in a projective plane. [4] defines a similar problem and some solutions are proposed by [2, 1]. Explaining the difference to the problem in a projective plane is beyond the scope of this paper. The related literature indicates that this work is entirely novel - to the best of our knowledge the problem we investigate has not been studied before.

2 Problem Description

Let C be a unit circle in a two-dimensional plane with its center positioned at point p_0 and let $\mathcal{L}_C$ be the set of all lines that intersect C.

Definition 1 (blocking set) *Given a set of lines L, a set B of unit circles is called a blocking set for L if and only if each line $l \in L$ intersects at least one of the circles in B.*

A blocking set for L is denoted as $B(L)$.

Definition 2 (d-apart blocking set) *Given circle C at position p_0, a blocking set $B(L)$ of n unit circles positioned at $p_1, \ldots, p_n$ is called d-apart if and only if for each pair $i, j \in \{0, 1, \ldots, n\}$ with $i \neq j$, the Euclidean distance $d(p_i, p_j) \geq d$, for a given distance d.*

Note that not only the mutual distance between the centers of the circles in $B(L)$ needs to be at least d, but also the distance to the center of circle C.

Problem 1 (Minimum blocking set problem) *Given a unit circle C, corresponding set $\mathcal{L}_C$ of lines and distance d, find a d-apart blocking set $B(\mathcal{L}_C)$ of minimum cardinality.*

The cardinality of a minimum blocking set for given d is denoted as N_d.

Let d be a given distance with $2 \leq d \leq 4$. The Minimum blocking set problem has a simple solution [3] of cardinality 4 for $d = 2$; see Figure 1. Each line that has at least one intersection point with the circle C in the middle, also has non-empty intersections with at least one of the 4 circles around it.

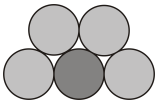


Figure 1: Minimum blocking set for $d = 2$.

Another example of a blocking set is shown in Figure 2. For $d = 4$, the 15 light shaded circles positioned at points of a regular triangular grid, as illustrated below, block all the lines that intersect the circle in the middle.

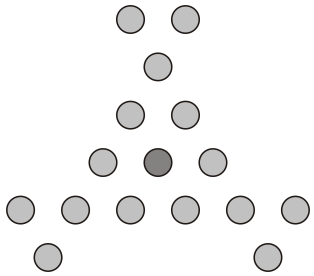


Figure 2: Example of a blocking set: 15 circles block all lines that intersect the dark circle in the middle.

The problem of determining N_d for an arbitrary distance $d > 2$ is difficult. We derive upper and lower bounds on N_d using two different approaches that we explain in detail in Section 3 and Section 4, respectively.

*Department of Mathematics and Computer Science, Eindhoven University of Technology, n.jovanovic@tue.nl

†Philips Research Europe, jan.korst@philips.com

‡Philips Research Europe, A.J.E.M.Janssen@philips.com

Next, we elaborate on the properties of a blocking set. Let a d-apart blocking set $B(\mathcal{L}_\mathcal{C})$ be positioned at points $p_1, p_2, \ldots, p_n$. We say that the position p_i of a circle in $B(\mathcal{L}_\mathcal{C})$ is *closest* if and only if $d(p_0, p_i) \leq d(p_0, p)$ for all points p on the line through p_0 and p_i for which it holds that $d(p, p_j) \geq d, j = 0, \ldots, n, j \neq i$.

Definition 3 (maximally shrunk blocking set) *A d-apart blocking set $B(\mathcal{L}_\mathcal{C})$ is maximally shrunk if and only if every circle of $B(\mathcal{L}_\mathrm{C})$ is on of its closest positions.*

A set B of circles is said to *dominate* another set of circles B' if and only if all lines that are blocked by B' are also blocked by B.

Lemma 1 *Any d-apart blocking set $B(\mathcal{L}_\mathcal{C})$ is dominated by a maximally shrunk d-apart blocking set $B'(\mathcal{L}_\mathcal{C})$.*

Proof. Let p_i be the position of an arbitrary circle of the blocking set $B(\mathcal{L}_\mathcal{C})$ and let p'_i be the corresponding closest position of the circle of the maximally shrunk blocking set $B'(\mathcal{L}_\mathcal{C})$ (see Figure 3).

By elementary calculus it can be shown that every line blocked by the circle at p_i is also blocked by the circle at p'_i. □

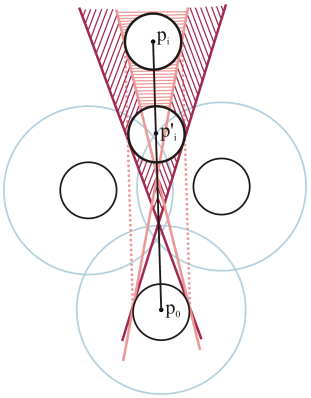


Figure 3: Every line blocked by the circle at p_i is also blocked by the circle at p'_i.

As a consequence of Lemma 1, there are optimal solutions of the minimum blocking set problem within the class of maximally shrunk blocking sets.

3 Deriving upper bounds

In this section, we construct a special class of blocking sets providing upper bounds on N_d. We focus on blocking sets that have an even number k of circles at a distance d from the center of the given circle C, such that they form a regular polygon, with either $k = 4$ or $k = 6$.

Each of these first k circles blocks some lines from the given set $\mathcal{L}_\mathcal{C}$. The remaining set of lines, denoted as $\mathcal{L}'_\mathcal{C}$, can be subdivided into disjunct *bundles of lines*. For $k = 4$ and $k = 6$, we obtain 2 and 3 bundles, respectively (see Figure 4).

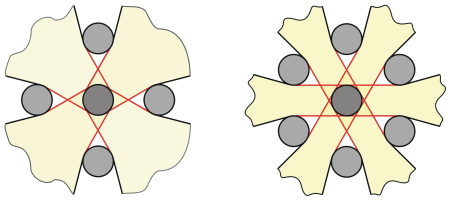


Figure 4: The remaining sets of lines grouped as disjunct bundles of lines.

Definition 4 (bundle of lines) *A bundle of lines $L(p_i, p_j) \subset \mathcal{L}'_\mathcal{C}$ between two adjacent circles positioned at p_i and p_j contains a line $l \in \mathcal{L}'_\mathcal{C}$ if and only if l intersects the line segment $p_i p_j$ and $d(p_i, l) \geq 1$, $d(p_j, l) \geq 1$.*

Note that the pairs of diametrically opposite circles from the first k circles define identical bundles of lines.

Let $L(p_i, p_j)$ be a bundle of lines and let l' be a line such that $p_0 \in l'$ and $l' \notin L(p_i, p_j)$. The two lines t and t' of the bundle $L(p_i, p_j)$ that form the largest and the smallest angle with the line l' are called *extremal lines* and the angle between them is denoted as θ. Note that the lines t and t' are tangent to the circles C_i and C_j positioned at p_i and p_j.

We can now define a subproblem of the minimum blocking set problem as follows.

Problem 2 (Bundle blocking problem) *Given a bundle of lines $L(p_i, p_j) \subset \mathcal{L}'_\mathcal{C}$, find a blocking set $B(L(p_i, p_j))$ of minimum cardinality, such that the blocking set $B(L(p_i, p_j)) \cup \{C_i, C_j\}$ is d-apart.*

Given the restriction on the mutual distance, for each of the circles we define a *boundary circle* that determines the region in which it is not possible to place any additional circles. Therefore, a blocking set for a bundle of lines can be chosen to consist of the circles positioned between the extremal lines and on or outside the boundary circles (see the shaded area in Figure 5 as an example).

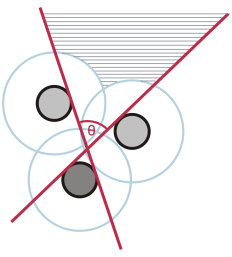


Figure 5: An example of a bundle of lines.

Every additional circle that we place in the shaded area reduces the set of lines of the bundle. However, depending

on the position of the added circle, the non-blocked lines can all be in one bundle or can be separated into two disjunct bundles. In both cases, the angle(s) between the extremal lines of the new bundle(s) is/are strictly smaller than the angle between the extremal lines before placing the additional circle.

Next we propose a heuristic algorithm that tries to block a given bundle of lines $L(p_i, p_j)$ by 1, 2, 3, 4, or 5 circles. We discuss each of the cases separately.

Blocking a bundle by 1 circle. To test whether or not one circle can block all the lines, we use a simple procedure. Let t and t' be the two extremal lines of $L(p_i, p_j)$ and let θ be the angle between them. Let $\bar{p}$ be the intersection point of the bisector of the angle θ and a boundary circle, such that $\bar{p}$ is not in the interior of any other boundary circle. If $d(\bar{p}, t) \leq 1$, it is possible to block the bundle with one circle (see Figure 6 - left).

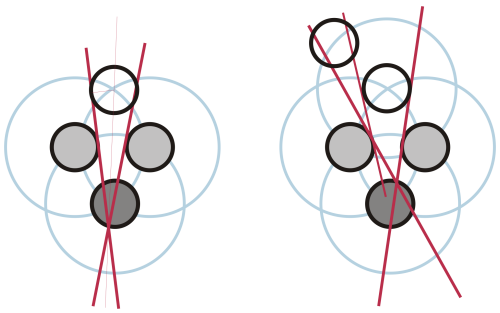


Figure 6: Blocking bundles of lines by one circle (left) and two circles (right).

Blocking a bundle by 2 circles. The essential part of the test whether or not two circles can block a given bundle is the observation that the first added circle can be chosen to be tangent to one of the extremal lines. Otherwise, it would separate the bundle into two disjunct bundles, requiring at least two additional circles for the blocking. Therefore, we add one circle to the closest position such that one of the extremal lines is tangent to the circle and test whether or not the rest of the lines (new bundle) can be blocked by one circle (see Figure 6 - right).

To test whether a bundle can be blocked by 3 or more circles, we need an additional construction method: find the closest position of one circle such that one of the new bundles of lines defined by that circle can be blocked by *exactly* one circle. The specific positions of the two circles can be found using analytic geometry and considering different cases, but we leave out the discussion in the interest of space.

Different positions of the first added circle result in different bundles of non-blocked lines. Therefore, we consider some cases of that positioning for testing whether the given bundle can be blocked by 3 or more circles.

Blocking a bundle by 3 circles. The analysis is by considering two cases. In the first case, we add the first circle such that it is tangent to one of the extremal lines. Then, the non-blocked lines are in one bundle, and we test whether or not these can be blocked by 2 circles.

In the second case, we use the construction method mentioned above to find the position of the first circle such that one of the two new bundles of lines can be blocked by exactly one circle. Then, we check whether the non-blocked bundle of lines can be blocked by 1 circle.

Blocking a bundle by 4 circles. This test consists of checking two cases, as in the test with 3 circles. In the first case, we add one circle such that it is tangent to one of the extremal lines and check whether the new bundle can be blocked by 3 circles.

In the second case, we add two circles using the construction method and check whether the remaining bundle can be blocked by 2 circles.

Blocking a bundle by 5 circles. Besides the two cases similar to those in tests with 3 and 4 circles, we have an additional one: we place the first circle at the intersection point of the angle bisector and a boundary circle and check whether both new bundles of lines can be blocked by 2 circles.

Blocking the lines by 6 or more circles has been considered. However, experimental results show that the bundles of lines defined by the first 4 or 6 circles on regular polygon positions can be blocked by at most 5 circles for $2 \leq d \leq 4$.

4 Deriving lower bounds

In this section, we explain the approach we use to obtain lower bounds on N_d. For this, we consider the set $L \subset \mathcal{L}_C$ that consists of all the lines from $\mathcal{L}_\mathcal{C}$ that pass through the center p_0 of the given circle C. The cardinality of a minimum blocking set $B(L)$ represents a lower bound on N_d since $L \subset \mathcal{L}_\mathcal{C}$. A minimum blocking set $B(L)$ can be constructed, since one can prove that in the set of minimum blocking sets for L, there are always non-overlapping ones, i.e. blocking sets for which the intersection of lines blocked by any two pairs of circles consists of at most one (tangent) line. The number of non-overlapping blocking sets for L of cardinality N can be reduced to a few cases, where for each case, we can determine the largest value of d possible for that case. The example in Figure 7 shows the optimal non-overlapping blocking set $B(L)$ for $k = 5$. The maximum distance d for which 5 circles can block the lines from L is simply derived as $d = 1/\sin\frac{\pi}{10}$.

5 Upper and lower bounds - results

In this section we present the upper and lower bounds on minimum blocking sets that we obtained by the methods explained in Sections 3 and 4.

In Figure 8, d is given on the horizontal axis. The number of circles is given on the vertical axis. For example, for $d = 3$, we have $5 \leq N_d \leq 9$.

Obviously, N_d is a monotonic function of d with non-negative integer values. Table 1 gives the d-values of the

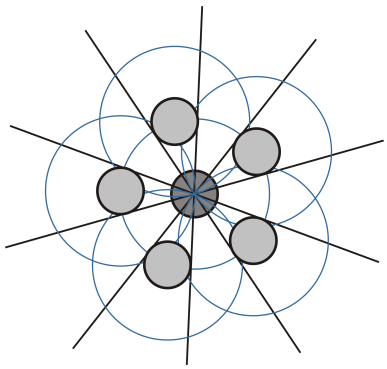


Figure 7: Every line passing through the center of the middle circle is blocked by at least one of the 5 circles around it.

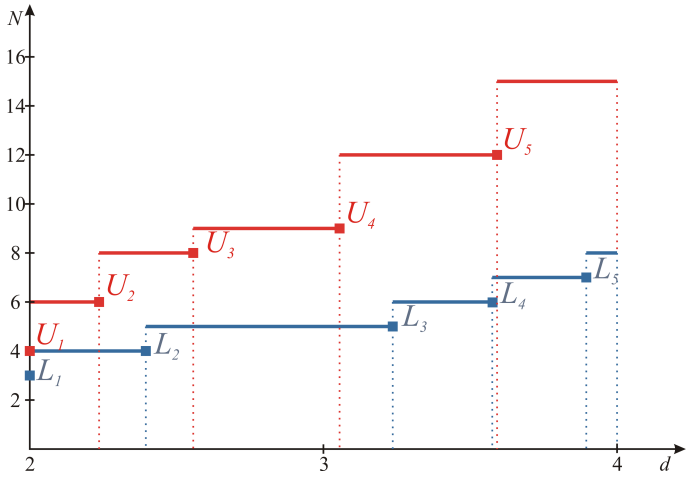


Figure 8: Upper and lower bounds on the cardinality of minimum blocking sets.

points where the bounds on that function change value. The d-value of the lower bound point L_4 is

$$d_4 = 1/\sqrt{\tfrac{9}{16} - \tfrac{1}{16}y - \tfrac{1}{2}\sqrt{\tfrac{9}{32} + \tfrac{1}{16\sqrt[3]{18}}x + \tfrac{1}{8x\sqrt[3]{12}} + \tfrac{3}{32y}}},$$

where $x = \sqrt[3]{81 - \sqrt{6549}}$ and $y = \sqrt{9 - \tfrac{4}{x}\sqrt[3]{\tfrac{2}{3}} - 2x\sqrt[3]{\tfrac{4}{9}}}$.

i	1	2	3	4	5
U_i	2	2.2361	2.5776	3.0551	3.5914
L_i	$1/\sin\frac{\pi}{6}$	$\frac{\sqrt{8}}{\sqrt{5-\sqrt{13}}}$	$1/\sin\frac{\pi}{10}$	d_4	$4\cos\frac{\pi}{14}$

Table 1: Values of d where upper bounds U_i and lower bounds L_i change value.

By randomly generating maximally shrunk blocking sets, we did not obtain sets with less circles than the corresponding upper bounds. This indicates that probably the lower bounds can be improved.

6 Conclusions

In this paper we investigated the two-dimensional geometrical problem of "blocking" lines that intersect a given unit circle with unit circles. The circles are positioned in such a way that the distance between every two circles is at least a given distance d. We focused on the minimum number N_d of circles that block the lines under given conditions.

The given problem is difficult and we did not find straightforward solutions for it. Our approach has been to consider examples of some simple positionings of circles and downsize the problem to the point where we can easily determine the minimum number of circles needed for the complete blocking. In that way, we provided upper bounds on N_d. We also proposed provable lower bounds on N_d.

The main challenge of this work still remains - finding the minimal number of circles that can cause occlusion for given minimal mutual distance between the circles. The problem can be generalized and investigated in three-dimensional space, as the problem of minimum blocking sets of spheres for a set of lines. Furthermore, one can investigate the asymptotic behavior of N_d (for $d \to \infty$), generalize to shapes other than circles, or to blocking half-lines instead of lines. Additionally, one can investigate to what extend values of N_d are affected, if circle positions are restricted to points in a grid.

Acknowledgments

We want to thank our colleagues Ramon Clout and Verus Pronk for their suggestions and comments. We are also grateful to Emile Aarts, Johan Lukkien and Zarko Aleksovski for fruitful discussions and inspiring ideas. We especially thank Radivoje Jovanovic for a very useful contribution on finding the lower bounds. Finally, we want to thank the anonymous referees for their in-depth and very useful suggestions that helped us to substantially improve the way we present our work in this paper.

References

[1] A. Aguglia, G. Korchmaros, and A. Siciliano. Minimal covering of all chords of a conic in pg (2, q), q even. *Bull. Belg. Math. Soc*, 12:651–655, 2005.

[2] S. Innamorati and F. Zuanni. Minimum blocking configurations. *Journal of Geometry*, 55(1):86–98, 1996.

[3] K. Stephenson. *Introduction to Circle Packing: The Theory of Discrete Analytic Functions*. Cambridge University Press, 2005.

[4] J. van Lint and R. Wilson. *A Course in Combinatorics*. Cambridge University Press, 2001.

Fault-Tolerant Conflict-Free Colorings

Mohammad Ali Abam* Mark de Berg† Sheung-Hung Poon‡

1 Introduction

Background. Consider a cellular network consisting of a set of base stations, where the signal from a given base station can be received by clients within a certain distance from the base station. In general, these regions will overlap. For a client, this may lead to interference of the signals. Thus one would like to assign frequencies to the base stations such that for any client within reach of at least one base station, there is a base station within reach with a unique frequency (among all the ones within reach). The goal is to do this using only few distinct frequencies. Recently, Even *et al.* [5] introduced *conflict-free colorings*, as defined next, to model this problem.

Let $\mathcal{S}$ be a set of n objects, and let $\mathcal{R}$ be a, possibly infinite, family of ranges. In this paper, we only consider objects and ranges that are subsets of $\mathbb{R}^2$, or sometimes of $\mathbb{R}^1$. For a range $r \in \mathcal{R}$, let $\mathcal{S}(r)$ be the subset of objects from $\mathcal{S}$ intersecting the range r. A *conflict-free coloring (CF-coloring) of $\mathcal{S}$ with respect to $\mathcal{R}$* is a coloring of $\mathcal{S}$ with the following property [5]: for any range $r \in \mathcal{R}$ for which $\mathcal{S}(r) \neq \emptyset$ there is an object $o \in \mathcal{S}(r)$ with a unique color in $\mathcal{S}(r)$, that is, with a color not used by any other object in $\mathcal{S}(r)$. Trivially, a conflict-free coloring always exists: just assign a different color to each object. However, one would like to find a coloring with only few colors. This is the conflict-free coloring problem. Note that if we take $\mathcal{S}$ to be a set of disks—namely, the regions within reach of each base station—and we take $\mathcal{R}$ to be the set of all points in $\mathbb{R}^2$, then we get exactly the range-assignment problem discussed earlier. However, other versions—for example the dual version, where $\mathcal{S}$ is a point set and $\mathcal{R}$ is the family of all disks—are interesting as well.

Related work. The CF-coloring problem for points with respect to disks was studied by Even *et al.* [5]. They showed that for this setting one can always find a CF-coloring using $O(\log n)$ colors, which is tight in the worst case. They also studies CF-colorings for points with respect to disks. Har-Peled and Smorodinsky [7] extended those results by considering other range spaces. In particular, they gave sufficient conditions for a range space to allow a CF-coloring with few colors. Recently, Smorodinsky [8] improved several results from [5] by providing deterministic coloring algorithms. Chen *et al.* [4] and Bar-Noy *et al.* [2] studied various CF-coloring problems in an on-line setting. Here the objects are given one by one, and each object has to be colored when it arrives, in such a way that the coloring remains conflict-free at all times.

Our results. Base stations in cellular networks are often not completely reliable: every now and then some base station may (temporarily or permanently) fail to function properly. This leads us to study fault-tolerant CF-colorings: colorings that remain conflict-free even after some objects are deleted from $\mathcal{S}$. More precisely, a *k-fault-tolerant CF-coloring (k-FTCF-coloring)* is a coloring that remains conflict-free after an arbitrary collection of k objects is deleted from $\mathcal{S}$. Thus a k-FTCF-coloring for $k = 0$ is simply a standard CF-coloring.

Such colorings for points with respect to disks were also studied by Abellanas *et al.* [1], who showed that any set of n points admits a k-FTCF coloring with respect to disks that uses $O(k \log n)$ colors—see Section 2. We show that this is tight, and we obtain upper and lower bounds on the worst-case number of colors needed in fault-tolerant colorings for various other types of range spaces. We also obtain results on *region-fault-tolerant CF-colorings (region-FTCF-colorings)*: colorings that remain conflict-free after the objects intersecting a geometric fault region are deleted from $\mathcal{S}$.

2 k-FTCF coloring of points with respect to disks

In this section we study conflict-free colorings of a point set $\mathcal{P} = \{p_1, \ldots, p_n\}$ in the plane with respect to the family $\mathcal{D}$ of all disks in the plane.

The algorithm for $k = 0$ from Even et al. [5] for this case works as follows. Compute a maximal independent set of the Delaunay triangulation of $\mathcal{P}$; assign all points in the independent set color 1; recursively assign colors to the rest of the points, not using color 1 anymore. (Thus in the i-th recursive call, the color i is assigned to the points in the independent set.)

As observed by Abellanas *et al.* [1], generalizing this

*MADALGO Center, Department of Computer Science, Arhus University, abam@madalgo.au.dk. MADALGO is a Center of the Danish National Research Foundation.

†Department of Computer Science, TU Eindhoven, mdberg@win.tue.nl. Supported by the Netherlands' Organisation for Scientific Research under project no. 639.023.301.

‡Department of Computer Science, National Tsing Hua University, Hsin-Chu, Taiwan, spoon@cs.nthu.edu.tw.

algorithm to $k > 1$ is rather easy: we only need to replace the Delaunay triangulation by the set of all k-order Delaunay edges [6]. (Two points $p, q \in \mathcal{P}$ form a k-order Delaunay edge if and only if there is a disk containing p and q and at most k other points.) This results in the following theorem.

Theorem 1 *For any set $\mathcal{P}$ of n points in the plane, there is k-FTCF coloring with respect to disks that uses $O(k \log n)$ colors, and this is tight in the worst case.*

Proof. The correctness proof and analysis of the number of colors[1] of the method described above were already given by Aballenas *et al.* For completeness, we summarize their argument.

To show that the algorithm produces a k-FTCF coloring, let D be a disk containing at least one point. If every point inside D has a unique color, we are done. Otherwise, let i be the maximum color appearing in the disk at least twice. We shrink the disk until it contains exactly two points p and q of color i. Since p and q are in the independent set in the i-th step of the algorithm, there must be at least $k+1$ points in the shrunk disk. By the choice of i, all these points have different colors. Hence, even after deleting k of those points we still have a unique color inside D. To prove the bound on the number of colors, we use that there are $O(kn)$ k-order Delaunay edges [6], which implies there is an independent set of size $\Omega(n/k)$. If $C(n)$ denotes the number of colors used by the algorithm, we therefore have $C(n) = 1 + C(n - \Omega(n/k))$. Hence, $C(n) = O(k \log n)$.

Next we prove the lower bound (which was not given in [1]). Consider a set $\mathcal{P}$ of n points on the x-axis. Obviously, there must be $k+1$ points with a unique color. We split the points into two roughly equal size subsets, A and B, such that all points in A are to the left to all points in B. One subset, say A, must contain at least $\lceil (k+1)/2 \rceil$ points with a unique color. In particular, those colors are not used in B and we can recurse on B. If $C(n)$ denotes the minimum number of colors needed for $\mathcal{P}$, we thus have $C(n) \geq (k+1)/2 + C(n/2)$. Hence, $C(n) = \Omega(k \log n)$. □

3 k-FTCF coloring of disks with respect to points

We now turn our attention to the case where we want to color a set of disks with respect to points. We start with the 1-dimensional version of this problem, where the set of objects is a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of n intervals on the real line. Our algorithm consists of two phases.

Phase 1: We process the intervals one by one, as follows. To process I_j, we check if every point in I_j is contained in at least $k+1$ intervals from the current set $\mathcal{I}$. If this is the case, we assign color 0 to I_j and remove I_j from $\mathcal{I}$, otherwise I_j stays in $\mathcal{I}$ and does not get a color yet.

We claim that after Phase 1, every point q is contained in at most $2k+2$ intervals. Indeed, the $k+1$ intervals containing q and extending the farthest to the left, and the $k+1$ intervals containing q and extending the farthest to the right, must cover every other interval containing q. Hence, any such other interval will be removed in Phase 1.

Phase 2: Now we color the remaining intervals, only using colors from the set $S = \{1, \ldots, \lceil (3k+3)/2 \rceil + 1\}$. To this end, we sweep from left to right. When the sweep arrives at the left endpoint of an interval I, we assign a color to I, as follows. Let S_I be a set of *forbidden colors* for I in the sense that if we assign one of them to I, then the collection of intervals colored so far is not a k-FTCF anymore. We take an arbitrary color from $S \backslash S_I$ and assign it to I.

Theorem 2 *For any set of n intervals on the real line, there is a k-FTCF coloring with respect to points that uses $\lceil (3k+3)/2 \rceil + 2$ colors. Moreover, for any $n \geq 2k+2$, there is a set of n intervals such that any k-FTCF coloring needs at least $\lceil (3k+3)/2 \rceil$ colors.*

Proof. To prove the upper bound, consider the algorithm described above. The intervals with color 0 can be ignored: any point contained in such an interval is contained in at least $k+1$ intervals from the set $\mathcal{I}$ processed in Phase 2, and we will show that Phase 2 produces a k-FTCF coloring. To show this, it suffices to argue that the algorithm does not get stuck. Thus we must prove that $S \backslash S_I$ is not empty when some interval I is processed. Let p be the left endpoint of I. If p is contained in at most k other intervals, we are done, since $|S_I| \leq k$ in this case. Otherwise, let $q \in I$ be the leftmost point that is contained in *exactly* $k+1$ intervals with unique colors. Such a point exists, because (i) p is contained in at least $k+1$ uniquely colored intervals, (ii) there is a point on I contained in at most k intervals due to Phase 1, and (iii) the current k-FTCF coloring is valid. Since at most $2k+2$ intervals contain q and exactly $k+1$ of these intervals have unique colors, the number of colors which is used to color intervals containing q is at most $k+1+\lceil (k+1)/2 \rceil = \lceil (3k+3)/2 \rceil$. Hence, there is at least one color, j, not used by an interval containing q. We claim that j is not forbidden. Indeed, points in I to the left of q have at least $k+2$ unique colors by the choice of q, so we do not have to worry about them. Moreover, an interval that already has a color and contains a point to the right of q also contains q, so such an interval will not have color j. Hence, j is not forbidden.

[1] Aballenas *et al.* give a bound of $\log n / \log(24k/(24k-1))$ on the number of colors, but it is easy to see that this is $O(k \log n)$.

To prove the lower bound, it suffices to look at the case $n = 2k+2$. Let A be a set of $k+1$ intervals starting at $x = 1$ and ending at $x = 2$, and let B be a set of $k+1$ intervals starting at $x = 2$ and ending $x = 3$. Consider a k-FTCF coloring of $A \cup B$. At point $x = 2$, there must at least $k+1$ unique colors. Therefore, one of the sets, say A, contains at least $\lceil (k+1)/2 \rceil$ unique colors. Consider the point $x = 3$. Since exactly $k+1$ intervals contain p, then all of them must have different colors. Therefore, there are at least $k+1+\lceil (k+1)/2 \rceil$ different colors. □

Now we go back to the 2D problem, namely k-FTCF coloring of a set $\mathcal{D} = \{D_1, \ldots, D_n\}$ of n disks with respect to points. Our algorithm is based on a generalization of admissible subsets [7], defined as follows. We say that $\hat{\mathcal{D}}$ is an *admissible subset* of $\mathcal{D}$ if for every point $p \in \mathbb{R}^2$ at least one of the following conditions holds:

1. $p \notin \bigcup \hat{\mathcal{D}}$.
2. $p \in \bigcup \hat{\mathcal{D}}$, but only one of the disks in $\hat{\mathcal{D}}$ contains p.
3. $p \in \bigcup \hat{\mathcal{D}}$, and there are k disks in $\mathcal{D} \backslash \hat{\mathcal{D}}$ containing p.

We will show that for a set $\mathcal{D}$ there is an admissible subset of size $\Omega(n/k)$. Based on this fact, our algorithm is as follows. Compute an admissible subset, assign to all disks in the admissible subset the color 1, and color the remaining disks recursively (where in the i-th step we assign the color i). If $C(n)$ is the number of colors used by the algorithm, we have $C(n) = 1 + C(n - \Omega(n/k))$ which gives us $C(n) = O(k \log n)$. The following example shows that our algorithm is tight. Consider n disks with radius 1 whose centers are on a line and that have a common point on the line. It is easy to see that for any m $(1 \leq m \leq n)$ consecutive disks there is a point just contained in those disks. Therefore, the same lower-bound proof given in Section 2 can be applied here.

Theorem 3 *For any set of n disks in the plane, there is a k-FTCF coloring with respect to points that uses $O(k \log n)$ colors, and this bound is tight in the worst case.*

Proof. *(Sketch.)* The lower bound has been described above. To prove the upper bound, we need to prove our claim that we can always find an admissible subset of size $\Omega(n/k)$. The proof of this fact is similar to the proof of Har-Peled and Smorodinsky [7] for the non-fault-tolerant case. Here we sketch the basic steps in our proof.

Following Har-Peled and Smorodinsky, we randomly and independently color each disk in $\mathcal{D}$ black or white, each with probability 1/2. We say a point p is *unsafe* if it is contained in at least two black disks and at most k white disks. We construct a graph G over the black disks, connecting two black disks if there is an unsafe point in their intersection. Note that any independent set of G is an admissible subset. As [7] showed, G has at least $n/3$ vertices with high probability. We show that G has $O(kn)$ edges, which implies there is an admissible subset of size $\Omega(n/k)$. To show that G has $O(kn)$ edges, we proceed as follows. For any point p in the plane, let $d(p)$ denote the number of disks containing p. The probability that p is unsafe (for points p with $d(p) > 1$) is $(1/2^{d(p)}) \sum_{i=0}^{k} \binom{d(p)}{i}$. We can use this to bound the expected number of edges created by unsafe points, which we then use to bound the overall number of edges in G by $O(kn)$. □

4 Region-FTCF coloring

Let $\mathcal{F}$ be a family of regions, which we call the *fault regions*. For a fault region $F \in \mathcal{F}$ and a point set $\mathcal{P}$, we define $\mathcal{P} \ominus F$ to be the set of points that remains after the points inside F have been removed from $\mathcal{P}$. An $\mathcal{F}$-FTCF coloring of $\mathcal{P}$ with respect to disks is a CF-coloring of $\mathcal{P}$ with respect to disks such that for any $F \in \mathcal{F}$, the coloring is a CF-coloring of $\mathcal{P} \ominus F$ with respect to disks. Unfortunately, when $\mathcal{F}$ is the set of convex regions, some sets $\mathcal{P}$ require n colors in any $\mathcal{F}$-FTCF coloring with respect to disks. To see this, consider a set $\mathcal{P}$ of n points on a line. For any two points $p, q \in \mathcal{P}$, there is a convex region $\mathcal{F}$ (which is an interval) containing all points between p and q. Points p and q are adjacent in $\mathcal{P} \ominus F$, and so they must have different colors.

The concept of region-FTCF coloring can also be defined for disks with respect to points. Let $\mathcal{D}$ be a set of n disks, and define $\mathcal{D} \ominus F$ to be set of disks that remains after the disks from $\mathcal{D}$ whose centers are inside F have been removed from $\mathcal{D}$. An $\mathcal{F}$-FTCF-coloring of $\mathcal{D}$ with respect to points is a CF-coloring of $\mathcal{D}$ with respect to points such that for any $F \in \mathcal{F}$, the coloring is a CF-coloring of $\mathcal{D} \ominus F$ with respect to points.

Unfortunately this variant does not allow a good solution either: there are sets of disks that do not admit a $\mathcal{F}$-FTCF coloring with few colors, even if $\mathcal{F}$ is the set of half-planes and the radii of the disks are equal. Consider n disks with radius 1 whose centers lie on a parabola very close to each other. Let $p_1, \ldots, p_n$ be the centers of the disks. For any i and j, there is a half-plane just containing centers $p_{i+1}, \ldots, p_{j-1}$. After removing the corresponding disks, there is a point in the plane just contained in the disks whose centers are p_i and p_j, which implies those disks must have different colors. Therefore, every two disks must have different colors, which means we need n colors.

However, we can get a coloring with few colors for the 1-dimensional version of this problem. Here we have a set $\mathcal{I}$ of n intervals on the real line, and the goal is to find an $\mathcal{F}$-FTCF coloring with respect to points, where

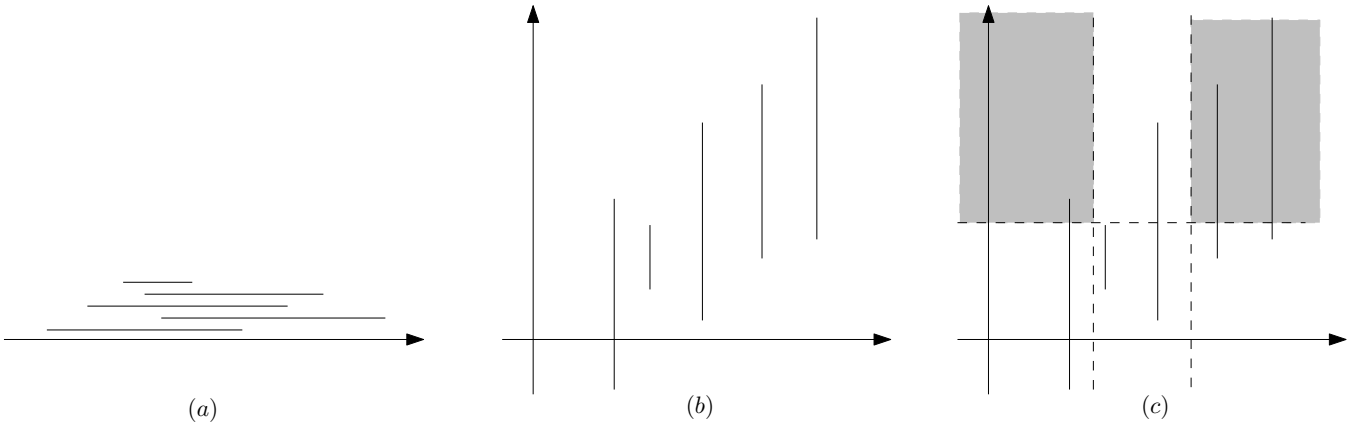


Figure 1: (a) A set of intervals (b) corresponding vertical segments in the xy-plane (c) cone regions.

$\mathcal{F}$ is the set of intervals. How to obtain such a coloring is explained next.

First we assume that all intervals have a common point (Fig. 1(a)). We map each interval $[p, q]$ with center c to the vertical segment in the xy-plane starting at (c, p) and ending at (c, q), as depicted in Fig. 1(b). Since the intervals have a common point, there is a horizontal line intersecting all vertical segments. A fault region $I = [a, b] \in \mathcal{F}$ is mapped to a vertical slab in the xy-plane, whose boundaries intersect the x-axis in $x = a$ and $x = b$. Every point $p \in \mathbb{R}^1$ is mapped to the horizontal line $y = p$ in the xy-plane. Now the problem reduces to the following. The goal is to color the vertical segments such that for every region depicted in Fig. 1(c) in gray—namely every region consisting of two axis-aligned cones whose apexes have the same y-coordinates—there is a segment with a unique color intersecting the region. To find such a CF-coloring, we apply the general approach: We construct a graph G over vertical segments, as follows. We connect two vertical segments if there is such a region intersecting just these two segments. It is easy to show that G has $O(n)$ edges. Therefore, it has an independent set of size $\Omega(n)$. We color all vertical segments in the independent set with color 1 and recursively color the rest of intervals. Since the size of independent set is $\Omega(n)$, the number of color used by the algorithm is $O(\log n)$.

If the intervals don't have a common point, we first construct an interval tree on the segments. Then we apply the above algorithm for the intervals stored in each internal node of the interval tree, where for each node we use the same set of colors. We then change the color assigned to each interval as follows: suppose it received color i and it is stored in a node in the interval tree at level ℓ. Then the new color of the interval is (i, ℓ). Since the tree has depth $O(\log n)$, the total number of colors used by the algorithm is $O(\log^2 n)$. Moreover, any two intervals stored at different nodes on the same level are disjoint, so this produces a valid coloring.

Theorem 4 *Let $\mathcal{F}$ be the family of all intervals on the real line. For any set of n intervals on the real line, there is an $\mathcal{F}$-FTCF coloring with respect to points that uses $O(\log^2 n)$ colors.*

Acknowledgement

We thank Bettina Speckmann for stimulating discussions during the early stages of our research.

References

[1] M. Abellanas, P. Bose, J. García, F. Hurtado, M. Nicolás, and P.A. Ramos. On properties of higher-order Delaunay graphs with applications. In *Abstr. 21st Europ. Workshop Comput. Geom.*, pages 119–122, 2005.

[2] A. Bar-Noy, P. Cheilaris and S. Smorodinsky. Conflict-free coloring for intervals: from offline to online. In *Proc. 18th Annu. ACM Sympos. Parallelism Algorithms Architectures (SPAA)*, pages 128–137, 2006.

[3] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[4] K. Chen, A. Fiat, H. Kaplan, M. Levy, J. Matousek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, U. Wagner and E. Welzl. Online conflict-free coloring for intervals. *SIAM J. Comput.*, in press.

[5] G. Even, Z. Lotker, D. Ron and S. Smorodinsky. Conflict-freee colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM J. Comput.* 33:94–136, 2004.

[6] J. Gudmundsson, M.H. Hammar and M.J. van Kreveld. Higher order Delaunay triangulations. *Comput. Geom. Theory Appl.* 23:85–98, 2002.

[7] S. Har-Peled and S. Smorodinsky. Conflict-Free coloring of points and simple regions in the plane. *Discr. Comput. Geom.* 34:47–70, 2005.

[8] S. Smorodinsky. On the chromatic number of some geometric hypergraphs. In *Proc. 17th Annu. ACM-SIAM Sympos. Discr. Algo. (SODA)*, pages 316–323, 2006.

A Pumping Lemma for Homometric Rhythms

Joseph O'Rourke* Perouz Taslakian† Godfried Toussaint‡

Abstract

Homometric rhythms (chords) are those with the same histogram or multiset of intervals (distances). The purpose of this note is threefold. First, to point out the potential importance of *isospectral vertices* in a pair of homometric rhythms. Second, to establish a method ("pumping") for generating an infinite sequence of homometric rhythms that include isospectral vertices. And finally, to introduce the notion of *polyphonic homometric rhythms*, which apparently have not been previously explored.

1 Introduction

Both chords of k notes on a scale of n pitches, and rhythms of k onsets repeated every n metronomic pulses, are conveniently represented by n evenly spaced points on a circle, with arithmetic mod n, i.e., in the group $\mathbb{Z}_n$. This representation dates back to the 13th-century Persian musicologist Safi Al-Din [Wri78], and continues to be the basis of analyzing music through geometry [Tou05] [Tym06]. Such sets of points on a circle are called *cyclotomic* sets in the crystallography literature [Pat44] [Bue78]. It is well-established that in the context of musical scales and chords, the intervals between the notes largely determine the aural tone of the chord. An *interval* is the shortest distance between two points, measured in either direction on the circle. This has led to an intense study of the *interval content* [Lew59]: the histogram that records, for each possible interpoint distance in a chord, the number of times it occurs. This same histogram is studied for rhythms [Tou05] and in crystallography.

Of special interest are pairs of noncongruent chords/rhythms/cyclotomic sets that have the same histogram: the sets are *homometric* in the terminology of Lindo Patterson [Pat44], who first discovered them. In crystallography, such sets yield the same X-ray pattern. In the pitch model, they are chords with the same interval content. One of the fundamental theorems in this area is the so-called *hexachordal theorem*, which states that two non-congruent complementary sets with $k{=}n/2$ (and n even) are homometric, whose earliest proof in the music literature is due to Milton Babbitt and David Lewin [Lew59].

Henceforth we specialize to the rhythm model, with each (n, k)-rhythm specified by k beats and $n{-}k$ rests on the $\mathbb{Z}_n$ circle; and we specifically focus on the structure of homometric rhythms. Figure 1 shows a pair of homometric rhythms with $(n, k){=}(12, 5)$.

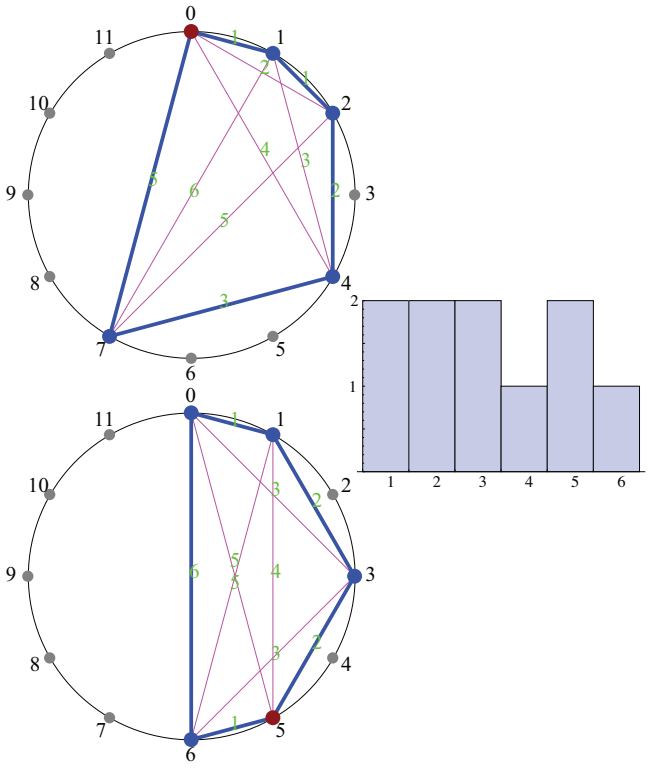


Figure 1: Homometric $(n, k){=}(12, 5)$ rhythms: $(0, 1, 2, 4, 7)$ and $(0, 1, 3, 5, 6)$. Vertices 0 and 5 in the first and second rhythms (respectively) are isospectral.

2 Isospectral Vertices

Let P and Q be two different rhythms, with $p \in P$ and $q \in Q$ vertices (onsets) in each. The vertices p and q are called *isospectral*[1] if they have the same histogram of distances to all other vertices in their respective rhythms. In Figure 1, vertex 0 in the first rhythm,

*Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu

†School of Computer Science, McGill University, Montreal, Canada. perouz@cs.mcgill.ca

‡School of Computer Science, McGill University, Montreal, Canada. godfried@cs.mcgill.ca

[1] The term is used in the literature on Golomb rulers.

and 5 in the second, are isospectral, with spectrum $\{1, 2, 4, 5\}$.

There are two reasons we consider isospectral vertices of potential significance. The first is that removal of a pair of isospectral vertices from a pair of (n, k) homometric rhythms leaves a homometric pair of $(n, k-1)$ rhythms. This raises the possibility of *shelling*: removing a particular onset from a rhythm while retaining a certain property. Shellings of Erdős-deep rhythms are studied in [DGMM+08]. Here we want to perform shelling by removing an onset from each rhythm while keeping the pair homometric.

Shellings of rhythms play an important role in musical improvisation. For example, most African drumming music consists of rhythms operating on three different strata: the unvarying timeline usually provided by one or more bells, one or more rhythmic motifs played on drums, and an improvised solo (played by the lead drummer) riding on the other rhythmic structures. Shellings of rhythms are relevant to the improvisation of solo drumming in the context of such a rhythmic background. The solo improvisation must respect the style and feeling of the piece, which is usually determined by the timeline. A common technique to achieve this effect is to "borrow" notes from the timeline, and to alternate between playing subsets of notes from the timeline and from other rhythms that interlock with it [Ank97][Aga86]. The borrowing of notes from the timeline may be regarded as a fulfillment of the requirements of style coherence, and shellings can be viewed as capturing a particular type of borrowing that achieves coherence through homometricity.

Second, as we show in Lemma 1 below, the presence of an isospectral pair permits "pumping" the rhythms to homometric pairs based on a larger $n' > n$. So isospectral pairs serve as a natural "pivot" from which to generate new homometric pairs from old ones both by removing or adding onsets.

This naturally raises the question of whether every homometric pair of rhythms must contain an isospectral pair of vertices. The answer is NO, as illustrated in Figure 2. We leave further investigation of isospectral vertices and shellings to future work.

3 The Pumping Lemma

Let P and Q be a homometric pair of (n, k)-rhythms on $\mathbb{Z}_n$, with isospectral vertices $p \in P$ and $q \in Q$. We define an (m, r)-*pumping* of P and Q, $m \geq 1$, $r \geq 0$, to be a new pair of (n', k') rhythms P' and Q' on $\mathbb{Z}_{n'}$, with $n' = mn$ and $k' = k + 2r$, obtained by replacing p in P' with $p + \{0, \pm 1, \pm 2, \ldots, \pm r\}$, and similarly replacing q in Q' with $q + \{0, \pm 1, \pm 2, \ldots, \pm r\}$.

Figure 3 shows a (m, r)=$(3, 2)$-pumping of the homometric pair from Figure 1 based on the isospectral pair p=0 and q=5. The original (n, k)=$(12, 5)$ rhythms have

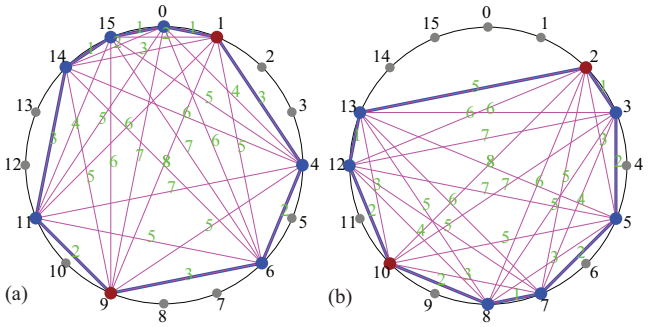


Figure 2: A pair of (n, k)=$(16, 9)$ homometric rhythms that has no isospectral pair of vertices, but does have a pair of two-vertex sets that is isospectral, $\{1, 9\}$ in (a) and $\{2, 10\}$ in (b).

been pumped to (n', k')=$(36, 9)$ rhythms. The "pumping" occurs both in $n \to mn$ and in $k \to k+2r$, although it may be that m=1 in which case n'=n, or r=0 in which case k'=k.

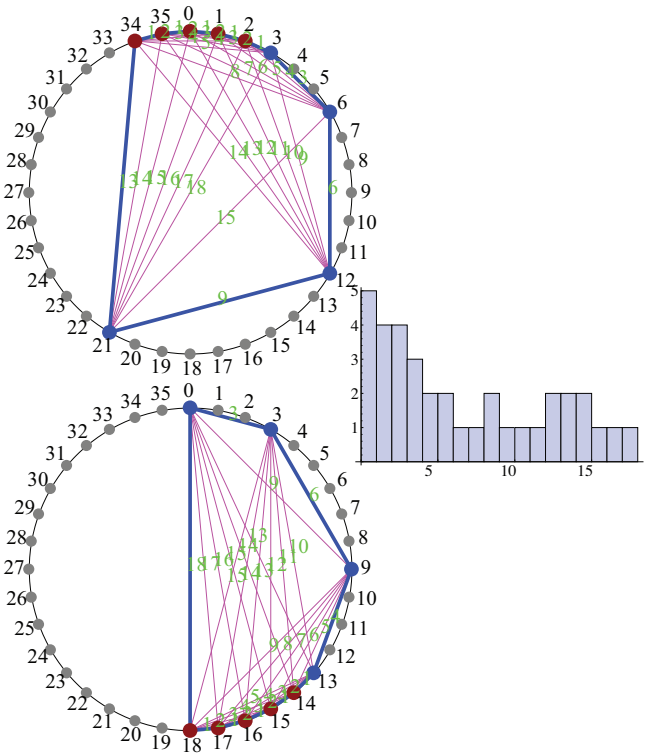


Figure 3: Pumping p=0 and q=5 in Fig. 1 with m=3, r=2, n'=mn=36. The rhythm is monophonic.

The literature focuses on *monophonic* rhythms, those whose vertices form a set with no repeated elements. The pumping lemma can produce *polyphonic* rhythms, ones in which at least one vertex has multiplicity greater than 1, i.e., the onsets form a multiset. These will be discussed further in Section 4.

Lemma 1 (Pumping) *Let P and Q be a homometric pair of (n,k) monophonic rhythms, with isospectral vertices $p \in P$ and $q \in Q$. Then any (m,r)-pumping of P and Q creates a new homometric pair P' and Q', also containing an isospectral pair. If $m \geq r+1$, then the new rhythms are monophonic; if $m \leq r$, the new rhythms could be polyphonic.*

Proof. Call the vertices $p + \{0, \pm 1, \pm 2, \ldots, \pm r\}$ in P'

$$p'_{-r}, \ldots, p'_{-2}, p'_{-1}, p'_0, p'_1, p'_2, \ldots, p'_r$$

and similarly for the q replacements in Q'.

To prove that P' and Q' are homometric, let (x', y') be a segment between two vertices of P'. Consider three cases.

1. Neither x' nor y' is among the p'_i. Then $d(x', y') = md(x, y)$, where x and y are the corresponding vertices in P.

2. $y' = p'_i$. Here there are two subcases. Let $d(x, y) = d(x, p) = d$. Note that the diameter of the circle $\mathbb{Z}_n$ is $n/2$.

   (a) $d = n/2$; or $r \leq n/2 - d$. (Figure 4(a)). Consider the latter inequality. It means that $d \pm r$ does not extend beyond the diameter $n/2$, so that the $p'_{\pm i}$ points and x' all fit inside a semicircle, as in (a) of the figure. Then $d(x', p'_{\pm i}) = md \pm i$ or $md \mp i$, depending on whether the path $x \to p$ or $p \to x$ is shorter, respectively. So, what was the distance d in P between x and y=p becomes the distance set $\{md - r, \ldots, md - 1, md, md + 1, \ldots, md + r\}$ in P'. If d is the diameter $n/2$, then the distance set is $\{md, md-1, md-1, md-2, md-2, ..., md-r, md-r\}$.

   (b) $r > n/2 - d$ (Figure 4(b)). Here $d \pm r$ does extend beyond the diameter $n/2$, at which point its increase or decrease reverses direction. In (b) of the figure, the new distance set is $\{md - 2, md - 1, md, md + 1, md\}$.

3. Both x' and y' are among the p'_i. Here we get a clique of new distances among the p'_i.

In any of these three cases, call the new distance set $D = \{d(x', p'_{\pm i}) \ : \ i = 0, \ldots, r\}$.

So now we see, in the $P \to P'$ transition, either the change $d \to md$ or $d \to D$. But we see exactly the same distance changes in the $Q \to Q'$ transition. For the distances not involving q are stretched by m, and the distances involving q'_i get stretched by $m \pm i$, $i = 0, \ldots, r$. Because P and Q are homometric, all the former changes are identical between them, and because p and q are isospectral, all the latter changes are identical between them. Even in the case where the inflation of (x, p) crosses a diameter in the $P \to P'$ inflation, there is a point $z \in Q$ that achieves the same distance $d(x, p) = d(z, q)$ (because p and q are isospectral), so the crossing-diameter behavior, and the distance set D, is exactly mirrored in the $Q \to Q'$ transition.[2] Therefore, P' and Q' are homometric.

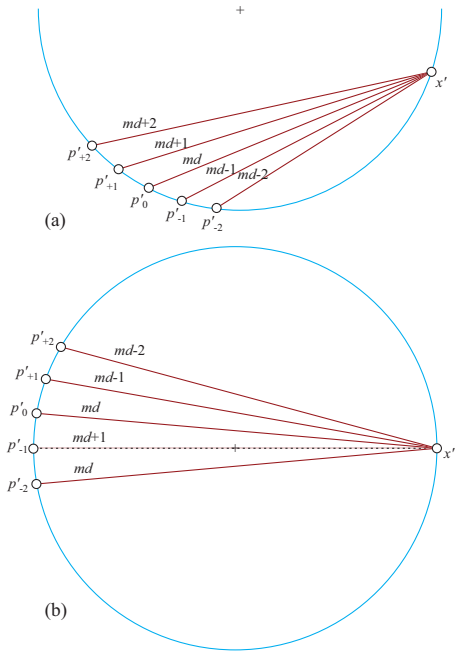


Figure 4: (a) The inflation fits inside a semicircle: r=2, new distances $\{md - r, \ldots, md - 1, md, md + 1, \ldots, md + r\}$. (b) The inflation crosses a diameter, here at (x', p'_{-1}).

The counterparts of p and q, p'_0 and q'_0, are isospectral in P' and Q', because their distance spectra are simply scaled by m (most clearly seen in Figure 3).

We turn now to the mono- and polyphonic claims of the lemma. It should be clear that if we inflate by $m \geq r + 1$, then the closest vertices, separated by 1 in P, become separated by $\geq r + 1$ in P', which is enough to accommodate the addition of r new vertices to each side of p. (If the closest vertices are separated by more than 1, then even smaller inflation will avoid overlap.) Continuing our example, inflation by m=r+1=3 suffices to avoid overlap and so maintain a monophonic rhythm, as illustrated in Figure 3.

When $m \leq r$, there could be overlap of the newly added vertices on top of the old vertices. So the resulting rhythm may be polyphonic. However, the rhythms are still homometric, where we treat vertices with multiplicity more than 1 as if they were distinct vertices (and distance 0 is ignored). This is illustrated in Figure 5,

[2] An instance of this behavior is illustrated in Figure 5 below, where the set D for segments $(7, 0) \in P$ and $(0, 5) \in Q$ have inflated distance set $D = \{3, 4, 5, 6, 5\}$.

where we have used m=1, i.e., n'=n. □

The inspiration for the transformation described in this lemma is Property 7 in [AG00], which similarly inflates a particular pair of homometric quadrilaterals by replacing a vertex in each by a sequence of vertices, and increasing n to accommodate. However, their inflation does not rely on isospectral vertices, and appears to only work on that specific quadrilateral pair.

Corollary 2 *From any pair of rhythms satisfying the preconditions of the pumping lemma, we can generate an infinite sequence of increasingly larger homometric pairs.*

Proof. Because (P', Q') again contain an isospectral pair, a pumped pair can be pumped again. □

Given the preconditions of the pumping lemma, it would be useful to characterize the homometric pairs of rhythms that contain an isospectral pair of vertices.

4 Polyphonic Rhythms

As mentioned in the proof above, if we do not pump n enough to accommodate the pumping of k without overlap, i.e., when $m \leq r$, an (m, r)-pumping may convert a monophonic rhythm to a polyphonic rhythm. In general, vertices of a rhythm have integer weights representing their multiplicity. In Figure 5, two vertices have weight 2 whereas all others have weight 1. The interval histogram still makes sense by treating a vertex of weight w as w-distinct colocated vertices. For example, in the histogram of the first rhythm, the distance 6 is achieved three times: by $(10, 4)$ and twice by $(7, 1)$ because vertex 1 has weight 2. And the pumping lemma still guarantees homometricity.

One could interpret onsets of weight greater than 1 as representing greater emphasis, or several drums with different timbre, or several voices sounded in unison in the pitch model, each an octave apart from the others (an elementary form of harmony). Homometricity in polyphonic rhythms is an apparently unexplored topic, which we believe opens new directions for research in music theory. For example, we have established that the hexachordal theorem extends to polyphonic rhythms (and beyond) [BBGM$^+$08]. Shellings of polyphonic rhythms are also a natural topic of investigation.

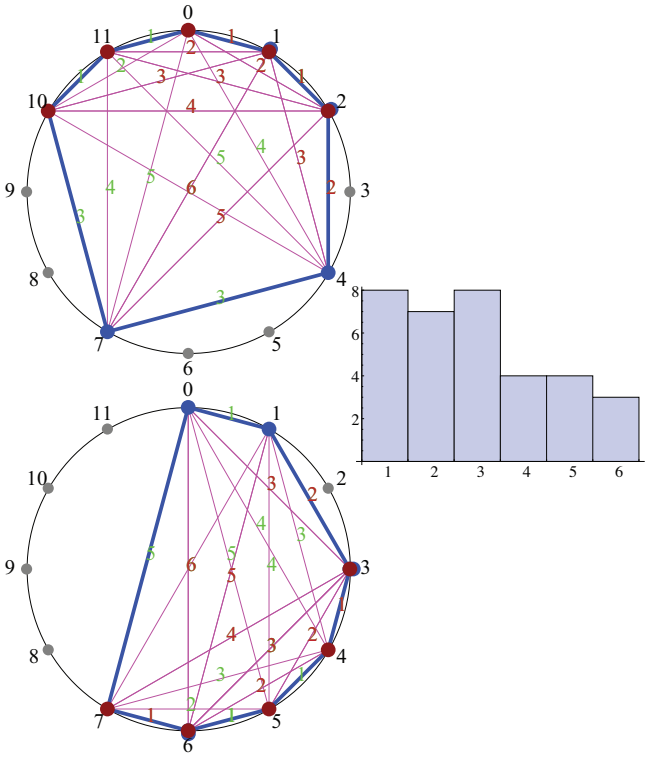


Figure 5: Fig. 1 pumped with m=1, r=2, n'=n=12. The rhythm is polyphonic: $\{1, 2\}$ in the first rhythm, and $\{3, 6\}$, in the second, have multiplicity 2.

References

[Ank97] W. Anku. Principles of rhythm integration in African music. *Black Music Research Journal*, 17(2):211–238, Autumn 1997.

[BBGM$^+$08] B. Ballinger, N. Benbernou, F. Gomez-Martin, J. O'Rourke, and G. Toussaint. The continuous hexachordal theorem. Manuscript in preparation, 2008.

[Bue78] M. J. Buerger. Interpoint distances in cyclotomic sets. *The Canadian Mineralogist*, 16:301–314, 1978.

[DGMM$^+$08] E. D. Demaine, F. Gomez-Martin, H. Meijer, D. Rappaport, P. Taslakian, G. Toussaint, T. Winograd, and D. Wood. The distance geometry of music. *Comput. Geom. Theory Appl.*, 2008. To appear.

[Lew59] D. Lewin. Intervallic relations between two collections of notes. *Journal of Music Theory*, 3(2):298–301, November 1959.

[Pat44] A. L. Patterson. Ambiguities in the X-ray analysis of crystal structures. *Physical Review*, 64(5-6):195–201, March 1944.

[Tou05] G. T. Toussaint. The geometry of musical rhythm. In J. Akiyama et al., editor, *Proceedings of the Japan Conference on Discrete and Computational Geometry*, volume LNCS 3742, pages 198–212, Berlin, Heidelberg, 2005.

[Tym06] D. Tymoczko. The geometry of musical chords. *Science*, 313(72):72–74, July 7 2006.

[Wri78] O. Wright. *The Modal System of Arab and Persian Music AD 1250-1300*. London Oriental Series 28. Oxford University Press, 1978.

[Aga86] V. K. Agawu. Gi Dunu, Nyekpadudo, and the study of West African rhythm. *Ethnomusicology*, 30(1):64–83, Winter 1986.

Maximal Covering by Two Isothetic Unit Squares

Priya Ranjan Sinha Mahapatra * Partha P. Goswami * Sandip Das †

Abstract

Let P be the point set in two dimensional plane. In this paper, we consider the problem of locating two isothetic unit squares such that together they cover maximum number of points from P. In case of overlapping, the points in their common zone are counted once. To solve the problem, we propose an algorithm that runs in $O(n^2 \log^2 n)$ time using $O(n \log n)$ space.

1 Introduction

Encloser problems of many variations involving a point set $P = \{p_1, p_2, \ldots, p_n\}$ have been extensively studied in computational geometry. Problems of computing smallest enclosing circle [15], triangle [4, 11, 14], square and rectangle [17] are well known. The problem of finding the smallest enclosing convex polygon is the famous convex hull problem.

Finding the smallest region of given type that contains k points of P, that is, the problem of computing smallest k enclosing region is an important variation of enclosure problem. Efrat et al. [9, 12] studied the problem of computing smallest k-enclosing circle and k-enclosing homothetic copy of a given convex polygon. Eppstein and Erickson [10] studied a number of extensions including finding subsets of size k from the given set P that minimize area, perimeter, diameter, and circumradius. Problems of computing k-enclosing rectangles and squares are also studied [1, 5, 8, 10, 16] extensively.

A closely related problem is to find the placement of one or more copies of a given region to maximize the size k of the subset covered. In other words, instead of fixing k and computing an optimal enclosing region, the problem is to maximize the number of points covered by the given region(s) of fixed size and shape. This type of problem has similar applications as the problems mentioned above. These so called problems of maximal covering by convex objects has also received attention of many researchers. Barequet et al. [2] proposed an algorithm to cover maximum number of points from a planar point set P by a given convex polygon with m vertices in $O(nk \log(mk) + m)$ time using $O(m + n)$ space. In the context of bichromatic planar point set, Diaz-Banez et al. [7] proposed algorithms for maximal covering by two disjoint isothetic unit squares and circles in $O(n^2)$ and $O(n^3 \log n)$ time respectively. They later improved the complexities to $O(n \log n)$ and $O(n^{8/3} \log^2 n)$ time respectively [6]. The optimal $O(n \log n)$ time algorithm for the maximal covering by two disjoint isothetic unit squares was proposed by Mahapatra et al. [13].

In this paper we consider another natural variation of the maximal covering problem. We study the problem of computing two isothetic unit squares, which may not be disjoint, such that together they cover maximum number of points from P. In case they are overlapping, points in their common zone are counted once. Our proposed algorithm for the problem runs in $O(n^2 \log^2 n)$ time and uses $O(n \log n)$ space.

2 Overview

Let P be the set of n points in two dimensional plane and R_1 and R_2 be two isothetic unit squares. Our objective is to compute placement of R_1 and R_2 such that the number of points in the region $R_1 \cup R_2$ is maximized. Number of points contained by a region R is denoted by $|R|$. In the optimal placement, following cases may occur.

- The squares are disjoint.
- The squares are overlapping and the common zone is empty.
- The squares are overlapping and the common zone is nonempty.

In case the optimal solution belongs to first two cases, the solution can be reported using the algorithms proposed by Mahapatra et al. [13]. When the optimal solution is overlapping, our proposed algorithm returns the optimal pair. We must concentrate in locating the optimal pair of squares on the region where there is a possibility of overlap between two squares. Note that, a pair of squares having same top boundary or same left boundary cannot be the optimal pair. In case R_1 and R_2 have an overlapping region, then depending upon the position of the overlapped region, the placement of R_1 and R_2 can be classified into two types as depicted in Figure 1. We tackle each case separately.

3 Characterization

Consider two arrays L_x and L_y containing the points of P in ascending order with respect to their x- and y-coordinates respectively. Let the x-coordinate of the i-th entry of L_x be x_i and similarly the y-coordinate of

*University of Kalyani, Kalyani, India
†Indian Statistical Institute, Kolkata, India

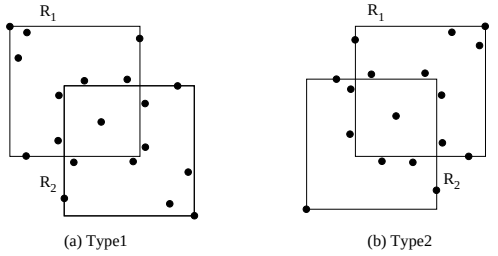


Figure 1: Placement of R_1 and R_2

the i-th entry of L_y be y_i, $1 \leq i \leq n$. We use the implicit grid obtained by drawing vertical and horizontal lines through each point of the given set P. The grid point (x_i, y_j), $(1 \leq i, j \leq n)$ is generated by the intersection of the vertical line through the point in the i-th entry of L_x and the horizontal line through the point in the j-th entry of L_y. The coordinate of a generic point p is denoted by (p_x, p_y).

A square can be specified using its top left corner. Here, $S(i, j)$ denotes a square whose top left corner is at (x_i, y_j). Consider a horizontal line l_α on the grid having y-coordinate y_α above the square $S(i, j)$ for some i, j, $1 \leq i, j \leq n$. Initially we are trying to identify a square S' whose top boundary is aligned with l_α such that the square S' together with $S(i, j)$ cover maximum number of points and whenever they overlap, configuration of overlapping region is of Type-1 (See Figure 1(a)). Hence we can obtain the optimal pair by choosing all possible $S(i, j)$ and α. Note that, the placement of the upper square S' depends upon the choice of α and the lower square $S(i, j)$. Upper square S' has the following characteristics.

(1) Given α and a lower square $S(i, j)$, the upper square S' belongs to the set $\{S(1, \alpha), S(2, \alpha), \ldots, S(i, \alpha)\}$
(2) If $|S(a, \alpha)| \geq |S(b, \alpha)|$ where $a < b \leq i$ then $|S(a, \alpha) \cup S(i, j)| \geq |S(b, \alpha) \cup S(i, j)|$.

3.1 Matching

Given an index α, the matching of $S(i, j)$ with respect to α $(\alpha \geq j)$ is defined as a square $S(k, \alpha)$ such that $|S(k, \alpha) \cup S(i, j)| > |S(k', \alpha) \cup S(i, j)|$ for $k' = 1, 2, \ldots k-1$ and $|S(k, \alpha) \cup S(i, j)| \geq |S(k', \alpha) \cup S(i, j)|$ for $k' = k+1, k+2, \ldots i$.

Lemma 1 *For a given α and j, $\alpha \geq j$, let the matching of $S(b, j)$ be $S(k, \alpha)$ and the matching of $S(c, j)$, $b < c \leq n$ be $S(k', \alpha)$. Then $k \leq k'$.*

Proof. As the matching of $S(b, j)$ is $S(k, \alpha)$, $|S(t, \alpha) - S(b, j)| < |S(k, \alpha) - S(b, j)|$ for $t = 1, 2, \ldots, k-1$. Again $(S(b, j) - S(c, j)) \cap S(t, \alpha) \subseteq (S(b, j) - S(c, j)) \cap S(k, \alpha)$. This implies $|S(t, \alpha) - S(c, j)| = |S(t, \alpha) - S(b, j)| + |(S(b, j) - S(c, j)) \cap S(t, \alpha)| < |S(k, \alpha) - S(b, j)| + |(S(b, j) - S(c, j)) \cap S(k, \alpha)| = |S(k, \alpha) - S(c, j)|$ for $t = 1, 2, \ldots, k-1$. Hence the result follows. □

Lemma 2 *Given indices α and j, $\alpha \geq j$, let the matching of both $S(a, j)$ and $S(b, j)$ be $S(k, \alpha)$, $1 \leq a, b \leq n$. If $a < b$ then the matching of each $S(i, j)$, $a \leq i \leq b$ is $S(k, \alpha)$.*

Proof. Let the matching of $S(i, j)$ be $S(k', \alpha)$ for some i, $a < i < b$. Since the matching of $S(a, j)$ is $S(k, \alpha)$ and $a < i$, from Lemma 1, we get $k \leq k'$. Similarly, as the matching of $S(b, j)$ is $S(k, \alpha)$ and $i < b$ so $k' \leq k$. This implies $k = k'$. □

Given α and j $(\alpha > j)$, for computing matching of all $S(i, j)$, $1 \leq i \leq n$, we can reduce the search space from the fact stated in Lemma 1. Observe that the matching of $S(i, j)$, for all i can be computed in $O(n \log n)$ time. This implies, for a given α, the matching of all $S(i, j)$'s, $1 \leq i \leq n$ and $\alpha > j$, can be computed in $O(n^2 \log n)$ time. Hence the matching of all $S(i, j)$'s, $1 \leq i \leq n$ and $1 \leq j \leq n$, can be computed in $O(n^3 \log n)$ time. In this paper, we propose an algorithm to compute the matching of all possible squares in $O(n^2 \log^2 n)$ time. Below we describe some more characterizations to achieve the sub-cubic complexity of the proposed algorithm.

From Lemma 2, we conclude that for given α and j $(\alpha \geq j)$, there exists an interval $[a, b]$ such that all $S(i, j)$'s for $a \leq i \leq b$ are matched with $S(k, \alpha)$ and the matching of each $S(i, j)$ for $b < i \leq n$ or $1 \leq i < a$ is different from $S(k, \alpha)$. Here we denote such an interval $[a, b]$ using notation $\mathcal{F}(S(k, \alpha), j)$. An interval $\mathcal{F}(S(k, \alpha), j)$ is empty whenever there does not exist any square $S(i, j)$, $1 \leq i \leq n$, whose matching is $S(k, \alpha)$.

Observation 1 *(a) Intervals $\mathcal{F}(S(k, \alpha), j)$ and $\mathcal{F}(S(k', \alpha), j)$ are disjoint for $k \neq k'$, $1 \leq k, k' \leq n$. (b) Moreover, if both the intervals are not $\emptyset$ and $k < k'$ then interval $\mathcal{F}(S(k', \alpha), j)$ is on the right side of the interval $\mathcal{F}(S(k, \alpha), j)$. (c) $\bigcup_{k=1}^{n} \mathcal{F}(S(k, \alpha), j) = [1, n]$.*

Proof. *(a)* Note that for a given α and j, the matching of $S(i, j)$, $1 \leq i \leq n$ is unique and hence $\mathcal{F}(S(k, \alpha), j) \cap \mathcal{F}(S(k', \alpha), j) = \emptyset$ for $k \neq k'$. Statement *(b)* follows from Lemma 1. □

Lemma 3 *For a given α and j, $\alpha > (j+1)$, if the matchings of $S(i, j)$ and $S(i, j+1)$ are $S(k, \alpha)$ and $S(k', \alpha)$ respectively then $k \geq k'$.*

Proof. As the matching of $S(i, j)$ is $S(k, \alpha)$, $|S(t, \alpha) - S(i, j)| \leq |S(k, \alpha) - S(i, j)|$ for $t = 1, 2, \ldots, k-1, k, k+1, \ldots, i$. Again $S(t, \alpha) \cap S(i, j) \supset S(k, \alpha) \cap S(i, j)$ for $t = k+1, \ldots, i$ and $S(t, \alpha) \cap S(i, j+1) - S(i, j) \supset S(k, \alpha) \cap S(i, j+1) - S(i, j)$ for $t = k+1, \ldots, i$. Hence, we have $|S(t, \alpha) - S(i, j+1)| \leq |S(k, \alpha) - S(i, j+1)|$ for $t = k+1, \ldots, i$. □

Observation 2 *For a given α and j, $\alpha > (j+1)$, let the matching of $S(i, j)$ be $S(k, \alpha)$ and $(S(i, j+1) \cap S(k, \alpha)) - S(i, j) = \emptyset$. Then the matching of $S(i, j+1)$ is $S(k, \alpha)$.*

Proof. Matching of $S(i,j)$ is $S(k,\alpha)$ and therefore, $|S(k,\alpha)-S(i,j)| > |S(t,\alpha)-S(i,j)|$ for $t = 1,2,\ldots k-1$, and $|S(k,\alpha) - S(i,j)| \geq |S(t,\alpha) - S(i,j)|$ for $t = k+1, k+2, \ldots i$. Now, $(S(i,j+1) \cap S(k,\alpha)) - S(i,j) = \emptyset$ implies $|S(k,\alpha) - S(i,j+1)| = |S(k,\alpha) - S(i,j)|$. Hence, $|S(k,\alpha) - S(i,j+1)| > |S(t,\alpha) - S(i,j)| \geq |S(t,\alpha) - S(i,j+1)|$ for $t = 1,2,\ldots k-1$, and $|S(k,\alpha) - S(i,j+1)| \geq |S(t,\alpha) - S(i,j+1)|$ for $t = k+1, k+2, \ldots i$. Hence the result follows. □

Lemma 4 *For $\mathcal{F}(S(k,\alpha),j) \neq \emptyset$, $\alpha > j+1$, either one of the following is true.*

(1) $\mathcal{F}(S(k,\alpha),j+1) = \emptyset$

(2) $\mathcal{F}(S(k,\alpha),j+1) = \mathcal{F}(S(k,\alpha),j)$

(3) $\mathcal{F}(S(k,\alpha),j+1) \subset \mathcal{F}(S(k,\alpha),j)$

(4) $\mathcal{F}(S(k,\alpha),j+1) \supset \mathcal{F}(S(k,\alpha),j)$

There exists at most one value of k, $1 \leq k \leq n$, that satisfy Case (3) *and similarly, there exist at most one value of k, $1 \leq k \leq n$, that satisfy Case* (4)*.*

Proof. Let $p(x_m, y_{j+1})$ be a point in P. Suppose $\mathcal{F}(S(k,\alpha),j) = [a,b]$. If $|(S(i,j+1) \cap S(k,\alpha)) - S(i,j)| = 0$ for some i's, $a \leq i \leq b$, then $|(S(r,j+1) \cap S(k,\alpha)) - S(r,j)| = 0$ for $r = i, i+1, \ldots, b$. In this case, from Observation 2, the matching of $S(r,j+1)$ is $S(k,\alpha)$. Therefore, if $p \notin S(k,\alpha)$ then $i = a$ and $\mathcal{F}(S(k,\alpha),j+1) \supseteq \mathcal{F}(S(k,\alpha),j)$. Moreover, if matching of $S(b+1,j)$ is $S(k',\alpha)$ and $|(S(b+1,j+1) \cap S(k',\alpha)) - S(b+1,j)| = 0$, the matching of $S(b+1,j+1)$ is $S(k',\alpha)$ and then it implies $\mathcal{F}(S(k,\alpha),j+1) = \mathcal{F}(S(k,\alpha),j)$.

Suppose the matching of $S(i,j)$ and $S(i,j+1)$ are $S(k,\alpha)$ and $S(k',\alpha)$ respectively. From Lemma 3, $k' \leq k$. Let p be in $((S(i,j+1) \cap S(k,\alpha)) - S(i,j)$. If $\mathcal{F}(S(k',\alpha),j) = (c,d]$ and $k' < k$, then $p \notin S(k',\alpha)$ and the matching of $S(r,j+1)$ is $S(k',\alpha)$ for $r = d+1, d+2, \ldots i$. Here, $\mathcal{F}(S(k,\alpha),j+1) \subset \mathcal{F}(S(k,\alpha),j)$ and $\mathcal{F}(S(k',\alpha),j+1) \supset \mathcal{F}(S(k',\alpha),j)$. In case, $i = b$, $\mathcal{F}(S(k,\alpha),j+1) = \emptyset$. Hence the lemma follows. □

Suppose the matching of $S(i-1,j)$ and $S(i,j)$ are $S(k',\alpha)$ and $S(k,\alpha)$ respectively with $k' < k$. Let the point $p(x_m, y_{j+1})$ lie inside the region $((S(i,j+1) \cap S(k,\alpha)) - S(i,j)$ but not inside the region $((S(i-1,j+1) \cap S(k',\alpha)) - S(i-1,j)$. Then using similar arguments as in the proof of Lemma 4, $\mathcal{F}(S(k',\alpha),j+1) \supseteq \mathcal{F}(S(k',\alpha),j)$ and for all $k'' < k'$, $\mathcal{F}(S(k'',\alpha),j+1) = \mathcal{F}(S(k'',\alpha),j)$. If $\mathcal{F}(S(k',\alpha),j+1) = \mathcal{F}(S(k',\alpha),j)$, then $\mathcal{F}(S(r,\alpha),j+1) = \mathcal{F}(S(r,\alpha),j)$ for $r = 1,2,\ldots,n$. When $\mathcal{F}(S(k',\alpha),j+1) \supset \mathcal{F}(S(k',\alpha),j)$, there exist an integer v, $0 \leq v \leq n$, such that the matching of $S(v,j+1)$ is $S(k',\alpha)$ but the matching $S(v+1,j+1)$ is not $S(k',\alpha)$. Suppose the matching of $S(v+1,j)$ is $S(h,\alpha)$, then from Lemma 4 along with similar arguments, we can conclude that $\mathcal{F}(S(r,\alpha),j+1) = \emptyset$ for $r = k'+1, k'+2, \ldots, h-1$, $\mathcal{F}(S(h,\alpha),j+1) \subseteq \mathcal{F}(S(h,\alpha),j)$ and $\mathcal{F}(S(r,\alpha),j+1) = \mathcal{F}(S(r,\alpha),j)$ for $r = h+1, h+2, \ldots, n$.

Observation 3 *For given indices α and β, $\alpha \geq \beta$, with $(y_\alpha - y_\beta) \geq 1$, all nonempty $\mathcal{F}(S(k,\alpha),\beta)$ can be computed in linear time.*

4 Algorithm

Here, we are looking for a pair of overlapping squares where top boundary of the upper square (R_1) is at y_α and top left corner of the lower square (R_2) is inside the interior of upper square such that together they cover maximum number of points of P. Note that R_1 and R_2 may together contain less number of points than a non-overlapping pair with top boundary of upper square at y_α. Below, we describe the algorithm to report the maximum number of points covered by a pair of squares with top boundary of the upper square at y_α and the top boundary of the lower square is above the bottom boundary of the upper square in $O(n \log^2 n)$ time.

Let the function $left(p_{x_i})$ output the minimum entry in L_x, say p_{x_j}, such that $x_i - x_j$ is less than or equal to unity. Then we can compute $left(p_{x_i}), 1 \leq i \leq n$ in $O(\log n)$ time. Note that $left(p_{x_1}) = p_{x_1}$. In a similar way, we define $right(.)$, $bottom(.)$, and $top(.)$. For orthogonal range searching on a given points set P in $2D$, we construct range tree R [3] that reports the number of points in a query rectangle in $O(\log n)$ time. The construction time and the space for range tree R are both $O(n \log n)$.

4.1 Data structure and initialization

Let y_γ be the $bottom(y_\alpha)$ $(1 \leq \alpha \leq n)$. Consider an index β such that $|y_\alpha - y_\beta| \geq 1$ and compute $\mathcal{F}(S(k,\alpha),\beta)$ for all k, $1 \leq k \leq n$, which are nonempty. Using observation 3, this computation can be done in linear time. Let the intervals be $[a_1,b_1],[a_2,b_2],\ldots,[a_\nu,b_\nu]$ from left to right such that $\mathcal{F}(S(k_i,\alpha),\beta) = [a_i,b_i]$ for $i = 1,2,,\ldots,\nu$.

Now we initialize β by γ and subsequently the value β varies from $\gamma+1, \gamma+2, \ldots, \alpha-1$.

Consider a balanced binary search tree T constituted by $x_1, x_2, \ldots, x_n$, where each value x_i corresponds to a leaf node, and the search is guided by the x-values. The leaf node corresponding to x_i keeps information about $|S(i,\beta) \cup S(k,\alpha)|$, where the matching of $S(i,\beta)$ is $S(k,\alpha)$. Two counter variables $\mathcal{M}$ and $\mathcal{C}$ are attached with nodes of T for computing the number of points covered by $S(i,\beta)$ along with its matching square for all i. To start with, the variables $\mathcal{M}$ and $\mathcal{C}$ corresponding to all nodes are initialized with zero. Given an interval, *Increment* operation modifies T such that count of all squares with top-left corner within the interval is incremented by one. Similarly, we can define the *Decrement*

operation. Detail algorithms for *Increment* and *Decrement* operations are discussed by Mahapatra et al. [13]. At any instance with α and β, given a lower square X, *Report* operation [13] is able to report the maximum number of points covered by a pair of squares where the lower square must be X and the upper square have the top boundary at y_α. It can also report the number of maximum covered points of a pair of squares whose top boundaries are aligned with y_α and y_β.

Construct another balanced binary search tree T' constituted by disjoint intervals $[x_{a_1}, x_{b_1}]$, $[x_{a_2}, x_{b_2}]$, ..., $[x_{a_\nu}, x_{b_\nu}]$ as leaf nodes. A variable W is attached with each leaf node to keep information of a square. All squares having left boundary within the interval corresponding to that leaf node are matched with W. Given a point p, we can report the interval containing p in $O(\log n)$ time. Insertion, deletion and update of an interval can be done in $O(\log n)$ time. We now describe main steps of the algorithm.

For $\beta = \gamma + 1, \gamma + 2, \ldots, \alpha - 1$, execute the following steps:

1. Suppose the β-th entry of L_y be the point p with coordinates, say (x_m, y_β).
2. Find the interval $[a, b]$ from T' containing p and let the corresponding square be $S(k, \alpha)$.
3. Find the inorder-predecessor of $S(k, \alpha)$, say $S(k', \alpha)$ and the corresponding interval is $[a', b']$.
   - 3.1 $p \notin S(k, \alpha)$: Perform *Increment* operation for the interval $[left(x_m), x_m]$ into T.
   - 3.2 $p \in S(k, \alpha)$ and $p \notin S(k', \alpha)$: Compute $|S(a, \beta) \cup S(k, \alpha)|$ and $|S(a, \beta) \cup S(k', \alpha)|$ by rectangular range searching from range tree R. If $|S(a, \beta) \cup S(k', \alpha)| \geq |S(a, \beta) \cup S(k, \alpha)|$,
     - 3.2.1 the matching of $S(a, \beta)$ is $S(k', \alpha)$;
     - 3.2.2 use divide and conquer method within interval $[a, b]$, select an index c such that $\mathcal{F}(S(k, \alpha), \beta)$ is $[c, b]$;
     - 3.2.3 update the interval $[a, b]$ in T' with $[c, b]$ and $[a', b']$ with $[a', c - 1]$;
     - 3.2.4 perform *Increment* operation for the interval $[left(x_m), x_{c-1}]$ and update the current optimal count with the value at the root of T.
   - 3.3 $p \in S(k, \alpha)$ and $p \in S(k', \alpha)$: Identify k'' such that $p \notin S(k'', \alpha)$ but the inorder successor $S(l, \alpha)$ of $S(k'', \alpha)$ contains p. Go to Step 3.2 with $k \leftarrow l$, $k' \leftarrow k''$.

Finally, the count of the optimal pair with top boundary of the upper square at y_α can be obtained at the root of T and the corresponding pair can be identified.

4.2 Complexity of the Algorithm

Theorem 5 *Given a set P of n points in $\mathcal{R}^2$, two disjoint or overlapping isothetic unit squares covering maximum number of points can be found in $O(n^2 \log^2 n)$ time using $O(n \log n)$ space.*

References

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12, 38–56, 1991.

[2] Gill Barequet, Matthew Dickerson, and Petru Pau, Translating a convex polygon to contain a maximum number of points. *Computational Geometry Theory and Applicaions*, 8, 167–179, 1997.

[3] M. de Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf, Computational Geometry, Algorithms and Applications. *Springer*, Berlin, 1997.

[4] S. Chandran and D. Mount, A parallel algorithm for enclosed and enclosing triangles. *International Journal of Computational Geometry and Applications*, 2, 191–214, 1992.

[5] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid, Static and dynamic algorithms for k-point clustering problems. *Journal of Algorithms*, 19, 474–503, 1995.

[6] Sergio Cabello, J. Miguel Diaz-Banez, Carlos Seara, J. Antoni Sellares, Jorge Urrutia, Inmaculada Ventura, Covering point sets with two disjoint disks or squares. *Computational Geometry Theory and Applicaions*, 40, 195–206, 2008.

[7] J. Miguel Diaz-Banez, Carlos Seara, J. Antoni Sellares, Jorge Urrutia, and Imma Ventura, Covering Points Sets with Two Convex Objects. *Proc. 21st European Workshop on Computational Geometry*, 2005.

[8] Sandip Das, Partha P. Goswami, and Subhas C. Nandy, Smallest k-point enclosing rectangle and square of arbitrary orientation. *Information Processing Letters*, 95, 259–266, 2005.

[9] A. Efrat, M. Sharir, and A. Ziv Computing the smallest k-enclosing circle and related problems. *Computational Geometry*, 4, 119–136, 1994.

[10] D. Eppstein and J. Erickson, Iterated nearest neighbors and finding minimal polytopes. *Discrete and Computational Geometry*, 11, 321–350, 1994.

[11] V. Klee and M.L. Laskowski, Finding the smallest triangles containing a given convex polygon. *Journal of Algorithms*, 6, 359–375, 1985.

[12] J. Matousek On geometric optimization with few violated constraints. *Discrete and Computational Geometry*, 14, 365–384, 1995.

[13] Priya Ranjan Sinha Mahapatra, Partha P. Goswami, and Sandip Das, Covering Points by Isothetic Unit Squares. *Proc. 19th Canadaian Conference on Computational Geometry*, 169–172, 2007.

[14] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, An optimal algorithm for finding minimal enclosing triangles. *Journal of Algorithms*, 7, 258–269, 1986.

[15] F.P. Preparata and M.I. Shamos, Computational Geometry: An Introduction. *Springer*, Berlin, (1985).

[16] M. Segal and K. Kedem, Enclosing k points in smallest axis parallel rectangle. *Information Processing Letters*, 65, 95–99, 1998.

[17] G.T. Toussaint, Solving geometric problems with the rotating calipers. *Proc. IEEE MELECON*, 1983.

Triangulating and Guarding Realistic Polygons

G. Aloupis * P. Bose † V. Dujmovic ‡ C. Gray § S. Langerman ¶ B. Speckmann §

Abstract

We propose a new model of realistic input: *k-guardable* objects. An object is k-guardable if its boundary can be seen by k guards in the interior of the object. In this abstract, we describe a simple algorithm for triangulating k-guardable polygons. Our algorithm, which is easily implementable, takes linear time assuming that k is constant.

1 Introduction

Algorithms and data structures in computational geometry often display their worst-case performance on intricate input configurations that seem artificial or unrealistic when considered in the context of the original problem. In "practical" situations, many algorithms and data structures—binary space partitions are a notable example—tend to perform much better than predicted by the theoretical bounds. An attempt to understand this disparity and to quantify "practical" or "normal" with respect to input are *realistic input models* [5]. Here one places certain restrictions on the shape and/or distribution of the input objects so that most, if not all, hypothetical worst-case examples are excluded. Analyzing the algorithm or data structure in question under these input assumptions tends to lead to performance bounds that are much closer to actually observed behavior.

Many realistic input models have been proposed. These include *low-density* scenes [5], where it is assumed that the number of "large" objects intersecting a "small" volume is bounded, and *local* polyhedra [7], where it is assumed that the ratio of lengths between edges coming from a single vertex is limited by a constant. One of the most widely studied realistic input models assumes that input objects are *fat*, that is, they are not arbitrarily long and skinny. There are several ways to characterize fat objects—see the full paper for formal definitions.

We propose a new model defining realistic input: the number of guards that are required to see the boundary of an input object. We use the term *k-guardable* to denote any object whose boundary can be seen by k guards. A rigorous definition of what it means for a guard to *see* can be found in the next section.

In the full paper, we discuss the connection between k-guardable polygons and other measures of realistic input. In particular, we show that (α, β)-*covered* polygons are k-guardable. An (α, β)-covered polygon is a type of fat polygon designed to model the intuitive notion of fatness for non-convex input. This is a type of polygon P that has the property that every point $p \in \partial P$ admits a triangle with minimum angle at least α and minimum edge length at least $\beta \cdot \mathrm{diam}(P)$ for given constants α and β. In the full paper we prove:

Theorem 1 *The boundary of any (α, β)-covered polygon can be guarded with $\lceil 32\pi/(\alpha\beta^2) \rceil$ guards.*

In this abstract we describe an algorithm for triangulating k-guardable polygons. Our algorithm, which was designed with simplicity in mind, takes $O(kn)$ time, that is, linear time assuming that k is constant. We also show that, if the *link diameter*—see the next section for a formal definition—of the input polygon is d, this algorithm takes $O(dn)$ time—a slightly stronger result. In the full paper we describe a second algorithm which also triangulates k-guardable polygons in $O(kn)$ time. That algorithm uses even easier subroutines than the one given in this abstract, but it requires the actual guards as input, which might be undesirable in certain situations.

In 1991 Chazelle [2] presented a linear time algorithm to triangulate any simple polygon. However, after all these years it is still a major open problem in computational geometry to design an implementable linear-time algorithm for triangulation. There are several implementable algorithms which triangulate polygons in near-linear time. For example, Kirkpatrick *et al.* [11] describe an $O(n \log\log n)$ algorithm and Seidel [15] presents a randomized algorithm which runs in $O(n \log^* n)$ expected time. We contend that our algorithm is conceptually simpler than the $O(n \log\log n)$ algorithm and that it has a slight advantage over the Seidel algorithm because it is deterministic. It is also interesting to note that the Seidel algorithm requires $\Omega(n \log n)$ random bits, which makes it theoretically undesirable in some models.

*Carleton University & Université Libre de Bruxelles, greg@cg.scs.carleton.ca.

†Carleton University, jit@cg.scs.carleton.ca.

‡McGill University, vida@cs.mcgill.ca.

§TU Eindhoven, {cgray,speckman}@win.tue.nl. C.G. is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301. B.S. is supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.022.707.

¶Chercheur Qualifié du FNRS, Université Libre de Bruxelles, Belgique, stefan.langerman@ulb.ac.be.

Relationships between shape complexity and the number of steps necessary to triangulate polygons have been investigated before. For example, it has been shown that monotone polygons [17], star-shaped polygons [14], and edge-visible polygons [16] can all be triangulated in linear time by fairly simple algorithms. Other measures of shape complexity that have been studied include the number of reflex vertices [9] or the sinuosity [3] of the polygon.

Several algorithms and data structures exist for collections of realistic objects. For example, the problem of ray-shooting in an environment consisting of fat objects has been studied extensively [1, 4, 10]. But there are few results concerning individual realistic objects. We hope that our results on triangulating realistic polygons will encourage further research in this direction.

The following section introduces the definitions used throughout this abstract and presents several useful tools. Section 3 describes the triangulation algorithm. We conclude in Section 4 with some open problems.

2 Tools and definitions

Throughout this paper let P be a simple polygon with n vertices. We denote the interior of P by $\text{int}(P)$, the boundary of P by ∂P, and the *diameter* of P by $\text{diam}(P)$. The boundary is considered part of the polygon, that is, $P = \text{int}(P) \cup \partial P$. We say that a point p is *in* P if $p \in \text{int}(P) \cup \partial P$.

The segment or edge between two points p and q is denoted by $\overline{pq}$. The same notation implies the direction from p to q if necessary. Two points p and q in P *see* each other if $\overline{pq} \cap P = \overline{pq}$. If p and q see each other, then we also say that p is *visible* from q and vice versa. We call a polygon P *k-guardable* if there exists a set G of k points in P called *guards* such that every point $p \in \partial P$ can see at least one point in G.

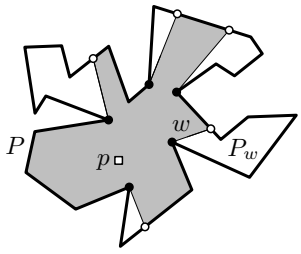


Figure 1: The visibility polygon $VP(p, P)$ is shaded. P_w is the pocket of w with respect to $VP(p, P)$.

A *star-shaped* polygon is defined as a polygon that contains a set of points—the *kernel*—each of which can see the entire polygon. If there exists an edge $\overline{pq} \subset \partial P$ such that each point in P sees some point on $\overline{pq}$, then P is *weakly edge-visible*. The *visibility polygon* of a point $p \in P$ with respect to P, denoted by $VP(p, P)$ is the set of points in P that are visible from p. Visibility polygons are star-shaped and have complexity $O(n)$.

$VP(p, P)$ can be computed in $O(n)$ time [6] with an algorithm that is non-trivial but fairly simple. It involves a single scan of the polygon and a stack. See O'Rourke's book [13] for a good summary.

A concept related to visibility in a polygon P is the *link distance*, which we denote by $ld(p, q)$ for two points p and q in P. Consider a polygonal path π that connects p and q while staying in $\text{int}(P)$. We say that π is a minimum link path if it has the fewest number of segments (links) among all such paths. The link distance of p and q is the number of links of a minimum link path between p and q. We define the *link diameter* d of P to be $\max_{p,q \in P} ld(p, q)$. The link diameter of a polygon may be much less than the number of guards required to see its boundary, and is upper bounded by the number of guards required to see the boundary. This can be seen in the so-called "comb" polygons that generally have a low link diameter but need a linear number of guards.

Let Q be a subpolygon of P, where all vertices of Q are on ∂P. If all vertices of Q coincide with vertices of P, then we call Q a *pure subpolygon*. If ∂P intersects an edge w of ∂Q only at w's endpoints, then w is called a *window* of Q. Any window w separates P into two subpolygons. The one not containing Q is the *pocket* of w with respect to Q (see Fig. 1).

The *edge-visibility polygon*, $EVP(e, P)$ of an edge e with respect to polygon P consists of all points in P that are visible from at least one point on e. We define an *extended edge-visibility polygon* of e with respect to P, denoted by $EEVP(e, P)$, to be the smallest pure subpolygon of P that contains $EVP(e, P)$. These concepts are illustrated in Figure 2.

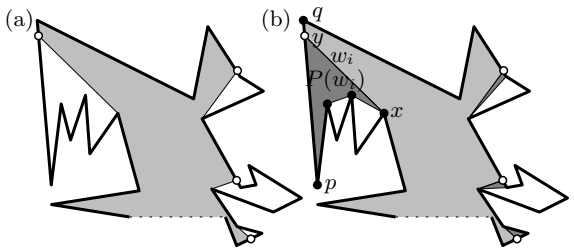


Figure 2: (a) The shaded area is the edge-visible polygon of the dashed edge; (b) the associated extended edge visible polygon.

The *geodesic* between two points in P is the shortest polygonal path connecting them that is contained in P. The vertices of a geodesic (except possibly the first and last) belong to ∂P.

Lemma 2 *Let x be a vertex of polygon P and let y be a point on edge $\overline{vw} \in P$. If y sees x, then the geodesic between x and v: (a) is a convex chain and entirely visible from y, and (b) can be computed in $O(n)$ time.*

Proof. Property (a) holds trivially if x sees v. Consider the case where x does not see v. Then, the triangle (x, y, v), denoted by T, must contain at least one vertex of P in its interior. Let I be all the vertices of P inside T and let $CH(I)$ be the convex hull of I. The path $S = CH(I) \setminus \overline{xv}$ is the geodesic from x to v. Any other path from x to v inside T can be shortened. Thus, Property (a) holds.

To prove property (b), note that since the geodesic we seek is entirely visible from y by part (a) it is fully contained in $VP(y, P)$. We compute $VP(y, P)$ in linear time. Let z be the first intersection of ∂P and the ray emanating from x in the direction $\overline{yx}$. Consider the polygonal chain from x to v along $\partial VP(y, P)$ that avoids y. By construction of $VP(y, P)$, the shortest path from x to v is part of the convex hull of this path. By Melkman's algorithm [12], the convex hull of a simple polygonal chain can be computed in linear time. □

The computation of a geodesic and of an edge-visibility polygon are the two subroutines that we use to compute the *EEVP*. Hence, we can compute the *EEVP* in linear time. Also, the *EEVP* is a very structured type of polygon—the union of an edge-visible polygon and "fan" polygons—and as such can be triangulated in linear time.

Lemma 3 *$EEVP(e, P)$ can be computed and triangulated in $O(n)$ time.*

3 Triangulating k-guardable polygons

In this section we show how to triangulate a k-guardable polygon in $O(kn)$ time. The most complicated procedure used in our algorithm is computing the visibility polygon from an edge in linear time [8]. Our algorithm relies on computing and triangulating extended edge-visibility polygons.

We begin with an arbitrary edge e of a polygon P and compute $EEVP(e, P)$. When $EEVP(e, P)$ is triangulated, the diagonals of P that are on the boundary of $EEVP(e, P)$ become windows of new pockets. Each such window serves as the edge from which a new visibility polygon will be computed and triangulated, within its respective pocket. In this recursive manner we break pockets into smaller components until all of P is triangulated. The procedure, although straightforward, is outlined below in more detail. This is followed by the analysis of the time complexity, where we show that the recursion depth is of the order of the number of guards that suffice to guard ∂P.

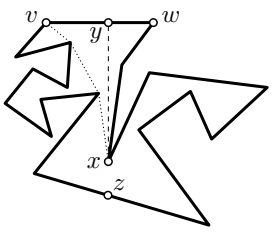


Figure 3: The geodesic from x to v.

We will maintain a queue $\mathcal{S}$ of non-overlapping polygons (pockets) such that each $P_i \in \mathcal{S}$ has one edge w_i labelled as a window. Thus elements of $\mathcal{S}$ are pairs (w_i, P_i). We start with $\mathcal{S} := (w, P)$, where w is an arbitrary boundary edge of P. We process the elements of $\mathcal{S}$ in the order in which they were inserted. The main loop of our algorithm is as follows:

TriangulateWithoutGuards(P)

```
1  S := (w, P) where w is an arbitrary edge of P
2  while S ≠ ∅
3        do for each (w_i, P_i) ∈ S
4           do remove (w_i, P_i) from S.
5              Compute and triangulate EEVP(w_i, P_i).
6              Add the edges of the triangulation to P.
7              for each boundary edge w_j of
                   EEVP(w_i, P_i) that is a diagonal of
                   P.
8                  do identify Q_j as the untriangulated
                      portion of P whose boundary is
                      enclosed by w_j and ∂P.
9     S := S ∪ {⋃_j (w_j, Q_j)}
10 return P.
```

Theorem 4 *The algorithm* TriangulateWithoutGuards *triangulates an n-vertex k-guardable polygon in $O(kn)$ time.*

Proof. We first note that a tree T is created by our algorithm. At the root of T is $EEVP(w, P)$. For every window w_j of $EEVP(w_i, P_i)$, $EEVP(w_j, P_j)$ is a child of $EEVP(w_i, P_i)$. The construction of the child nodes from their parents ensures that no *EEVP* overlaps with any other and that the triangulation covers the entire polygon P.

We now show that T has at most $3k$ levels (a *level* is a set of nodes, each of which has the same distance from the root), which implies that the main loop of the algorithm performs at most $3k$ iterations. Let ℓ_i, ℓ_{i+1}, and ℓ_{i+2} be three successive levels of T, such that all nodes in ℓ_{i+1} are descendants of nodes in ℓ_i, and all nodes in ℓ_{i+2} are descendants of nodes in ℓ_{i+1}. Suppose that a set of points G is a *guarding* of P: every point $p \in \partial P$ sees at least one guard of G. Assume, for the purpose of obtaining a contradiction, that there are no guards from G in ℓ_i, ℓ_{i+1}, and ℓ_{i+2}.

Let g be a guard which sees into a node n_i at level ℓ_i through window w_i. There are two cases: either g is in a level higher than ℓ_i or it is in a lower level. If g is in a higher level and is visible from a window of n_i, then g can be in only one level: ℓ_{i+1} (because ℓ_{i+1} contains the union of all the edge-visibility polygons of the windows of the nodes in ℓ_i). We have assumed that this can not happen. Otherwise, if g is in a lower level, it cannot see into any higher level than ℓ_i, because w_i must be the window which created n_i.

The combination of these two facts implies that no guard from G can be able to see into ℓ_{i+1}. This is a contradiction to G being a guarding set. Therefore, G must have at least one guard in ℓ_i, ℓ_{i+1}, or ℓ_{i+2}. This implies that there is at least one guard for every three levels, or at most three levels per guard.

Each level of the tree can be processed in $O(n)$ time by Lemma 3, since all nodes of a level are disjoint. Thus the algorithm terminates successfully in $O(kn)$ time. □

As is apparent from the proof of Theorem 4, our algorithm runs in $O(tn)$ time, where t is the number of iterations of the while-loop. The above argument also implies a stronger result. The number of iterations, t, of the while loop is proportional to the link diameter, d, of the polygon, since any minimum link path between two points must have at least one bend for every three levels. This leads to the following corollary:

Corollary 5 *The algorithm* TriangulateWithoutGuards *triangulates an n-vertex polygon with link diameter d in $O(dn)$ time.*

4 Open Problems

Our work raises some open problems. First, can these techniques be used to design a triangulation algorithm which does not depend on the number of guards? Second, are there other problems that can be solved efficiently for k-guardable polygons? Finally, can we find similar results for other measures of realistic input?

Acknowledgments

This research was initiated at the Carleton-Eindhoven Workshop on Computational Geometry, July 18–22, 2005, organized by Mark de Berg and Prosenjit Bose. The authors are grateful to Mark de Berg for suggesting the problems studied in this paper and Boris Aronov for many discussions. We would also like to thank an anonymous reviewer for suggesting that we consider the link diameter as a measure of polygon complexity.

References

[1] B. Aronov, M. de Berg, and C. Gray. Ray shooting and intersection searching amidst fat convex polyhedra in 3-space. In *Proc. 22nd Symposium on Computational Geometry*, pages 88–94, 2006.

[2] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6(5):485–524, 1991.

[3] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Transactions on Graphics*, 3(2):135–152, 1984.

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

[5] M. de Berg, A. F. van der Stappen, J. Vleugels, and M. J. Katz. Realistic input models for geometric algorithms. *Algorithmica*, 34(1):81–97, 2002.

[6] H. A. El Gindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2:186–197, 1981.

[7] J. Erickson. Local polyhedra and geometric graphs. *Computational Geometry: Theory and Applications*, 31:101–125, 2005.

[8] P. J. Heffernan and J. S. B. Mitchell. Structured visibility profiles with applications to problems in simple polygons. In *Proc. 6th Symposium on Computational Geometry*, pages 53–62, 1990.

[9] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proc. FCT-Conference on Fundamentals of Computation Theory*, pages 207–218, LNCS 158, Springer Verlag, 1983.

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

[11] David G. Kirkpatrick, Maria M. Klawe, and Robert Endre Tarjan. Polygon triangulation in $O(nloglogn)$ time with simple data structures. *Discrete & Computational Geometry*, 7:329–346, 1992.

[12] A. A. Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25(1):11–12, Apr. 1987.

[13] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987.

[14] A. Schoone and J. van Leeuwen. Triangulating a star-shaped polygon. Technical Report RUU-CS-80-03, Institute of Information and Computing Sciences, Utrecht University, 1980.

[15] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1:51–64, 1991.

[16] G. T. Toussaint and D. Avis. On a convex hull algorithm for polygons and its application to triangulation problems. *Pattern Recognition*, 15(1):23–29, 1982.

[17] G. T. Toussaint. A new linear algorithm for triangulating monotone polygons. *Pattern Recognition Letters*, 2:155–158, 1984.

Computing Dehn Twists and Geometric Intersection Numbers in Polynomial Time

Marcus Schaefer
Department of Computer Science
DePaul University
243 South Wabash
Chicago, Illinois 60604, USA
mschaefer@cs.depaul.edu

Eric Sedgwick
Department of Computer Science
DePaul University
243 South Wabash
Chicago, Illinois 60604, USA
esedgwick@cs.depaul.edu

Daniel Štefankovič
Department of Computer Science
University of Rochester
Rochester, New York, USA
stefanko@cs.rochester.edu

Abstract

Simple curves on surfaces are often represented as sequences of intersections with a triangulation. However, topologists have much more succinct ways of representing simple curves such as normal coordinates which are exponentially more succinct than intersection sequences.

Nevertheless, we show that the following two basic tasks of computational topology, namely performing a *Dehn-twist* of a curve along another curve, and computing the *geometric intersection number* of two curves, can be solved in polynomial time even in the succinct normal coordinate representation. These are the first algorithms that solve these problems in time polynomial in the succinct representations.

As an application we show that a generalized notion of crossing number can be decided in **NP**, even though the drawings can have exponential complexity.

1 Introduction

In an earlier paper we started investigating algorithms for basic problems of computational topology [14]; we extend this work to deal with crossings of curves in surfaces which has applications to graph drawing.

One of the driving problems of computational topology, long before it acquired the name, has been the problem of recognizing the unknot. The story begins in 1930 with Kneser [9] who introduced the succinct *normal coordinate* representation for curves and surfaces. This led to the theory of normal surfaces which was used by Haken in 1961 to show that the unknot can be recognized by an algorithm. Haken's approach was pushed further by Hass, Lagarias, and Pippenger who exploited the succinctness of the representation to show that the unknot can be recognized in **NP** [8]. To this end they had to verify in polynomial time that a special type of normal surface is an essential disk, in particular, that it is connected. Agol, Hass, and Thurston [1] strengthened this result by showing that the number of connected components of a normal surface can be computed in polynomial time. This immediately implies polynomial time algorithms for checking whether a normal surface is connected, and whether it is orientable.

Independently, we developed a set of tools for algorithms on normal curves in [14], that grew out of our work on the string graph recognition problem [15]. We showed how to compute connected components, count them, decide isotopy of curves, and compute the algebraic intersection number of two curves. The novel ingredient in our approach was that it was based on recent developments of algorithms over free monoids rather than groups.

In the current paper we continue the study of curves by showing how to efficiently perform Dehn twists, a fundamental topological operation. As a consequence we obtain an algorithm for computing the geometric intersection number of two curves. The following theorem summarizes our main results.

Theorem 1 *The tasks of performing a Dehn twist of a curve along another curve on an arbitrary surface, and computing the geometric intersection number of two curves on a surface with boundary can be solved in polynomial time in the normal coordinate representation (in a given triangulation).*

These results are very strong, since the normal coordinate representation is very succinct: the length of a curve is exponential in the size of its representation. As an application of Theorem 1 we show that a generalized notion of crossing number lies in **NP**, which unifies several non-trivial complexity results in graph drawing.

There has been previous work on calculating the effect of a Dehn twist. Penner [12] gave explicit formulas describing the action of the Lickorish generators on the Dehn-Thurston coordinates for the set of all isotopy classes of simple curves. This solves the problem for a very restricted case only, the Lickorish generators. Dehn twists along other curves can be obtained by applying Dehn twists along Lickorish generators, but exponentially many Dehn twists may be needed.

More recently Hamidi-Tehrani and Chen [7] gave an algorithm to compute the action of a set of generators on the space $\mathrm{CS}(M)$ given by measured π-train tracks, but its running time is exponential in the representation.

Hamidi-Tehrani's thesis [6] gave an algorithm for the geometric intersection number running in time polynomial in the length of the curves. Again, this translates only to an exponential time algorithm in the succinct representations.

2 Surfaces, Curves, and Words

2.1 Representations of Surfaces and Simple Curves

By a *surface* M we mean a connected, compact, orientable 2-manifold with boundary ∂M.[1] A *curve* in M is the image of $[0,1]$ under some continuous function f; it is *simple* if f is injective with the possible exception of $f(0) = f(1)$. Intuitively, the curve is simple if it has no self-intersections; a curve is *closed* if it has no endpoints, that is, $f(0) = f(1)$. A closed curve is *trivial* if it can be contracted to a point (null-homotopic) or a boundary component. We call a curve a *properly embedded simple arc* if it is a simple curve with both of its endpoints on ∂M.

A simple *multi-curve* α in M is the disjoint union of any number of non-trivial simple closed curves and properly embedded arcs.[2] Call a simple multi-curve *closed* if all its connected components are closed. By *isotopy* we mean isotopy rel boundary, i. e., a continuous deformation which leaves ∂M fixed and does not introduce self-intersections. Let $\mathrm{CS}(M)$ be the set of all isotopy classes of simple multi-curves. Let $\mathrm{CS}_0(M) \subseteq \mathrm{CS}(M)$ be the set of isotopy classes of simple multi-curves whose components are simple closed curves.

The *geometric intersection number*, $i(\alpha, \beta)$ of two (isotopy classes of) simple multi-curves $\alpha, \beta \in \mathrm{CS}(M)$ is the minimal number of intersections between any of their representatives.

A *triangulation* T of a surface M is a set of points V in M and an embedded collection of arcs E such that each component of $M - E$ is an open disc bounded by three curves from E. A triangulation is *minimal* if the vertices V are on the boundary ∂M and each boundary component contains exactly one vertex of V.

Let M be given by a triangulation. A simple multi-curve γ is *normal* w.r.t. T if all intersections of γ with T are transversal and if γ enters a triangle t via an edge e then it leaves t via an edge different from e. Any simple multi-curve can be made normal by simple redrawing moves, so for any simple multi-curve there is a normal multi-curve isotopic to it. Normal curves are very well-behaved with respect to the triangulation; in particular, given a triangle a, b, c the curve segments within the triangle fall into three types: segments crossing from ab to ac, from ab to bc and from ac to bc.

Moreover, the number of segments of each type is determined by the number of intersections of the multi-curve with ab, bc and ac: for example there are $(|\gamma \cap ab| + |\gamma \cap ac| - |\gamma \cap bc|)/2$ segments crossing from ab to ac. This allows us to describe a normal curve γ by its *normal coordinates*, which is the vector $(|\gamma \cap e|)_{e \in T}$. The *complexity* of the normal coordinates is the number of bits needed to encode the vector $(|\gamma \cap e|)_{e \in T}$, by writing each coordinate in binary. Any two simple multi-curves with the same normal coordinates are isotopic if they agree on the boundary. If the triangulation is minimal then the converse is true—any isotopic curves have the same normal coordinates. (Note that only surfaces with non-empty boundary have minimal triangulations.)

Let α be a simple closed curve in M. A *Dehn twist* $D_\alpha : M \to M$ along α is a homeomorphism of M obtained by cutting M along α, rotating one of the copies of α by 360 degrees and gluing the two copies back together. More precisely, if an annular neighborhood around α is parameterized by $\{(x, \varphi), 1 \le x \le 2, 0 \le \varphi \le 2\pi\}$ then D_α is $(x, \varphi) \mapsto (x, \varphi + 2\pi(x-1) \bmod 2\pi)$ on the annulus and the identity elsewhere. If α is a simple closed multi-curve, we define D_α to be the composition of $D_{\alpha'}$ for all connected components α' of α (note that all α' are simple closed curves).

2.2 Quadratic Word Equations and Compressed Representation of Words

Let Σ be an *alphabet*. A word in Σ^* can be represented by a *straight-line program* (SLP), which is a sequence of assignments $X_i := \textsc{expr}$, $i = 1, \ldots, n$, where EXPR is a either a symbol from Σ or $X_j X_k, 1 \le j, k < i$. The *length* n of an SLP representing a word w can be exponentially smaller than $|w|$, the length of w.

Example 2 *Over the alphabet $\{a, b\}$ the SLP $X_1 = a$, $X_2 = b$, $X_3 = X_1X_2$, $X_4 = X_3X_3$, $\cdots$, $X_n = X_{n-1}X_{n-1}$ of length n represents the word $(ab)^{2^{n-3}}$.*

Equality of two words given by SLPs of lengths m and n can be tested in deterministic time $O(m^2n^2)$ [11] and in randomized time $O(m+n)$ [5].

A *word equation with specified lengths* is an equation over a free monoid in which every variable has to be replaced with a word of a specified length.

Example 3 *The word equation $XabY = YbaX$ in variables X and Y has two shortest solutions: $X = \epsilon, Y = a$ and $X = b, Y = \epsilon$, where ϵ is the empty word. If we require $|X| = 4$ and $|Y| = 2$, the equation has no solution. However, if we require $|X| = 3$ and $|Y| = 1$, then $X = aba$ and $Y = a$ is a solution.*

[1] In other words, each point in the manifold has a neighborhood homeomorphic to a disk or a half-disk (if it is on the boundary).

[2] Formally, it is a proper 1-dimensional submanifold of M such that no component of α is null-homotopic or homotopic to the boundary.

Finding a solution of a word equation in general is **NP**-hard. However, the word equations arising in our context all have specified lengths and are *quadratic*: every variable occurs at most twice in the equation. For quadratic word equations a faster and simple algorithm is known which also guarantees that the solution can be expressed as an SLP.

Theorem 4 (Robson, Diekert [13]) *The solvability of a quadratic word equation with specified lengths can be decided in time linear in the complexity of the equation. If there exists a solution, a linear-size SLP for any subword of the solution can be found in linear time.*

2.3 Representations of Simple Curves

We have seen that simple curves can be succinctly represented using normal coordinates, assuming they have been normalized with respect to some triangulation T. Another natural way of representing such a simple curve would be by listing the order in which it crosses the edges of the triangulation: Arbitrarily fix an orientation $\vec{e}$ of each edge $e \in T$. Given an oriented simple curve γ let w be a word obtained by traversing γ and appending e to w if $\vec{e}$ is crossed from left to right and appending e^{-1} is $\vec{e}$ is crossed from right to left. Then w is called an *intersection sequence of γ with the triangulation*. (Note that for closed curves the intersection sequence is only determined up to cyclic permutation.)

An intersection sequence can be exponentially long compared to the normal coordinates of a simple curve, but, as it turns out, intersection sequences are highly compressible since they can be obtained as solutions of quadratic word equations and can therefore be represented by SLPs. Moreover, moving from an SLP representation of an intersection sequence to normal coordinates is simple: we just need to count the number of occurrences of each symbol in the compressed word, a task that can be performed in time $O(n)$ for each symbol, see [4]. In short, normal coordinate representation and SLP representation of intersection sequences are polynomially equivalent.

Theorem 5 *Let M be a surface given by a triangulation T. Let $\alpha \in \mathrm{CS}(M)$ be a simple curve in M. If α is given by an intersection sequence with T encoded by an SLP of length n, then the normal coordinates of α can be computed in time $O(n \cdot |T|)$. If α is given by normal coordinates in T with complexity n, then an SLP of size $O(n)$ for an intersection sequence of α with T can be obtained in time $O(n)$.*

We omit the proof of this result, the interesting direction (from normal coordinates to an SLP for an intersection sequence), follows techniques suggested in [14]. Note that while normal coordinates can encode multi-curves (which can have many connected components), intersection sequences are restricted to simple curves (which have a single component).

3 Dehn Twists and Applications

3.1 Computing Dehn Twists

The Dehn twist of a curve given by normal coordinates can be computed in time polynomial in the size of the representation. Our algorithm works for all surfaces (that is, with or without boundary).

Theorem 6 *Let M be a surface of genus g given by a triangulation T. Let be $\alpha \in \mathrm{CS}(M)$ and $\beta \in \mathrm{CS}_0(M)$ be simple multi-curves in M given by normal coordinates of complexity n. The normal coordinates of a representative of the Dehn twist $D_\beta(\alpha)$ can be computed*

- *in time $O(g \cdot n^3)$ by a randomized algorithm with small probability of error; and*
- *in time $O(g \cdot n^9)$ by a deterministic algorithm.*

We have to omit the proof, but we can give a rough outline: we first prove the result for a simple closed curve β. Performing a Dehn twist along a simple closed curve α can be captured by a quadratic word equation and therefore $D_\beta(\alpha)$ can be represented by a compressed SLP. There is some subtlety here, since the word obtained as a solution of the quadratic word equation may not correspond to a normalized simple curve; however, normalization can be performed on the word (it corresponds to cancellation over a free monoid with inverses); this step is computationally expensive, since it requires iterated equality testing. The result can then be generalized to simple closed multi-curves α; this might require performing an exponential number of Dehn twists, but this can be done "in parallel" using word equations.

3.2 Computing Geometric Intersection Numbers

For surfaces with a boundary we can compute geometric intersection numbers in polynomial time once we can compute Dehn twists in polynomial time. The reason is that the complexity of Dehn twists is closely related to the geometric intersection number: Considering representatives of β and γ which intersect minimally shows that for any simple curve α

$$|D_\gamma^n(\beta) \cap \alpha| \leq |\beta \cap \alpha| + n \cdot i(\beta, \gamma)|\gamma \cap \alpha|,$$

where D_γ^n is the n-fold application of D_γ. Hence, $D_\gamma^n(\beta)$ grows by a rate of at most $i(\beta, \gamma)|\gamma \cap \alpha|$ in n. For sufficiently large n, it grows exactly at that rate:

Lemma 7 *Let M be a surface and α, β, γ simple curves on M with γ closed. Let $n = 2i(\alpha, \beta)$. Then*

$$i(\gamma, \beta) = \frac{i(\alpha, D_\gamma^{n+1}(\beta)) - i(\alpha, D_\gamma^n(\beta))}{i(\alpha, \gamma)}.$$

We omit the proof which is based on results by Fathi, Laudenbach, Poénaru [3] and Luo [10]. Lemma 7 is the core observation needed to establish the main result:

Theorem 8 *The geometric intersection number of two curves given by normal coordinates on a surface with boundary can be computed in polynomial time.*

3.3 Generalized Crossing Number

We can apply our algorithm for calculating the geometric intersection number of two curves to a notoriously intractable graph drawing problem: the crossing number. The *crossing number*, $\mathrm{cr}(G)$, of a graph $G = (V, E)$ is the smallest number of intersections in a drawing of G in the plane (making certain standard assumptions about the drawing). Given a weight function $w : E^2 \to \mathbb{N}$, we can define a generalization, $\mathrm{cr}_w(G)$ of the crossing number as the minimum value of

$$\sum_{e,f \in E} i_D(e, f) \cdot w(e, f)$$

over all drawings D of G in the plane. For $w(e, f) = 1$ we obtain the usual crossing number.

Theorem 9 *Deciding whether* $\mathrm{cr}_w(G) \leq k$ *lies in* **NP** *for any polynomial-time computable weight function* w.

The generalized crossing number is a powerful modeling tool. For example, the weak realizability problem is a special case, and, therefore, lies in **NP**. This, in turn implies that the string graph problem, topological inference, and several other problems also lie in **NP**, see [15]). As another application, we consider the recently introduced simultaneous graph drawing model SCM^+ introduced by Chimani, Jünger, and Schulz [2]: given two planar graphs on the same vertex set, a *simultaneous drawing with fixed edges* is a drawing in which each graph by itself is planar and shared edges are drawn identically. Now, if we relax the condition that the two graphs be planar, we get the *simultaneous crossing number*, which is the smallest number of crossings in a drawing of the union of the two graphs that occur between edges of the same graph. If, in addition, we require that the total number of crossings be minimized, we get the SCM^+ model. It is easy to see that this problem is a special case of our generalized crossing number problem.

References

[1] I. Agol, J. Hass, and W. Thurston. 3-manifold knot genus is NP-complete. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing (STOC-2002)*, 2002.

[2] M. Chimani, M. Jünger, and M. Schulz. Crossing minimization meets simultaneous drawing. Technical report, Zentrum für Angewandte Informatik Köln, Lehrstuhl Jünger, May 2007.

[3] A. Fathi, F. Laudenbach, and V. Poénaru. Travaux de Thurston sur les surfaces. *Astérisque*, 66–67, 1979.

[4] L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. *in Proceedings of SWAT'96, LNCS 1097*, pages 392–403, 1996.

[5] L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter. Randomized efficient algorithms for compressed strings: The finger-print approach. In *LNCS 1075*, pages 39–49, 1996.

[6] H. Hamidi-Tehrani. *Algorithms in the Mapping Class Groups*. PhD thesis, Columbia, 1997.

[7] H. Hamidi-Tehrani and Z.-H. Chen. Surface diffeomorphisms via train-tracks. *Topology and its Applications*, 73:141–167, 1996.

[8] J. Hass, J. Lagarias, and N. Pippenger. The computational complexity of knot and link problems. *Journal of ACM*, 46(2):185–211, 1999.

[9] H. Kneser. Geschlossene Flächen in dreidimensionalen Mannigfaltigkeiten. *Jahresbericht der Deutschen Mathematikver-Vereinigung*, pages 248–260, 1930.

[10] F. Luo. Some applications of a multiplicative structure on simple loops in surfaces. In *Knots, braids, and mapping class groups*, volume 24, pages 123–129. Amer. Math. Soc., Providence, RI, 2001.

[11] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *LNCS 1264*.

[12] R. C. Penner. The action of the mapping class group on curves in surfaces. *L'Enseignement Mathématique*, 30:39–55, 1984.

[13] J. M. Robson and V. Diekert. On quadratic word equations. *LNCS*, 1563:217–226, 1999.

[14] M. Schaefer, E. Sedgwick, and D. Stefankovic. Algorithms for normal curves and surfaces. In *COCOON '02*, pages 370–380, London, UK, 2002. Springer-Verlag.

[15] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. *J. Comput. System Sci.*, 67(2):365–380, 2003. Special issue on STOC2002 (Montreal, QC).

An Efficient Query Structure for Mesh Refinement

Benoît Hudson* Duru Türkoğlu†

Abstract

We are interested in the following mesh refinement problem: given an input set of points P in $\mathbb{R}^d$, we would like to produce a good-quality triangulation by adding new points in P. Algorithms for mesh refinement are typically incremental: they compute the Delaunay triangulation of the input, and insert points one by one. However, retriangulating after each insertion can take linear time. In this work we develop a query structure that maintains the mesh without paying the full cost of retriangulating. Assuming that the meshing algorithm processes bad-quality elements in increasing order of their size, our structure allows inserting new points and computing a restriction of the Voronoi cell of a point, both in constant time.

1 Introduction

A central task in scientific computing is to discretize a domain of interest into a simplicial mesh. Concretely, we investigate the following setting: we are given a point cloud P in $[0,1]^d$, typically $d = 2$ or 3. A meshing algorithm must produce a simplicial decomposition of $[0,1]^d$ (a triangulation in $d = 2$), in which all of the input points appear as vertices of the decomposition. Each simplex should have good aspect ratio (*quality*), since that is a necessary or at least sufficient condition for many scientific computing applications. To ensure this, the algorithm must *refine* the point set, creating new (*Steiner*) vertices. But it should not add too many vertices: if the smallest possible mesh of good aspect ratio has m vertices, the algorithm should output $O(m)$ vertices.

There are two categories of mesh refinement algorithms with provable guarantees: those based on quadtrees [BEG94, MV00], and those based on Delaunay triangulations [Rup95, She98]. Quadtrees can be constructed in $O(n \log n + m)$ time [BET99]. Traditional Delaunay methods produce substantially smaller meshes in practice, but are quadratically slow in the worst case. The runtime behaviour of these algorithms is dominated by the cost of maintaining the mesh as the output is incrementally computed. Two Delaunay refinement algorithms sidestep this cost and match the quadtree runtime: Har-Peled and Üngör [HPÜ05] hybridize Delaunay and quadtree methods, while Hudson, Miller, and Phillips [HMP06] carefully maintain a triangulation that is always sparse. Here, we extend the hybrid approach to three or more dimensions.

*Toyota Technological Institute at Chicago
†University of Chicago

INITIALIZE(P)	$O(n \log n + m)$
APPROXIMATENN(v)	$O(1)$
CLIPPEDVORONOI(v, β)	$O(1)$
ADDVERTEX(p, v)	$O(1)$

Fig. 1: The interface of our data structure, and runtimes assuming a bottom-up mesh refinement algorithm.

Mesh quality: In a mesh where all simplices have good aspect ratio, the vertices of the mesh are well-spaced in the following sense [Tal97]. Denote by $\mathrm{NN}(v)$ the distance from v to its nearest neighbour, and by $R(v)$ the distance from v to the farthest node of its Voronoi cell. Then we say that v is ρ-well-spaced if $R(v) \leq \rho NN(v)$.

Mesh spacing: The *local feature size* $\mathrm{lfs}(x)$ is the distance from x to its second-nearest input point [Rup95]. In a *size-conforming* mesh, at every output vertex v, $\mathrm{NN}(v) \in \Theta(\mathrm{lfs}(v))$. A size-conforming mesh has $O(m)$ vertices.

Bottom-up meshing: We say that a mesh refinement algorithm is bottom-up if it incrementally ensures that small Voronoi cells are well-shaped before processing large Voronoi cells. That is, there are constants ρ and γ such that before the algorithm inserts a new Steiner point p with a nearest other mesh vertex at distance $\mathrm{NN}(p)$, every mesh vertex v with $\mathrm{NN}(v) < \gamma\,\mathrm{NN}(p)$ is ρ-well-spaced.

Our contribution: Expanding upon the idea of Har-Peled and Üngör, we provide a data structure for a meshing algorithm to use as a black box to maintain information about the partial mesh as it is constructed. After initializing our structure, the algorithm has access to two queries and an update routine (see Figure 1). CLIPPEDVORONOI(v, β) computes the nearby portion of the Voronoi cell of a vertex v. We show in Section 2 that this clipped version of the Voronoi cell is sufficient to determine where to insert a Steiner vertex near v. Upon deciding to insert a new Steiner vertex at p, the algorithm can call ADDVERTEX(p, v) to effect the insertion. We prove in Theorem 7 that both CLIPPEDVORONOI and ADDVERTEX run in constant time, assuming the algorithm is a bottom-up meshing algorithm that produces a size-conforming mesh. To help the algorithm produce the bottom-up ordering, we provide APPROXIMATENN(v), which returns an approximation of the true distance from v to its nearest neighbour. Using our structure, a bottom-up mesh refinement algorithm can mesh a point cloud in $\mathbb{R}^d$ in $O(n \log n + m)$ time.

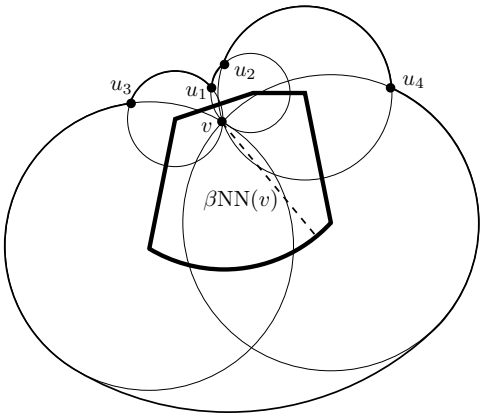


Fig. 2: The β-clipped Voronoi cell of v and its certificate region, bounded by neighbours u_1 through u_4.

2 Choosing Steiner Points

Regardless of runtime considerations, the fundamental question in mesh refinement is about where to put the Steiner points. Traditional solutions compute Steiner points based on the Delaunay triangulation, which is too expensive to maintain, so we need a more local way of choosing a Steiner point. The following region includes the points defined by two prior proposals for Steiner point choices — the *off-center* [Üng04] in 2D, and the CoreDisk [JÜ07] in 3D:

Definition 1 *The β-**clipped Voronoi cell** of a vertex v, denoted $V^\beta(v)$, is the intersection of the Voronoi cell of v, and the ball centered at v with radius $\beta\,\mathrm{NN}(v)$.*

Any point x in $V^\beta(v)$ is in the Voronoi cell of v; therefore, the open ball centered at x with radius $|vx|$ is empty of any mesh vertices. We call these **certificate balls**, and define the **certificate region** of $V^\beta(v)$ as the union of these balls (Figure 2). An algorithm that correctly computes $V^\beta(v)$ must have verified that the entire certificate region is empty; otherwise, the algorithm could falsely report x as being in the clipped Voronoi when it lies outside. Furthermore, an algorithm must verify that for all x on the boundary, either x is equidistant to v and another vertex, or x is at the clipping distance $\beta\,\mathrm{NN}(v)$ from v; otherwise, the algorithm could falsely report the clipped Voronoi cell is smaller than it is.

3 Data structure

Our data structure is based on the leaves of a quadtree. Each leaf square (or hypercube) stores the set of vertices contained in the square, and pointers to the neighbouring squares. Also, each vertex v stores a pointer to the square in which it lies, which we denote square(v). Our proofs of fast runtime depend on the quadtree having certain properties: **(A)** Each leaf square should only have a bounded number of neighbours. **(B)** Each leaf square should be sized in proportion to the local feature size of the points in the cell. That is, if a quadtree square has sides of length l, then for all x in the square, $\mathrm{lfs}(x) \in \Theta(l)$. The balanced quadtree of Bern, Eppstein, and Gilbert [BEG94] satisfies our requirements, though in practice it will be preferable to split the squares more coarsely.

It takes INITIALIZE $O(n \log n + m)$ time to compute the quadtree leaves [BET99]. The mesh spacing requirement on the meshing algorithm, and requirement (B) on the quadtree, together guarantee that the square size is within a constant factor of $\mathrm{NN}(v)$. This allows APPROXIMATENN(v) to return the size of square(v), in constant time. To implement ADDVERTEX(p, v), we first look up square(v), then walk from square to square along the segment pv. Upon reaching the square that contains p, we add p to its list of vertices. The description of CLIPPEDVORONOI merits its own section.

4 Clipped Voronoi Computation

We represent the β-clipped Voronoi cell of a vertex v using a set U of mesh vertices (e.g. $U = \{u_1, u_2, u_3, u_4\}$ in Figure 2). To find them, we perform a scan starting at v and growing a ball outward up to a maximum radius: at time t the ball has radius $t < t_{\max} = \beta\,\mathrm{NN}(v)$. When our scan reaches a vertex u at time t, we add u to U_t if it is a Voronoi neighbour, and $|uv| \leq 2t_{\max}$ — i.e. u is on the boundary of the certificate region. At time $t_{\max}$, we will know that we have covered the entire certificate region, and that $U_{t_{\max}}$ is an accurate representation of $V^\beta(v)$.

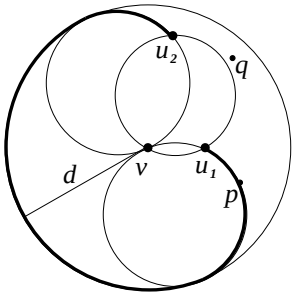


Fig. 3: The thick curve is the set of points p with distance $d_v^U(p) = d$. No empty ball reaches q, so $d_v^U(q) = \infty$.

For efficiency, we need to ensure that the scan does not exceed the certificate region. Therefore, we use a distance function different from the Euclidean one. For any point p, we define the distance $d_v^M(p)$ as the diameter of the smallest certificate ball for p. If no such ball exists then we say $d_v^M(p) = \infty$.

Computing $d_v^M(p)$ exactly is inefficient. However, we can relax the requirement that the balls be empty of any mesh vertices, and instead at time t compute $d_v^{U_t}(p)$ — the diameter of the smallest ball with v and p on its surface that includes no vertex of U_t in its interior. This distance function is non-decreasing in time: adding vertices can only make it harder for a ball to be empty. Therefore, $d_v^{U_t}(p) \leq d_v^M(p)$ for all t. Interestingly, when the scan reaches p, the efficiently-computed lower bound is tight:

Lemma 2 *If point p has $d_v^{U_t}(p) \leq t$, then $d_v^{U_t}(p) = d_v^M(p)$.*

Proof. The assumption of the lemma implies that there exists a ball B of diameter $d_v^{U_t}(p)$ that contains no vertex of U_t. Consider a point q strictly inside B. It has a strictly smaller

CLIPPEDVORONOI(v, β)
1: Let $Q \leftarrow \{\text{square}(v)\}, t_{\max} \leftarrow \infty$, and $U \leftarrow \emptyset$
2: **while** Q contains an element q with $d_v^U(q) < t_{\max}$ **do**
3: $q \leftarrow$ Minimum of Q with respect to d_v^U
4: **if** q is a vertex **then**
5: Add q to U
6: **if** q is the first vertex **then** $t_{\max} \leftarrow \beta|vq|$
7: **if** q is a quadtree square **then**
8: Add each vertex in the square to Q
9: Add every unvisited neighbouring square to Q
10: **return** U

Fig. 4: The CLIPPEDVORONOI algorithm, which performs a scan through the quadtree and mesh vertices.

ball that is empty: $d_v^{U_t}(q) < t$. Because the computed distance function is nondecreasing in t, we must already have visited q, and discovered whether or not q is a vertex of M. In fact, we know it is not because if it were it would be both a vertex in U_t and inside B, contradicting that B is empty. This applies for all $q \in B$: all of B is in fact empty of vertices of M. Monotonicity further implies that B is the smallest empty ball, which proves $d_v^{U_t}(p) = d_v^M(p)$. □

As a corollary, this shows that the $U_{t_{\max}}$ computed by the scan faithfully represents the β-clipped Voronoi cell: The scan visits the entire certificate region, because all points in the certificate region have distance less than $t_{\max}$. It also visits no more than the certificate region, because any point we visit has an empty ball with its center in $V^\beta(v)$.

Implementation (see Figure 4): We discretize the scan using quadtree squares. Starting at square(v), we explore outward from v using a queue Q of events — vertices and squares. Upon processing a vertex, we update the current Voronoi cell U; and upon processing a square, we enqueue the vertices it contains, and the squares it neighbours. We compute $d_v^U(p)$ using a convex program: we find a point c that is the center of an empty ball of minimum radius.

$$\begin{array}{ll} \text{minimize} & |cv| \\ \text{subject to} & |cv| = |xp| \\ & |cv| \geq |cu_i| \text{ for all } u_i \in U \end{array}$$

For a quadtree square, the distance is the minimum distance to any p in the square; this corresponds to letting p be free variables in the above program, and, in dimension d, adding $2d$ constraints on the coordinates of p.

5 Runtime Proof

We prove that CLIPPEDVORONOI and ADDVERTEX are fast in two parts. First, we present a basic geometric fact about empty balls in partially-constructed meshes. This then implies that the certificate region intersects a bounded number of quadtree squares. From there it is an easy corollary to show that CLIPPEDVORONOI and ADDVERTEX run in constant time.

5.1 Points in an empty ball have large lfs

We first show that empty balls do not contain points with small local feature size. We adapt a prior argument that assumed the entire mesh was well-spaced [HMP06]; in the present version, we only require that vertices with small nearest neighbour distance are well-spaced, so that our result will apply in the intermediate stages of a bottom-up algorithm. For clarity, we define a *local mesh size* function induced by the set of vertices the algorithm has output so far (including all of the input points). We say that $\text{lms}_M(p)$ is the distance from p to the second-nearest vertex of the partial mesh M. This differs from the local feature size $\text{lfs}(p)$ in that the local feature size only considers input points, not output vertices. In particular, $\text{lfs}(p) \geq \text{lms}_M(p)$.

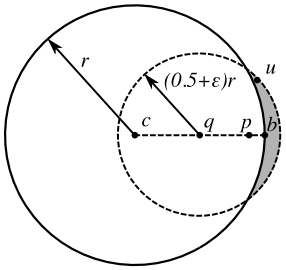


Fig. 5: Setup for the proofs of Section 5.1.

Theorem 3 *Given a mesh M and an empty ball of radius r, assume every vertex u in M with $\text{NN}(u) \leq \gamma r$ is ρ-well-spaced. Then there is a constant ϵ that depends only on γ and ρ such that at any point p in the empty ball, $\text{lms}_M(p) \geq \epsilon r$.*

Proof. If p is at distance at least ϵr from all vertices, we are done. Otherwise, we have the situation in Figure 5. Let c be the center of the empty ball in question. We identify two points on the ray from c to p: q is at distance $r/2$ from the center, and b is on the boundary. Finally, u is the vertex nearest q: q is in the Voronoi cell of u. Lemma 4 will show that $\text{NN}(u) \geq \min(\gamma, \frac{1}{2\rho})r$. This then implies Lemma 5: for appropriate ϵ, u is also the vertex nearest p. The distance from any point in the Voronoi cell of u to a second vertex is minimized at the point x equidistant between u and its nearest neighbour; there, $\text{lms}_M(x) = \text{NN}(u)/2$. Therefore, $\text{lms}_M(p) \geq \frac{\eta}{2}r$. Since $\eta/2 > \epsilon$, the result holds. □

Lemma 4 *The vertex nearest u is at distance $\text{NN}(u) \geq \eta r$, where $\eta = \min(\gamma, \frac{1}{2\rho})$.*

Proof. If $\text{NN}(u) \geq \gamma r$, we are done. Otherwise, we know that u is well-spaced: $\text{NN}(u) \geq R(u)/\rho$. At the same time, we know that $R(u) \geq |qu|$ since q is in the Voronoi cell; and that $|qu| \geq r/2$ because u is outside the empty ball. □

Lemma 5 *If there is some vertex within ϵr of p, then u, the nearest vertex to q, is also the nearest vertex to p.*

Proof. We know there is a vertex u' within distance ϵr of p, and we know that $|pq| \leq r/2$. By the triangle inequality, $|qu'| \leq (1/2 + \epsilon)r$. Then u lies somewhere within the ball centered at q, of radius $(1/2 + \epsilon)r$, since u is nearer to q than is u'. We also know that u must be outside the ball

centered at c, of radius r. These two constraints define a crescent, shown shaded in Figure 5. Consider the plane that goes through c, b, and u. Said plane also contains q since c, b, and q are collinear. We can conformally transform the plane so that c is the origin, $q = \langle 0.5, 0\rangle$, $b = \langle 1, 0\rangle$, and $u = \langle x, y\rangle$. Under this transformation, $r = 1$. The farthest u can be while still lying in the crescent is the point of the crescent where $|cu| = 1$ and $|qu| = 1/2 + \epsilon$. We rewrite the first equality as $x^2 + y^2 = 1$, while the second gives us that $x = 1-\epsilon-\epsilon^2$. Then we can conclude that $|bu|^2 \leq 2-2x = 2(\epsilon+\epsilon^2)$. Recall from the prior lemma that the nearest neighbour of u is at distance η; therefore, every point at distance $\eta/2$ from u is within the Voronoi cell of u. Solving for $|bu| = \eta/2$ we see that by setting $\epsilon = \frac{1}{4}\sqrt{4+2\eta^2} - 1/2$, we ensure that b lies in the Voronoi cell of u. Since both b and q lie in the same convex set, p also lies in it. □

5.2 Queries and updates are fast

Lemma 6 *Assume we are given a size-conforming mesh M, a value r with the guarantee that every vertex u in M with $\mathrm{NN}(u) \leq \gamma r$ is ρ-well-spaced, and a quadtree that satisfies condition (B). Then the certificate region of any vertex v with $\mathrm{NN}(v) \leq r$ intersects $O(1)$ leaf squares of the quadtree.*

Proof. At any x in the certificate region, there is an empty ball containing x that has diameter at least $\mathrm{NN}(v)$. Therefore, Theorem 3 shows that $\mathrm{lfs}(x) \in \Omega(\mathrm{NN}(v))$. Any quadtree square that intersects the query region necessarily includes at least one such x, which, conjoined with condition (B) on the quadtree, implies that the squares each have side length in $\Omega(\mathrm{NN}(v))$. Finally, a volume packing argument applies: within a ball of volume $O(\beta\,\mathrm{NN}(v))^d$, every visited quadtree square consumes $\Omega(\mathrm{NN}(v))^d$ volume. □

Theorem 7 *When a bottom-up mesh refinement algorithm calls* AddVertex *or* ClippedVoronoi*, our data structure responds in constant time.*

Proof. In AddVertex(p, v), if p lies within the β-clipped Voronoi cell of v, then every square visited by the call intersects the certificate region. Upon finding the destination square, AddVertex simply appends to a list. Thus AddVertex does constant work. When the ClippedVoronoi algorithm visits a quadtree square, that square either intersects the certificate region, or is a neighbour of a square that intersects the certificate region. Thanks to the guarantee that squares have a bounded number of neighbours, the latter outnumber the former by at most a constant factor. Thus ClippedVoronoi visits only $O(1)$ squares. The function also does work iterating over the vertices each square contains. By the mesh spacing requirement, a mesh vertex v has a nearest neighbour no closer than $\Omega(\mathrm{lfs}(x))$; meanwhile, the quadtree square that contains v has sides of length $O(\mathrm{lfs}(v))$. Therefore, each square only hosts $O(1)$ mesh vertices, and the total work is $O(1)$. □

6 Higher-dimensional input features

In the present work, we showed how to support a class of mesh refinement algorithms with a search structure that could find appropriate new Steiner points in constant time, assuming the input is a point cloud. Typically, engineers and graphic artists will want the mesh to respect more than just a point cloud: it should respect some meaningful segments and polygons, and perhaps also curves and curved surfaces. Of course, the client application could simply specify a denser packing of points on the surfaces, but ideally the meshing algorithm could handle the features directly. Nothing fundamental blocks the extension of our structure to handle the case of higher-dimensional features (even curved, and with small input angles), although we would need need to enrich the interface our data structure presents.

References

[BEG94] Marshall Bern, David Eppstein, and John R. Gilbert. Provably good mesh generation. *J. Computer and System Sciences*, 48(3):384–409, 1994.

[BET99] Marshall Bern, David Eppstein, and Shang-Hua Teng. Parallel construction of quadtrees and quality triangulations. *IJCGA*, 9(6):517–532, 1999.

[HMP06] Benoît Hudson, Gary L. Miller, and Todd Phillips. Sparse Voronoi Refinement. In *IMR*, pages 339–356, 2006.

[HPÜ05] Sariel Har-Peled and Alper Üngör. A time-optimal Delaunay refinement algorithm in two dimensions. In *SoCG*, pages 228–236, 2005.

[JÜ07] Ravi Jampani and Alper Üngör. Construction of sparse well-spaced point sets for quality tetrahedralizations. In *IMR*, pages 63–80, 2007.

[MV00] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Computing*, 29(4):1334–1370, 2000.

[Rup95] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[She98] Jonathan Richard Shewchuk. Tetrahedral Mesh Generation by Delaunay Refinement. In *SoCG*, pages 86–95, 1998.

[Tal97] Dafna Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 1997.

[Üng04] Alper Üngör. Off-centers: A new type of Steiner point for computing size-optimal quality-guaranteed Delaunay triangulations. In *LATIN*, pages 152–161, 2004.

Application of computational geometry to network p-center location problems

Binay Bhattacharya *† Qiaosheng Shi †

Abstract

In this note we showed that a $p(\geq 2)$-center location problem in general networks can be transformed to the well known Klee's measure problem [3]. This resulted in an improved algorithm for the continuous case with running time $O(m^p n^{p/2} 2^{\log^* n} \log n)$. The previous best result for the problem is $O(m^p n^p \alpha(n) \log n)$ where $\alpha(n)$ is the inverse Ackermann function [9]. When the underlying network is a partial k-tree (k fixed), by exploiting the geometry inherent in the problem we showed that the discrete p-center problem can be solved in $O(pn^p \log^k n)$ time.

1 Introduction

The *network p-center problem* is defined on a weighted undirected network $G = (V(G), E(G))$, where each vertex $v \in V(G)$ has a non-negative weight $w(v)$ and each edge $e \in E(G)$ has a positive length $l(e)$. Let $A(G)$ denote the continuum set of points on the edges of G. For $x, y \in A(G)$, $\pi(x, y)$ denote the *shortest path* in G from x to y, and $d(x, y)$ denote the length of $\pi(x, y)$. Let $\mathcal{D}(G)$ be the set of demand points (or the *demand set*) and $\mathcal{X}(G)$ be the set of candidate facility locations in G. In a p-center problem, a set X of p centers is to be located in $\mathcal{X}(G)$ so that the maximum (weighted) distance from a demand point in $\mathcal{D}(G)$ to its nearest center in X is minimized, i.e.,

$$\min_{X \subseteq \mathcal{X}(G), |X|=p} \{F(X, \mathcal{D}(G)) = \max_{y \in \mathcal{D}(G)} \{w(y) \cdot d(y, X)\}\}.$$

Here $d(y, X) = \min_{x \in X} d(y, x)$ and $F(X, \mathcal{D}(G))$ denotes the cost of serving the demand set $\mathcal{D}(G)$ using facilities in X.

A value $r > 0$ is *feasible* if there exists a set of at most p points (centers) in $\mathcal{X}(G)$ such that the distance between any demand point in $\mathcal{D}(G)$ and its nearest center is not greater than r. An approach to test whether a given positive value is feasible is called a *feasibility test*.

Our study in this paper is restricted to the case where $\mathcal{D}(G) = V(G)$. When p centers are restricted to be vertices of G, we call it a *discrete problem*. Accordingly, the problem is called a *continuous problem* when $\mathcal{X}(G) = A(G)$. The continuous/discrete problems have been shown to be NP-hard in general networks [5]. But, center problems are no longer NP-hard when either p is constant [5, 7, 9], or the underlying network is restricted to be a specialized network, such as a tree [5], a cactus [1], or a partial k-tree (fixed k) [6]. In the paper we study p-center problems in general networks and partial k-trees (k fixed) for a constant p and provide improved algorithms by exploiting the geometric properties of the problems.

*Research was partially supported by MITACS and NSERC

†School of Computing Science, Simon Fraser University, Burnaby B.C., Canada. V5A 1S6, {binay, qshi1}@cs.sfu.ca

2 Continuous p-center problem in general networks

The best known algorithms [5, 7, 9] to solve the continuous p-center problem in a general network are based on the following two simple observations.

Observation 1 *[5] There exists a p-center solution such that all the centers are intersection points of service functions of pairs of vertices on edges and therefore, the optimal objective value is of the form $(w(u) \cdot w(v) \cdot L(u, v))/(w(u) + w(v))$, where $L(u, v)$ is the length of the shortest path connecting u and v through edge e for some pair of vertices $u, v \in V(G)$.*

Therefore, there are at most $O(n^2)$ candidate points on each edge e where centers may be located in an optimal solution. Also, each candidate point determines a candidate optimal cost. Let $\mathcal{R}$ denote the set of $O(mn^2)$ candidate values. Based on Observation 1, Kariv and Hakimi [5] proposed an $O(m^p n^{2p-1} \log n/(p-1)!)$-time algorithm for finding an optimal p-center.

The second observation is for feasibility tests of the continuous p-center problem.

Observation 2 *[7] If r is feasible, then there is a p-center solution in which each center is located at a (weighted) distance of exactly r from some vertex and all vertices are covered with service cost $\leq r$.*

The advantage of Observation 2 is that only $O(mn)$ candidate points are needed to be considered for a feasibility test. Based on this property, Moreno [7] proposed an $O(m^p n^{p+1})$-time algorithm to test the feasibility of a given value r. Since the optimal service cost, denoted by r_p, is an element of a set $\mathcal{R}$ (Observation 1), r_p can be found by performing $O(\log n)$ feasibility tests.

Tamir [9] improved Moreno's result [7] by efficiently solving a feasibility test for the 2-center problem in $O(m^2n^2\alpha(n))$ time. Here $\alpha(n)$ is the inverse Ackermann function. Therefore, the $O(m^pn^p\alpha(n)\log n)$ bound is achieved for the continuous p-center problems.

Next we present an improved algorithm for the continuous p-center problem in general networks.

2.1 Main idea and overall approach

To achieve a better upper bound, we continue to decrease the size of the set that contains an optimal p-center. Observation 3 shows that instead of $O(mn)$ candidate points, only m candidate continuous regions (i.e., edges) and n candidate points (i.e., vertices) are considered for a feasibility test.

Observation 3 *If r is feasible, then there is a p-center solution in which every edge (not including its two endpoints) contains at most one center and all vertices are covered with service cost $\leq r$.*

A *local feasibility test* of r is to determine if there exists a set of p centers on a given set $E_{p'}$ of p' ($0 \leq p' \leq p$) edges $\{e_1, \ldots, e_{p'}\}$ (note that each edge contains one center and does not include its two endpoints) and a given set of $p-p'$ vertices such that all vertices can be served within r. It is easy to see that the feasibility test of r can be completed by solving $O((m+n)^p) = O(m^p)$ local feasibility tests of r on all possible subsets of p' edges and $p-p'$ vertices, $0 \leq p' \leq p$.

Our algorithm is described as follows.

Step 1: Compute $\mathcal{R}$ that contains the optimal cost r_p.

Step 2: Perform a binary search over $\mathcal{R}$. At each iteration, test the feasibility of a non-negative value r as follows. For each set $E_{p'} \subseteq E(G)$ of p' edges and each set of $p-p'$ vertices, $0 \leq p' \leq p$,

- **Step 2.1:** remove all vertices that can be covered by $p-p'$ vertices with service cost $\leq r$; and
- **Step 2.2:** for the remaining vertices, execute the local feasibility test of r on the set $E_{p'}$ as described in the remaining part of this section.

It is sufficient to show our approach for a local feasibility test of r on a set E_p of p edges. The main idea is to transform the local feasibility test of r on E_p to a general geometrical problem called p-dimensional *Klee's measure problem* (for short, KMP) [3].

Definition 1 (Klee's Measure Problem) *Given a set of intervals (of the real line), find the length of their union.*

The natural extension of KMP to d-dimensional space is to ask for the d-dimensional measure of a set of d-boxes. A *d-box* is the cartesian product of d intervals in d-dimensional space. It is known that, given a set of n d-boxes, a $d(\geq 2)$-dimensional KMP can be solved in time $O(n^{d/2}\log n)$ using $O(n)$ storage [3], which can be reduced to $O(n^{d/2}2^{\log^* n})$ [3] ($\log^*$ denotes the iterated logarithm). Thus, a feasibility test can be solved in $O(m^pn^{p/2}2^{\log^* n})$ time if we are able to transform a local feasibility test into a KMP. The following theorem is then implied.

Theorem 2 *The continuous p-center problems, for $p \geq 2$, can be solved in $O(m^pn^{p/2}2^{\log^* n}\log n)$ time.*

In the remaining part of this section, we show the process of transforming a local feasibility test to a p-dimensional KMP.

2.2 Transformation of a local feasibility test to a p-dimensional KMP

Let us consider the case where $p = 2$. The transformation for the case where $p > 2$ can be developed in a similar way. Let $e_1 : \overline{u_1v_1}$ and $e_2 : \overline{u_2v_2}$ be the two edges to test the local feasibility of r. A local 2-center solution is composed of two points (not vertices) in which one point lies on e_1 and the other one lies on e_2.

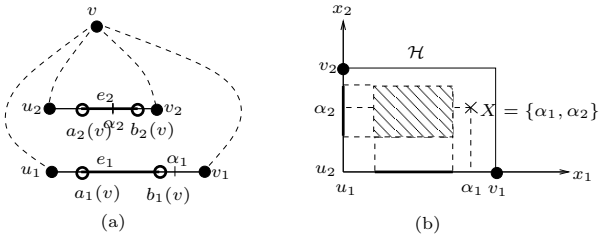


Figure 1: Mapping a 2-center local feasibility test to a 2-dimensional KMP.

We consider a 2-dimensional space in which x_i-axis represents edge $e_i, i = 1, 2$. Let u_1 and u_2 be the origin, as shown in Figure 1(b). In this coordinate system, the x_i-coordinate of a point represents a location on edge e_i with respect to $u_i, i = 1, 2$. Therefore, a point in this 2-dimensional space can be considered as a possible 2-center solution on edges e_1, e_2. We denote a point y by $(x_1(y), x_2(y))$. Clearly, only points within the bounded rectangular area $\mathcal{H} : \{y | 0 < x_1(y) < l(e_1), 0 < x_2(y) < l(e_2)\}$ are candidate 2-center solutions on e_1, e_2.

For a vertex v, there is at most one continuous region on each edge $e_i, i = 1, 2$, denoted by $R_i(v)$, which contains all points on e_i with (weighted) distance to v larger than r. It is possible that $R_i(v)$ is empty for some i ($\in \{1, 2\}$), in which case v can be served by any 2-center solution on e_1, e_2 with service cost $\leq r$. In Figure 1(a), the bold (partial) edge of e_1 (resp. e_2) is $R_1(v)$ (resp. $R_2(v)$). Let $a_i(v)$ (resp. $b_i(v)$) be the

left (resp. right) endpoint of $R_i(v), i = 1, 2$. Note that $R_i(v), i = 1, 2$ must be an open region.

A rectangular area in the 2-dimensional space (the shadow part in Figure 1(b)) is obtained for every demand vertex v, denoted by $\mathcal{H}(v)$, which is constructed from the two continuous regions $R_1(v), R_2(v)$. That is, $\mathcal{H}(v) = \{y | x_1(y) \in R_1(v), x_2(y) \in R_2(v)\}$. It is easy to see that any 2-center solution (point) in $\mathcal{H}(v)$ cannot cover v with a service cost $\leq r$ and any 2-center solution in $\mathcal{H} \setminus \mathcal{H}(v)$ can cover v with a service cost $\leq r$. In Figure 1, the 2-center solution $X = \{\alpha_1, \alpha_2\}$ can cover v with a service cost $\leq r$, but any solution in the shadow area cannot. We call $\mathcal{H}(v)$ the *forbidden area* of v. Note that, the boundary of $\mathcal{H}(v)$ is not included in $\mathcal{H}(v)$.

We compute such forbidden areas for all remaining vertices. Thus, the local feasibility test on edges e_1, e_2 is transformed into the following question: *does the union* $\bigcup_{v \in V(G)} \mathcal{H}(v)$ *of forbidden areas cover* $\mathcal{H}$*?* If the answer is 'yes' then r is infeasible on edges e_1, e_2, otherwise r is feasible. This question can be answered by solving a 2-dimensional KMP on a new set of rectangles, which are constructed from these forbidden areas. We have to be careful since the boundary of a forbidden area is not forbidden. This is handled by appropriately shrinking/expanding the boundary appropriately. The details are omitted in this note. Thus a local feasibility test on edges e_1, e_2 can be solved in $O(n \log n)$ time. Hence we now have the following theorem.

Theorem 3 *The continuous 2-center problems in a general network can be solved in* $O(m^2 n \log^2 n)$ *time.*

The extension of the above approach to the case when $p > 2$ is straightforward. Now a local p-center solution is represented as a point in a p-dimensional box (p-box) $\mathcal{H}'$ and for each demand vertex v, we obtain a p-box in $\mathcal{H}'$ containing all p-center solutions that serve v with a service cost $> r$. Thus, the local feasibility test on edges $e_1, \ldots, e_p$, is transformed into the following p-dimensional Klee's measure problem: *does the union of* $O(n)$ *axis-parallel* p*-boxes cover* $\mathcal{H}'$*?* Therefore, we have the following lemma.

Lemma 4 *A local feasibility test of the weighted continuous p-center problem on p edges* $e_1, \cdots, e_p$ *can be solved in* $O(n^{p/2} 2^{\log^* n})$ *time, for* $p > 1$.

This establishes Theorem 2.

3 Discrete p-center problems in a partial k-tree

A *partial k-tree* is a graph whose treewidth is k. It is known that a tree decomposition (of treewidth k) of a partial k-tree G (fixed k), denoted by $\mathcal{TD}(G)$, can be found in linear time [4].

Given a tree decomposition $\mathcal{TD}(G)$ of treewidth k, an $O(p^2 n^{k+2})$ algorithm [6] was proposed to solve the discrete p-center problems, which is based on the dynamic programming technique.

In fact, the approach of Granot and Skorin-Kapov [6] can be adopted to solve the continuous p-center problem by combining the result from Observation 2. Due to page restrictions the discussions of the continuous p-center problem in a partial k-tree are omitted here.

Theorem 5 *Given a tree decomposition (of treewidth k) of a partial k-tree, the continuous p-center problem can be solved in* $O(p^2 n^{2k+3} \log n)$ *time.*

In this section, we present an $O(pn^p \log^k n)$-time algorithm for the discrete p-center problem in a partial k-tree when p is small. Note that the discrete p-center problem, $p \geq 2$, in a general network is trivially solvable in $O(pn^{p+1})$ time by testing all possible solutions.

3.1 An $O(pn^p \log^k n)$-time algorithm

A distance query of a pair of points x, y in a network is to obtain the distance between x, y. Considering the tight relationship between the service cost and distance queries, an efficient approach is to preprocess the network so that distance queries can be efficiently answered. This approach is particularly promising when the network is sparse [4]. Chaudhuri and Zaroliagis [4] gave algorithms for distance queries that depend on the treewidth of the input network. Their algorithms can answer each distance query in $O(1)$ time for constant treewidth networks after $O(n \log n)$ preprocessing. Based on this result, we introduced a two-level tree decomposition structure [1] on a partial k-tree network, which can be built on any partial k-tree G in $O(n \log^2 n)$ time requiring $O(n \log^2 n)$ storage space for $k = 2$ and in $O(nk \log^{k-1} n)$ time requiring $O(nk \log^{k-1} n)$ storage space for $k > 2$. Given such a two-level tree decomposition structure, the service cost of any set X of p centers to the demand set $V(G)$, i.e., $F(X, V(G))$, can be answered in time $O(p \log^2 n)$ for $k = 2$ and in time $O(pk \log^{k-1} n)$ for $k > 2$. The main idea behind this two-level tree decomposition data structure is briefly described below for the case when G is a partial 2-tree. Recently similar idea has been used in [2] to compute some graph properties.

Given a subgraph G' represented by a subtree of $\mathcal{TD}(G)$ and a point x outside G', there is a 2-separator in G', say $\{u_1, u_2\}$, between G' and x. The service cost of x to cover $v \in V(G')$ is $w(v) \cdot \min\{d(v, u_1) + d(u_1, x), d(v, u_2) + d(u_2, x)\}$. Let $a = d(x, u_1) - d(x, u_2)$ and $a' = d(v, u_1) - d(v, u_2)$. Clearly the shortest path $\pi(v, x)$ goes through u_1 if $a + a' < 0$, otherwise $\pi(v, x)$ goes through u_2.

Based on the above observation, we create two lists of the vertices in G', $\mathcal{J}_1$ and $\mathcal{J}_2$. The vertices of $\mathcal{J}_1$

are sorted in the increasing order of $\chi_1(\cdot)$ where $\chi_1(v)$ is the distance difference from a vertex $v(\in V(G'))$ to the 2-separator $\{u_1, u_2\}$, i.e., $\chi_1(v) = d(v, u_1) - d(v, u_2)$. The vertices of $\mathcal{J}_2$ are sorted in the increasing order of $\chi_2(\cdot)$ where $\chi_2(v) = d(v, u_2) - d(v, u_1)$ for all $v \in V(G')$. These two lists $\mathcal{J}_1$ and $\mathcal{J}_2$ are associated with u_1 and u_2 respectively.

It is not difficult to see that, by constructing a balanced binary search tree over $\mathcal{J}_1$ (resp. $\mathcal{J}_2$) and applying the fractional cascading technique, $F(x, V(G'))$ can be computed in $O(\log |V(G')|)$ time for any point x outside G'.

A two-level tree decomposition of G Since the tree decomposition $\mathcal{TD}(G)$ of G might not be balanced, we add another balanced tree structure over $\mathcal{TD}(G)$, such that the height of the new tree $\overline{\mathcal{TD}}(G)$ is logarithmic. We call such a balanced tree structure $\overline{\mathcal{TD}}(G)$ *a two-level tree decomposition of* G. There are several methods to achieve this, such as centroid tree decomposition, spine tree decomposition etc. We can see that a two-level tree decomposition data structure of a partial 2-tree can be computed in $O(n \log^2 n)$ time requiring $O(n \log n)$ storage space and the service cost of a set of p points in the partial 2-tree can be answered in $O(p \log^2 n)$. Thus,

Theorem 6 *Given a tree decomposition (of treewidth 2) of a partial 2-tree G, the discrete p-center problem can be solved in $O(pn^p \log^2 n)$ time.*

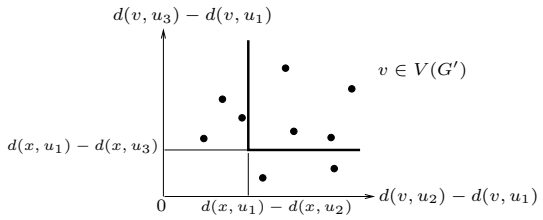


Figure 2: The set of vertices in $V(G')$ to which the shortest path from x goes through u_1.

In the case when G is a partial k-tree, given a subgraph G' represented by a subtree of $\mathcal{TD}(G)$ and a point x outside G', there is a k-separator in G' between G' and x. Refer to Figure 2 in which G is a partial 3-tree. All vertices in $V(G')$ are embedded in a 2-dimensional space (note that it is a $(k-1)$-dimensional space when G is a partial k-tree). Given a point x with its distances to the 3-separator $\{u_1, u_2, u_3\}$, all the vertices lying above and right of the bold line in Figure 2 are the vertices in $V(G')$ to which the shortest path from x goes through u_1. The service cost of x to these vertices can be computed in $O(\log |V(G')|)$ time by the combining priority search tree and the fractional cascading technique after $O(|V(G')| \log |V(G')|)$ preprocessing time.

Similarly, when G is a partial k-tree, we can compute $F(x, V(G'))$ in $O(k \log^{k-2} |V(G')|)$ time after $O(k|V(G')| \log^{k-2} |V(G')|)$ preprocessing time. Hence, for $k > 2$, a two-level tree decomposition data structure of a partial k-tree can be computed in $O(nk \log^{k-1} n)$ time requiring $O(nk \log^{k-1} n)$ storage space such that the service cost of a set of p points in the partial k-tree can be answered in $O(pk \log^{k-1} n)$.

We have the following theorem.

Theorem 7 *Given a tree decomposition (of treewidth $k > 2$) of a partial k-tree G, the discrete p-center problem can be solved in $O(pkn^p \log^{k-1} n)$ time.*

4 Future work

For the general p-center problem in which the demand set contains all points of the underlying network (i.e., $\mathcal{D}(G) = A(G)$), a candidate set containing the optimal solution value is characterized in Tamir's paper [8]. In spite of the nice structure, the size of this set is not polynomial even for simple structures such as partial 2-trees. Until now, no efficient algorithm is known for the problem in a general network or a partial k-tree.

References

[1] B. Ben-Moshe, B.K. Bhattacharya, Q. Shi, and A. Tamir, "Efficient algorithms for center problems in cactus networks", *Theor. Comput. Sci.*, 378(3):237–252, 2007.

[2] S. Cabello and C. Knauer, "Algorithms for graphs of bounded treewidth via orthogonal range searching", in *EuroCG'08*.

[3] T.M. Chan, "A (slightly) faster algorithm for Klee's measure problem", in *SoCG'08*.

[4] S. Chaudhuri and C.D. Zaroliagis, "Shortest paths in digraphs of small treewidth. part I: sequential algorithms", *Algorithmica*, 27(3):212–226, 2000.

[5] O. Kariv and S.L. Hakimi, "An algorithmic approach to network location problems. I: the p-centers", *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[6] D. Granot and D. Skorin-Kapov, "On some optimization problems on k-trees and partial k-trees", *Disc. App. Math.*, 48(2):129–145, 1994.

[7] J.A. Moreno, "A new result on the complexity of the p-center problem", *Technical Report, Universidad Complutense*, 1986.

[8] A. Tamir, "On the solution value of the continuous p-center location problem on a graph", *Math. Oper. Res.*, 12(2):340–349, 1987.

[9] A. Tamir, "Improved complexity bounds for center location problems on networks by using dynamic data structures", *SIAM J. Discret. Math.*, 1(3):377–396, 1988.

Generalized Ham-Sandwich Cuts for Well Separated Point Sets

William Steiger* Jihui Zhao‡

Abstract

Bárány, Hubard, and Jerónimo recently showed that for given well separated convex bodies $S_1, \ldots, S_d$ in $\mathbb{R}^d$ and constants $\beta_i \in [0,1]$, there exists a unique hyperplane h with the property that $\mathrm{Vol}(h^+ \cap S_i) = \beta_i \cdot \mathrm{Vol}(S_i)$; h^+ is the closed positive transversal halfspace of h, and h is a "generalized ham-sandwich cut". We give a discrete analogue for a set S of n points in $\mathbb{R}^d$ which is partitioned into a family $S = P_1 \cup \cdots \cup P_d$ of well separated sets and are in *weak* general position. The combinatorial proof inspires an $O(n(\log n)^{d-3})$ algorithm which, given positive integers $a_i \leq |P_i|$, finds the unique hyperplane h incident with a point in each P_i and having $|h^+ \cap P_i| = a_i$. Finally we show that the conditions assuring existence and uniqueness of generalized cuts are also necessary.

1 Introduction.

Given d sets $S_1, S_2, ..., S_d \in \mathbb{R}^d$, a ham-sandwich cut is a hyperplane h that simultaneously bisects each S_i. "Bisect" means that $\mu(S_i \cap h^+) = \mu(S_i \cap h^-) < \infty$, h^+, h^- the closed halfspaces defined by h and μ a suitable, "nice" measure on Borel sets in $\mathbb{R}^d$, e.g., the volume. The well known ham-sandwich theorem guarantees the existence of such a cut. As with other consequences of the Borsuk-Ulam theorem [10] there is a discrete version that applies to sets $P_1, \ldots, P_d$ of points in general position in $\mathbb{R}^d$. For example Lo et. al [9] gave a direct proof of a discrete version of the ham-sandwich theorem which inspired an efficient algorithm to compute a cut. More recently Bereg [4] studied a discrete version of a result of Bárány and Matoušek [2] that showed the existence of wedges that simultaneously equipartition three measures on $\mathbb{R}^2$ (they are called equitable two-fans). By seeking a direct, combinatorial proof of a discrete version (for counting measure on points sets in $\mathbb{R}^2$) he was able to strengthen the original result and also obtained a beautiful, nearly optimal algorithm to construct an equitable two-fan. Finally, Roy and Steiger [12] followed a similar path to obtain complexity results for several other combinatorial consequences of the Borsuk-Ulam theorem.

*Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, New Jersey 08854-8004; ({steiger,zhaojih}@cs.rutgers.edu.)

The present paper is in the same spirit. The starting point is a recent, interesting result about generalized ham-sandwich cuts.

Definition 1: (see [8]) *Given $k \leq d+1$, a family $S_1, \ldots, S_k$ of connected sets in $\mathbb{R}^d$ is well-separated if, for every choice of $x_i \in S_i$, the affine hull of $x_1, \ldots, x_k$ is a $(k-1)$-dimensional flat in $\mathbb{R}^d$.*

Bárány et.al. [1] proved

Proposition 1 *Let $K_1, ..., K_d$ be well separated convex bodies in $\mathbb{R}^d$ and $\beta_1, \ldots, \beta_d$ given constants with $0 \leq \beta_i \leq 1$. Then there is a unique hyperplane $h \subset \mathbb{R}^d$ with the property that $Vol(K_i \cap h^+) = \beta_i \cdot Vol(K_i)$, $i = 1, \ldots, d$.*

Here h^+ denotes the closed, positive transversal halfspace defined by h: that is the halfspace where, if Q is an interior point of h^+ and $z_i \in K_i \cap h$, the d-simplex $\Delta(z_1, ..., z_d, Q)$ is *negatively* oriented [1]. Specifying this choice of halfspaces is what forces h to be uniquely determined. Bárány et. al. give analogous results for such *generalized ham-sandwich cuts* for other kinds of well separated sets that support suitable measures.

We are interested in a version of Proposition 1 for n points partitioned into d sets in $\mathbb{R}^d$; i.e., points in $S = P_1 \cup \cdots \cup P_d$, $P_i \cap P_j = \phi, i \neq j$, $|S| = n$. For this context we use

Definition 2: *Point sets $P_1, ..., P_d$ are well separated if their convex hulls, $\mathrm{Conv}(P_1), \ldots, \mathrm{Conv}(P_n)$, are well separated.*

We need some kind of general position, and will assume the following weaker form.

Definition 3: *Points in $S = P_1 \cup \ldots \cup P_d$ have weak general position if, for each $(x_1, \ldots, x_d)$, $x_i \in P_i$, $aff(x_1, \ldots, x_d)$ is a $(d-1)$-flat that contains no other point of S.*

This does not prohibit more than d data points from being in a hyperplane, e.g. if they are all in the same P_i. For the discrete analogue of a generalized cut we use

Definition 4: *Given positive integers $a_i \leq |P_i|$, an $(a_1, ..., a_d)$-cut is a hyperplane h for which $h \cap P_i \neq \phi$ and $|h^+ \cap P_i| = a_i$, $1 \leq i \leq d$.*

As in Proposition 1, a cut is a transversal hyperplane (here incident with at least one data point in each P_i)

and h^+ its positive halfspace. The discrete version of Proposition 1 is

Theorem 2 *If $P_1, \ldots, P_d$ are well separated point sets in $\mathbb{R}^d$, then (i) if an $(a_1, ..., a_d)$-cut exists, it is unique. Also (ii) if the points have weak general position, cuts exist for every $(a_1, ..., a_d)$, $1 \le a_i \le |P_i|$.*

This might be proved using some results of [1] via a standard argument that takes the average of n probability measures, one centered at each data point. The variance of the measures is decreased to zero, and one argues about the limit (see [7]). Instead we give a direct combinatorial proof in Section 2. In addition, and of independent interest, we show that both well-separated and weak general position are also necessary for every possible $(a_1, ..., a_d)$-cut to exist and be unique. An analogous converse is likely to hold for Proposition 1.

There is also interest in the algorithmic problem where, given n points distributed among d well separated sets in $\mathbb{R}^d$, and in weak general position, the object is to find the cut for given $a_1, \ldots, a_d$. Our proof of Theorem 2 leads to the formulation of an efficient, $O(n(\log n)^{d-3})$ algorithm for generalized cuts. This appears in Section 3. Throughout, because of space limitations, some details are omitted.

2 Proof of the Discrete Version.

There are several equivalent forms of the well separated property for connected sets [3], in particular the fact that such a family is well separated if and only if the convex hulls are well separated. Others include

1. Sets $S_1, \ldots, S_k, k \le d+1$ are well separated if and only if, when I and J are disjoint subsets of $1, \ldots, d+1$, there is a hyperplane separating the sets $S_i, i \in I$ from the sets $S_j, j \in J$.

2. $S_1, \ldots, S_d$ are well separated in $\mathbb{R}^d$ if and only if there is no $(d-2)$-dimensional flat that meets all $\mathrm{Conv}(S_i), i = 1, \ldots, d$.

In view of Definition 2, they hold for the discrete context as well.

Given points $p_i \in \mathrm{Conv}(P_i), i = 1, \ldots, d$ (not necessarily data points in S), the hyperplane $h \equiv \mathrm{aff}\{p_1, \ldots, p_d\}$ is a transversal hyperplane of dimension $d-1$. As in Bárány et. al. [1], if a unit vector c satisfies $\langle c, p_i \rangle = t$ for some fixed constant t and for all i, the unit normal vector v of h can be chosen as either c or $-c$. The *positive transversal hyperplane* arises when v is chosen so that,

$$\det \begin{vmatrix} p_1 & p_2 & \cdots & p_d & v \\ 1 & 1 & \cdots & 1 & 0 \end{vmatrix} > 0.$$

We can write h as $\{p \in \mathbb{R}^d : \langle p, v \rangle = t\}$, and h^+, the *positive transversal halfspace*, as

$$h^+ = \{p \in \mathbb{R}^d : \langle p, v \rangle \le t\}.$$

The relation $p \in h^+$ is invariant under translation and rotation.

Proof of Theorem 2: The proof is by induction. The base case $d = 2$ is probably folklore (but see [11]). Well separated implies that points in P_1 may be dualized to (red) lines having positive slopes and those in P_2, to (blue) lines having negative slope. If a red/blue intersection q has a_1 red lines and a_2 blue lines above it, vertex q is the dual of an (a_1, a_2)-cut. It must be the unique one because the red levels have positive slope and blue ones have negative slope, proving (i).

If P_1 and P_2 also have weak general position, every red/blue intersection in the dual is a distinct vertex, $|P_1| \cdot |P_2|$ of them in all, and each is incident with just those two lines. This implies that each level in the first arrangement has a unique intersection with every level of the second, proving (ii). In fact the unique intersection can be found in linear time by adapting the prune-and-search algorithm given in [11] for intersection of median levels.

Next, suppose the claim holds for dimension $j < d$. Let π be a hyperplane that separates P_1 from $\bigcup_{i=2}^d P_i$. Fix a point $x \in \mathrm{Conv}(P_1)$, project each data point $z \in \bigcup_{i=2}^d P_i$ onto π, and write P_i' for the set of images in π of the points $z \in P_i$.

Fact 1: $P_2', \ldots, P_d'$ are $d-1$ well-separated sets in π.

If not there is a $d-3$ flat $\rho \subset \pi$ that meets all $\mathrm{Conv}(P_i'), i \ge 2$. But the span of x and ρ is a $d-2$ flat that meets all $P_1, \ldots, P_d$, a contradiction. ■

Fact 2: If $P_1, \ldots, P_d$ have weak general position and if $x \in P_1$ then $P_2', \ldots, P_d'$ have weak general position in π.

A transversal flat $\rho_x \subset \pi$ has dimension $d-2$ by Fact 1. If it contains one point x_i from each $P_i', i > 1$ *and any other* $z \in \bigcup_{i=2}^d P_i'$, then x and ρ_x span a hyperplane that violates weak general position for $P_1, \ldots, P_d$. ■

Therefore the induction hypotheses apply to $P_2', \ldots, P_d'$.

Given a point $x \in \mathrm{Conv}(P_1)$ and $(a_2, \ldots, a_d)$, a hyperplane h_x containing x is *an $(a_2, \ldots, a_d)$ semi-cut (or just a semi-cut)* if, for each $i > 1$, it is incident with a point $p_i \in P_i$ and $|h_x^+ \cap P_i| = a_i$. It's not hard to prove the following useful fact.

Lemma 3 *Given $x \in \mathrm{Conv}(P_1)$ and $(a_2, \ldots, a_d)$. If there is an $(a_2, \ldots, a_d)$ semi-cut h_x then it is unique.*

To advance the induction, fix $(a_1, \ldots, a_d)$ and suppose h_x is a cut with these values, $x \in P_1$. By Lemma 3,

it is the unique semi-cut cut containing x, so suppose there is an $(a_2, \ldots, a_d)$ semi-cut h_y through $y \in P_1$, $y \notin h_x$. h_x and h_y cannot meet in $\mathrm{Conv}(P_1)$ since any such point would be in two different $(a_2, \ldots, a_d)$ semi-cuts, violating Lemma 3. But this implies that $a_1 \neq |P_1 \cap h_y^+|$. Therefore h_x is unique, which proves (i).

Now suppose $P_1, \ldots, P_d$ have weak general position and fix $x \in P_1$ and $(a_2, \ldots, a_d)$. Projecting from x, there is a unique $(a_2, \ldots, a_d)$-cut $\rho_x \subset \pi$ by the induction hypothesis and the fact that each $z' \in \bigcup_{i=2}^d P_i'$ is the image of a distinct $z \in \bigcup_{i=2}^d P_i$. x and ρ_x span a hyperplane h_x that is an $(m_x, a_2, \ldots, a_d)$-cut, m_x denoting $|P_1 \cap h_x^+|$. Lemma 3 implies that there is no other $(m_x, a_2, \ldots, a_d)$-cut. Also, repeating this procedure for every $x \in P_1$, existence and uniqueness imply that the integers $m_x, x \in P_1$ form a permutation of $1, \ldots, |P_1|$. So for some $z \in P_1$ we have the unique $(a_1, \ldots, a_d)$-cut, and this proves (ii). ∎

In fact the conditions of the Theorem are also necessary.

Corollary 4 *Well separation and weak general position are necessary if all $(a_1, \ldots, a_d)$-cuts exist and are unique.*

Weak general position is necessary for the existence and uniqueness of all $(a_1, \ldots, a_d)$-cuts by simple counting. There are $|P_1| \cdot |P_2| \cdots |P_d|$ different d-tuples $(a_1, \ldots, a_d)$ and there are this many *different* transversal hyperplanes through data points only if we have weak general position.

Now suppose $P_1, \ldots, P_d$ are not well separated. By property 1 at the beginning of this section, there is a partition $I \cup J$ of $\{1, \ldots, d\}$, such that $A = \mathrm{Conv}(\bigcup_{i \in I} P_i) \cap \mathrm{Conv}(\bigcup_{j \in J} P_j) \neq \phi$. For points in A on the boundaries of the convex hulls, weak general position is violated. For points of A interior to both convex hulls, any half space containing $\bigcup_{i \in I} P_i$ also contains at least one point in $\bigcup_{j \in J} P_j$ in its interior. If we set $a_i = 1$ for $i \in I$, $a_i = |P_i|$ for $i \in J$, no $(a_1, \ldots, a_d)$-cut can exist. ∎

3 An Algorithm for Generalized Cuts.

From now on we assume weak general position and well separation. Theorem 2 implies that there is a unique set of data points $p_1, \ldots, p_d$, $p_i \in P_i$, for which $\mathrm{aff}(p_1, \ldots, p_d)$ is an $(a_1, \ldots, a_d)$-cut. So we could use a brute force enumeration and find it in $O(n^{d+1})$, $O(n)$ being the cost to test each d-tuple.

A small improvement can be obtained by resorting to the following algorithmic result of [9] (slightly restated to reflect new upper bounds on k-sets [6], [13]).

Proposition 2. *Given n points in $\mathbb{R}^d$ which are partitioned into d sets $P_1, \ldots, P_d$ in $\mathbb{R}^d$, a ham-sandwich cut can be computed in time proportional to the (worst-case) time needed to construct a given level in the arrangement of n given hyperplanes in $\mathbb{R}^{d-1}$. The latter problem can be solved within the following bounds:*

$$\begin{array}{ll} O(n^{4/3} \log^2 n / \log^* n) & \text{for } d = 3, \\ O(n^{5/2} \log^{1+\delta} n) & \text{for } d = 4, \\ O(n^{d-1-a(d)}) & \text{for } d \geq 5. \end{array}$$

$\delta > 0$ is an appropriate constant and $a(d) > 0$ a small constant; also $a(d) \to 0$ and $d \to \infty$.

It is not difficult to verify that the ham-sandwich algorithms given in [9] may be extended to find generalized cuts for well separated points sets having weak general position - given that they exist - and in this way, the complexity of finding generalized cuts may be reduced to $O(n^{d-1-a(d)})$.

Finally, we will describe a much more practical algorithm, applying ideas from the proof in Section 2. We showed there that for each data point $x \in P_1$ and $(a_2, \ldots, a_d)$, there is a unique $(m_x, a_2, \ldots, a_d)$-cut h_x that contains x. Furthermore, for each j, $1 \leq j \leq |P_1|$, there is a unique $x \in P_1$ for which $m_x = j$. Thus we could loop through all $x \in P_1$, project onto π, find the unique $(a_2, \ldots, a_d)$ cut $\rho_x \subset \pi$, and count $m_x = |P_1 \cap h_x^+|$ for h_x, the hyperplane spanned by x and ρ_x. At some stage we will find the $z \in P_1$ for which $m_z = a_1$ and h_z is the $(a_1, \ldots, a_d)$-cut. The cost would be bounded by the cost to solve n $(d-1)$-dimensional problems.

In fact we will find the desired $z \in P_1$ by solving at most $O(\log n)$ $(d-1)$-dimensional problems. The key is to be able to prune a fixed fraction of remaining points in P_1 after each search step, and uses the fact that if $n_x < a_1$, no point $y \in h_x^+ \cap P_1$ has $n_y = a_1$.

ALGORITHM GEN-CUT

1. **choose $c > 0$, a small, fixed integer (say 10)**
2. **Find a hyperplane π that separates P_1 from $P_2 \cup \cdots \cup P_d$**
3. $C \leftarrow P_1$
4. $a \leftarrow a_1$
5. **WHILE $|C| > c$ DO**
   - **Construct A, an ϵ-approximation to C**
   - **FOR each $x \in A$ DO**
     (a) **Project each $y \in P_2 \cup \cdots \cup P_d$ onto π; let P_i' denote the projections of the points in P_i**
     (b) **Find the $(a_2, ..., a_d)$-cut $\rho_x \subset \pi$ for the projections $P_2', \ldots, P_d'$ by solving a $(d-1)$-dimensional problem**

(c) **Get h_x, the hyperplane that spans x and ρ_x.**

(d) **Compute the number of points of C in the positive transversal halfspace h_x^+**

(e) **END FOR**

- **Prune from C points $x \in P_1$ whose m_x is too small or too large, and adjust C and a**
- **END WHILE**

6. **For each remaining data point in $x \in C$, project, find the $(a_2, ..., a_d)$-cut ρ_x in π for the projections by solving a $(d-1)$-dimensional problem, get h_x and compute $m_x = |P_1 \cap h_x^+|$, stopping when $m_x = a_1$.**

Finding a separating hyperplane π can be formulated as a linear programming problem and can be solved in time $O(n)$, for fixed dimension d. C is the set of candidates for the sought point $z \in P_1$; initially $C = P_1$. a denotes the number of undeleted points in the positive transversal halfspace of z's semicut; initially $a = a_1$.

In the while loop we construct an ϵ-approximation to C. The range space $(C, \mathcal{A})$, has VC dimension $d+1$, where $\mathcal{A}$ denotes the set of all halfspaces in $\mathbb{R}^d$ that contain some points in C. By [5], in $O(|C|)$ time [i.e., linear; in fact its $O((d+1)^{3(d+1)}(\frac{d+1}{\epsilon^2}\log\frac{d+1}{\epsilon})^{d+1}|C|)$] we can construct an ϵ-approximation $A \subset C$, having constant size [in fact $|A| = k = O(\frac{d+1}{\epsilon^2}\log\frac{d+1}{\epsilon})$].

The FOR loop is traversed $k = |A|$ times. The cost of each traversal is dominated by $O(B_{d-1})$, the cost of the $(d-1)$-dimensional problem in (b); the cost of (a) is $O(n)$ and (d) is $O(|C|)$.

At the end of the FOR we have for each $x \in A$, the value of $n_x = |h_x^+ \cap C|$. These distinct values order the elements $x \in A$, and our target value, a, is less than the smallest n_x, greater than the largest n_x, or between a successive pair in the ordering. In the first case we delete all $y \in C$, $y \notin h_u^+$, where $n_u = \min(n_x, x \in A)$. In the third case we delete all $y \in C$, $y \in h_v^+$, where $n_v = \max(n_x, x \in A)$; here we also reduce a by $a \leftarrow a - n_v$. The middle case is similar. Since A is an ϵ-approximation, only a constant fraction $(< 1/(k+1) + 2\epsilon)$ of the points in C remains after pruning.

The geometric decrease in $|C|$ implies that the number of iterations of the WHILE loop is bounded by $O(\log|P_1|) = O(\log n)$. Therefore Step 5b contributes $O(B_{d-1}\log n)$ to the total cost of the loop, where B_k denotes the complexity of the present algorithm in dimension k. This dominates the total cost of the loop because all other steps have cost either $O(n)$ or $(O|C|)$ and contribute a total of $O(n\log n)$ to the loop.

When the loop terminates, each remaining point in C is treated in time $O(B_{d-1})$ by executing Steps 5a through 5d. Then, instead of Step 5e, we test whether $|h^+ \cap P_1| = a_1$; exactly one point will have this property. Since the base case for dimension $d = 2$ has linear running time, the present algorithm will find a generalized cut in $O(n(\log n)^{d-2})$.

Finally, for $d = 3$, Lo, et. al. [9] showed how to find a ham-sandwich cut for well separated point sets in linear time. That algorithm is easily adapted to generalized cuts. Using this as the base case, the algorithm just described now has running time $O(n(\log n)^{d-3})$ for dimensions $d \geq 3$.

We tried to find a way to do the inductive step in constant time, similar to the way Lo et. al. did for separated ham sandwich cuts in R^3, but did not succeed. A main open question is whether there is an $O(n)$ algorithm for this problem.

Acknowledgement: We thank the reviewers for insightful comments.

References

[1] Bárány, I., Hubard, A., Jerónimo, J. " Slicing Convex Sets and Measures by a Hyperplane". *Discrete and Computational Geometry 39*, 67-75 (2008).

[2] I. Bárány and J. Matoušek. "Separating Measures by Two-Fans". *Discrete and Computational Geometry*

[3] I. Bárány and P. Valtr. "A positive fraction Erdős-Szekeres theorem". *Discrete and Computational Geometry 19*, 335-342 (1998).

[4] S. Bereg. "Equipartitions of Measures by 2-Fans". *Discrete and Computational Geometry 34 (1)*, 87-96 (2005).

[5] B. Chazelle: The Discrepancy Method: Randomness and Complexity, Cambridge University Press (2000).

[6] T. Dey. "Improved Bounds for Planar k-Sets and Related Problems". *Discrete and Computational Geometry 19 (3)* 373- 382 (1998).

[7] H. Edelsbrunner. Algorithms in Combinatorial Geometry. Springer, New York (1987)

[8] H. Kramer and A. Nemeth. "Supporting spheres for families of independent convex sets". *Arch. Math. 24*, 91-96 (1973).

[9] C.-Y. Lo, J. Matoušek, and W. Steiger. "Algorithms for Ham-Sandwich Cuts". *Discrete and Computational Geom. 11*, 433-452 (1994).

[10] J. Matoušek. *Using the Borsuk-Ulam Theorem.* Springer-Verlag (2003).

[11] N. Megiddo. "Partitioning with two lines in the plane". *J. Algorithms 6*, 430-433 (1985)

[12] S. Roy and W. Steiger. "Some Combinatorial and Algorithmic Aspects of the Borsuk-Ulam Theorem". *Graphs and Combinatorics 23*, 331-341, (2007).

[13] M. Sharir, S. Smorodinsky, and G. Tardos. "An Improved Bound for k-Sets in Three Dimensions". *Discrete and Computational Geometry 26*, 195-204 (2001).

Direct Planar Tree Transformation and Counterexample

Selim G Akl, Kamrul Islam, and Henk Meijer
School of Computing, Queen's University
Kingston, Ontario, Canada K7L 3N6

Abstract

We consider the problem of planar spanning tree transformation in a two-dimensional plane. Given two planar trees T' and T'' drawn on a set S of n points in general position in the plane, the problem is to transform T' into T'' by a sequence of simple changes called edge-flips or just flips. A flip is an operation by which one edge e of a geometric object is removed and an edge f ($f \neq e$) is inserted such that the resulting object belongs to the same class as the original object. Generally, for geometric transformation, the usual technique is to rely on some 'canonical' object which can be obtained by making simple changes to the initial object and then doing the reverse operations that transform the canonical object to the desired object. In this paper, we present a technique for such transformation that does not rely on any canonical tree. It is shown that T' and T'' can be transformed into each other by at most $n - 1 + k$ flips ($k \geq 0$) when S is in convex position and we also show results when S is in general position. We provide cases where the approach performs an optimal number of flips. A counterexample is given to show that if $|T' \setminus T''| = k$ then they cannot be transformed to one another by at most k flips.

1 Introduction

The problem of transforming of a certain class of geometric objects consisting of straight line segments and points in the plane, by applying small changes called flips in the objects, has been studied extensively [2, 5, 4, 6]. Given any two objects for a certain set of points, the question is whether the two objects can be transformed to each other by a sequence of flips and how many such transformations are required. A flip can be informally defined as the removal of an edge from, and insertion of another edge to, the object given. Originally, triangulations were investigated with positive results by K. Wagner [8]. Since then the problem has been studied for other classes of planar graphs such as tetrahedrons, linked-edge lists, pseudo-triangulations, planar spanning trees, crossing-free Hamiltonian paths and so on. Algorithms for such transformation as well as lower and upper bounds for achieving transformation results can be found in [1, 2, 4, 6].

One of the best-known results in the case of planar tree transformation is by Avis and Fukuda [2] who showed that for n points in general position every planar tree T' can be transformed into another planar tree T'' by means of at most $2n - 4$ flips. Later, the bound was slightly improved to $2n - m - s - 2$ in [7] (which is better if $m + s > 2$ otherwise it is at least as good as that in [2]). Here m is the maximum degree of any vertex v of T', where v is a point on the convex hull of the point set representing the vertex set and s is the degree of v in T''. Both of these approaches rely on the use of some 'canonical' tree. Informally, a canonical tree is a planar tree that has some particular characteristics such as, for example, all the vertices are directly connected to some vertex called the *root*. Surprisingly, most results related to transformations of different classes of graphs are based on the notion of some 'canonical' form of these graphs, as mentioned in [3]. The main idea of these techniques can be stated as follows: Given two objects A and B of a certain class of graph, the technique is to transform A into some canonical object C of that class by a sequence of transformations. Later, the sequence of transformations that transform B to C is reversed to obtain the desired transformation from A to B. This is an indirect approach. The main problem with this approach is that it takes a long sequence of additional flips to obtain the canonical graph even if the two objects are quite similar or they differ only in few edges.

Here we study the transformation of planar spanning trees using flips for a set S of n points in general position in the plane, avoiding the use of canonical trees. We provide results when the points are in convex position. With this approach, trees could be transformed in a more direct manner. We determine bounds on the number of transformations needed and show that an upper bound on the number of flips using this transformation is $n - 1 + k$, (k is the number of edges of one planar tree crossed by edges of the other planar tree drawn on S). We provide a counterexample where this direct approach cannot apply when S is in general position. Our algorithm obtains an optimal bound on the number of flips when there are no crossings.

The organization of the paper is as follows. In Section 2, we provide the definitions and terminologies that will be used throughout the paper. The technique of our al-

gorithm for transformation and the results of the paper are presented in Section 3 and we conclude in Section 4.

2 Preliminaries

A **graph** $G = (V, E)$ consists of a set of vertices V, and an edge set $E = \{(v_i, v_j) | v_i, v_j \in V\}$. A graph G is called **planar** if it can be drawn in the plane so that no two edges cross, except at their common vertex. If $(v_i, v_j) \in E$, then v_i and v_j are **adjacent**. Let $S = \{v_0, v_1, v_2, \cdots, v_{n-1}\}$ be a set of n points in general position (no three points are collinear) in the two-dimensional Euclidean plane. Trees are drawn in the plane where the vertices (V) and edges (E) of a tree are represented by points of S and straight line-segments. Two vertices v_i and v_j, $v_i \neq v_j$ in an embedding of G are **visible** to each other if the straight line segment $(v_i, v_j) \in E$ between them does not cross any of the edges in G. A **flip** in tree T' is the operation of removal of an edge e and addition of an edge f such that $T'' = T' \backslash \{e\} \cup \{f\}$ is a tree.

Let $\mathcal{T}(S)$ denote the set of all trees of S and the geometric tree graph $T_G(S)$ denote the graph having $\mathcal{T}(S)$ as vertex set. Two trees $T', T'' \in \mathcal{T}(S)$ are adjacent if $T'' = T' \backslash \{e\} \cup \{f\}$ for some edges e and f. In the rest of the paper, it is assumed that a tree is planar unless otherwise mentioned.

3 Tree Transformation

Let $T' = (V, E')$ and $T'' = (V, E'')$ be any two trees belonging to $\mathcal{T}(S)$. It is required to construct T'' by applying a sequence of flips one by one to T'. In general, we say that T'' can be transformed from T' by p flips if there is a set of trees $T_0, T_1, \cdots, T_p$ where $T' = T_0$ and $T'' = T_p$ such that T_{i+1} can be obtained from T_i by a single flip. This implies that for any i, T_i and T_{i+1} are adjacent in $T_G(S)$ and it is known that the diameter of $T_G(S)$ is linear. Consider Fig. 1 where the tree T'' is obtained from T' by a sequence of flips.

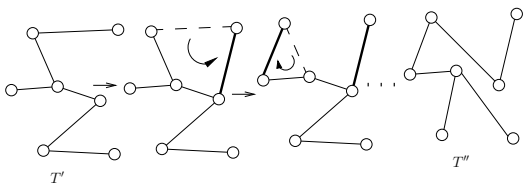


Figure 1: Transformations (shown with thick edges) applied on T' to construct T''.

In the following, we outline the main idea of our algorithm which does not rely on any form of canonical tree but obtains the desired transformation.

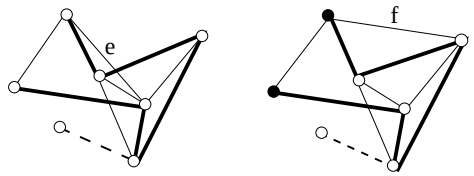
We draw two trees T' and T'' on S in the plane and obtain the graph $G = (V, E' \cup E'')$ where E' and E'' denote the edge-sets of T' and T'', respectively. Let $G_0 = G = (V, E_0 \cup E'')$ (where $E_0 = E'$). If there are p flips that transform T' into T'', our idea is to apply the sequence of flips on the edges of E' on G such that the resulting graphs are represented by $G_1 = (V, E_1 \cup E'')$, $G_2 = (V, E_2 \cup E'')$, $G_3 = (V, E_3 \cup E''), \cdots, G_p = T'' = (V, E'')$ where G_{i+1} is obtained from G_i by a single flip. In $G_i = (V, E_i \cup E'')$, E_i represents the edge set of $T_i = (V, E_i)$ being transformed into T''. Note that after the pth flip, the graph G_p turns into tree T'', since we expect that as flips are applied on the edges of T', gradually T' is turned into T'' and each instance of the intermediate trees $T_i = (V, E_i)$ along with $T'' = (V, E'')$ is reflected in G_p. In other words, we remember the order and the set of flips carried on G_i to produce G_p, then we apply these sequence of flips on T' in order to obtain T''.

To distinguish the edges of E_i from the edges of E'' in G_i, we color them with different colors. Edges $(u, v) \in E_i \backslash E''$ are colored in *red*, edges $(u', v') \in E'' \backslash E_i$ in *blue*, and edges $(u'', v'') \in E_i \cap E''$ in *purple*. Observe that, in graph G_i, only red edges can cross blue edges and there will be no crossings between red and purple or blue and purple edges since T' and T'' are planar. If a red edge is crossed by one or more blue edges, then we call it a *crossed red edge*. We count the total number of such crossed red edges after forming $G_0 = G = (V, E' \cup E'')$ at the beginning of our algorithm and denote it by k.

Lemma 1 *Suppose $G_i = (V, E_i \cup E'')$ is not planar. The removal of a crossed red edge, $e \in E_i \backslash E''$ from G_i splits E_i into two edge sets E_i' (and vertex set V_i') and E_i'' (and vertex set V_i''). Assume $CH(V) \cap V_i' \neq \emptyset$ and $CH(V) \cap V_i'' \neq \emptyset$ where $CH(V)$ is the convex hull of V. There exists an edge $f \in V \times V$ such that $G_{i+1} = (V, E_i \backslash \{e\} \cup E'' \cup \{f\})$ where f is an edge on the convex hull of V connecting a vertex of V' to a vertex of V.*

Proof. Begin by removing a crossed red edge $e = (v_k, v_\ell)$ from $G_i = (V, E_i \cup E'')$ and obtain two edge sets E_i' and E_i''. Let $V_i' \subset V$ and $V_i'' \subset V$ denote the incident vertices of E_i' and E_i'' respectively. The aim is to connect $v_i \in V_i'$ and $v_j \in V_i''$ ($(v_i, v_j) = f$) so that the edge f does not cross any edges in G_{i+1}.

Color the vertices of V_i' and V_i'' black and white, respectively. It suffices now to connect a black vertex to a white one without yielding any crossing. Select any of the black vertices v_i, $v_i \in V_i'$, on the convex hull of $CH(V)$ and start walking along the boundary of $CH(V)$ in some order. Once a walk is complete (that is, we reach the same vertex from which we started), we get a sequence of white and black vertices. We can insert an edge f by connecting any two consecutive white and black vertices in the sequence that does not generate any crossing in G_{i+1} since the edge is drawn on the boundary of $CH(V)$.



(a) $G_0 = (V, E' \cup E'')$ (b) $G_1 = (V, E_1 \cup E'')$

Figure 2: (a) Graph $G_0 = (V, E' \cup E'')$ is drawn with trees $T' = (V, E')$ and $T'' = (V, E'')$ where thin edges represent red edges, thick edges denote blue edges and dashed edges denote purple edges. (b) Edge e is removed and edge f is inserted without making any crossing in the graph. Black vertices belong to V_1 while the rest belong to V_2.

□

An illustration of the above lemma is shown in Fig.2.

We identify a case where we can obtain an optimal number of flips for the desired transformation: this is given in the following lemma.

Lemma 2 *Any tree $T' = (V, E')$ can be transformed into another tree $T'' = (V, E'')$ with at most $n - 1$ flips when the number of crossed red edges is zero.*

Proof. Obtain the graph $G_0 = (V, E_0 \cup E'')$, where $E_0 = E'$. Since there are no crossed red edges, G_0 is planar. Begin in the following way. At each step, remove an arbitrary red edge $(u, v) \in E_i \backslash E''$ from G_i and colour the vertices black and white as in the proof of Lemma 1. Insert a purple edge between a black and a white vertex, otherwise the purple edge will make a cycle if the two incident vertices are of the same color. Since at every step a flip is carried out, we get a new graph $G_{i+1} = (V, E_{i+1} \cup E'')$, where $|E_{i+1} \cap E''| = |E_i \cap E''| + 1$ $(0 \leq i < p)$. The procedure stops when $|E_p \cap E''| = n - 1$, meaning that T' has been transformed into T'' and G_p becomes $G_p = (V, E'')$. Since there can be zero purple edge in $G_0 = (V, E_0 \cup E'')$, the number of flips is at most $n - 1$. □

The above two lemmas allow us to formulate the following theorem.

Theorem 3 *Any tree $T' = (V, E')$ can be transformed into another tree $T'' = (V, E'')$ with at most $n - 1 + k$ flips where k is the number of crossed red edges, provided that for any flip $1 \leq i \leq k$, $CH(V) \cap V_i' \neq \emptyset$ and $CH(V) \cap V_i'' \neq \emptyset$.*

Proof. Consider the graph $G_0 = (V, E_0 \cup E'')$, where $E_0 = E'$. The graph can be made planar by removing all the crossings between red and blue edges, as previously shown. Thus we need at most k flips to make the graph planar provided for any flip $1 \leq i \leq k$ $CH(V) \cap V_i' \neq \emptyset$ and $CH(V) \cap V_i'' \neq \emptyset$. As the graph is made planar, we can now follow Lemma 2 to obtain T''. It takes at most $n - 1 + k$ flips to transform T' into T''. □

If the set of points are in convex position, then each flip must reduce the number of crossings between red and blue edges by at least one, since there will always be two consecutive black and white points available to make the flip successful. Now we have the following corollary:

Corollary 4 *When the set of points is in convex position we need at most $n - 1 + k$ flips for the above transformation since for any flip $1 \leq i \leq k$, $CH(V) \cap V_i' \neq \emptyset$ and $CH(V) \cap V_i'' \neq \emptyset$.*

3.1 Counterexample

In this section, we show that there exist two trees defined on the same point set such that there does not exist any flip in one of the two trees that reduces the total number of crossings by at least 1 in G_i. Such an example is shown in Fig. 3 where the tree, T' in Fig. 3(a) has three edges different from the tree, T'' in Fig. 3(b), that is, $|T' \setminus T''| = 3$. However, there is no way (as evident from Fig. 3(c)) that any of the trees can be transformed to the other by three flips. This means that the direct transformation would fail after looking for all possible removal of edge crossings and this exhaustive searching would take time proportional to the number of crossings. However, once this fails we can then resort to the technique of using a canonical tree [1] which guarantees to take at most $2n - m - s - 2$ flips for such transformation. The way the algorithm in [1] works is as follows: Let T' be the tree to be transformed into another tree T''. Let m be the maximum degree of any vertex v of T', where v is a point on the convex hull of the point set representing the vertex set. Similarly s is the degree of v in T''. We can first make T' into some canonical tree T_c where the degree of vertex v is $n - 1$ by a sequence of $n - 1 - m$ flips and then perform the flips that transform T'' into T_c by having $n - 1 - s$ flips in reverse order. Thus we incur $2n - m - s - 2$ flips for such transformation.

3.2 Remarks

The technique we present in this paper has the obvious advantage that in some cases it leads to the optimal number of flips to complete the transformation. It is well known that an approach for transforming a given tree (in general, it is true for other planar graphs of some class, e.g., planar paths, pseudotriangulations, etc.) into another via a flip operation which depends on a canonical tree never leads to the computation of

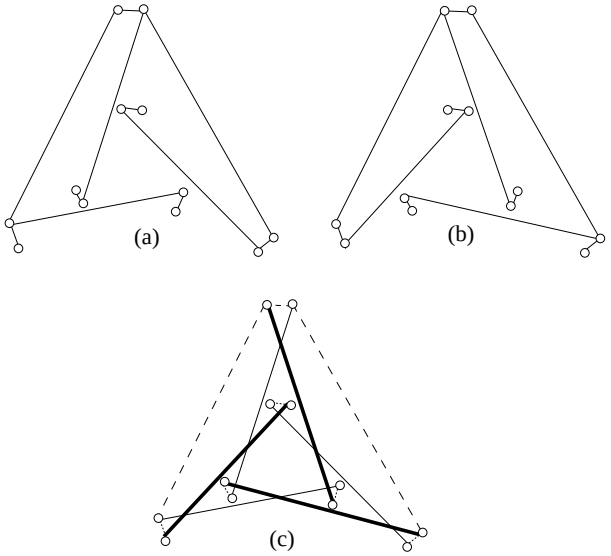


Figure 3: A counterexample with two trees defined on the same point where there does not exist any flip in one of the two trees such that the total number of crossings is reduced by at least one in G_i.

the optimal number of flips. This is because the two objects may differ only in a very small number of edges, whereas to transform them into a canonical form may take a large number of flips. As can be seen from Fig. 4, transforming any of the trees into the other takes 5 flips via a canonical tree based approach whereas only one flip suffices.

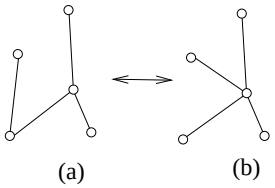


Figure 4: Transforming one tree into another takes only one flip optimally, but 5 flips through a canonical tree.

Finally, we provide a simple average case analysis of the number of flips of our algorithm. First, we determine the average number of crossed red edges. The number of crossed red edges varies from 0 to $n-3$. Thus, the total number of crossed red edges is $\sum_{k=0}^{n-3} k$ yielding the average number of crossed red edges, $\sum_{k=0}^{n-3} k/(n-2) = \frac{1}{n-2}(1+2+3+\cdots+n-3) = (n-3)/2$, where k is assumed to be uniformly distributed in $[0, n-3]$. Then the average number of flips required by our algorithm is $(n-1)+(n-3)/2 = 1.5n-2.5$. The above analysis is based on the fact that for any flip and for $1 \leq i \leq k$, $CH(V) \cap V_i' \neq \emptyset$ and $CH(V) \cap V_i'' \neq \emptyset$. However, the average-case analysis is based on the simplifying assumption that the number of crossings is uniformly distributed over a given interval. It is an interesting open problem to derive a more sophisticated value for the average number of flips required by our algorithm.

4 Conclusion

In this paper, we present a technique for tree transformation through flips when the points are in general position and also investigate the results when the points are in convex position. In this approach, we avoid the use canonical tree and directly transform one tree into another and show it takes at most $n-1+k$ flips ($k \geq 0$) for such transformation when the points are in convex position. We also show results when the points are in general position and provide an upper bound on the number of flips. If there are no crossings in the union of edges of the given trees, it is shown that this technique performs an optimal number of flips. Finally, a counterexample is given to show that if two planar trees on the same point set differ by k edges, they cannot be transformed to one another by at most k flips.

References

[1] S. Akl, K. Islam, and H. Meijer, On planar path transformation, 18th Canadian Conference of Computational Geometry (CCCG), Kingston, Ontario, August 14-16, 2006.

[2] D. Avis and K. Fukuda, Reverse search for enumeration, Discrete Applied Mathematics 65: 21-46, 1996.

[3] P. Bose and F. Hurtado, Flips in planar graphs, Computational Geometry: Theory and Applications, accepted, 2007.

[4] C. Hernando, M. E. Houle, and F. Hurtado, On local transformation of polygons with visibility properties, Theoretical Computer Science 289(2): 919-937, 2002.

[5] C. Hernando, F. Hurtado, A. Marquez, M. Mora, and M. Noy, Geometric tree graphs on points in convex position, Discrete Applied Mathematics, 93: 51-66, 1999.

[6] B. Joe, Construction of three dimensional Delaunay triangulations using local transformation, Computer Aided Geometric Design, 16: 419-453, 1991.

[7] K. Islam, Bounds on some geometric transforms, M.Sc. Thesis, 2005.

[8] K. Wagner, Bemerkungen zum vierfarbenpmblem, Jber. Deutsch. Math.-Verein. 46 (1936), 26-32.

Partitioning a Polygon into Two Mirror Congruent Pieces

Dania El-Khechen* Thomas Fevens* John Iacono† Günter Rote‡

1 Introduction

Polygon decomposition problems are well studied in the literature [6], yet many variants of these problems remain open. In this paper, we are interested in partitioning a polygon into mirror congruent pieces. Symmetry detection algorithms solve problems of the same flavor by detecting all kinds of isometries in a polygon, a set of points, a set of line segments and some classes of polyhedra [2]. Two open problems with unknown complexity were posed in [2]: the minimum symmetric decomposition (MSD) problem and the minimal symmetric partition (MSP) problem. Given a set D in R^d ($d \in 2, 3$), the goal is to find a set of symmetric (nondisjoint for MSD and disjoint for MSP) subsets $\{D_1, D_2, \ldots, D_k\}$ of D such that the union of the D_i is D and k is minimum. The following problem is a decision version of MSP where $k = 2$:

Problem 1 *Given a polygon P with n vertices, compute a partition of P into two (properly or mirror) congruent polygons P_1 and P_2, or indicate such a partition does not exist.*

Erikson claims to solve the aforementioned problem in $O(n^3)$ [4]. Rote observes that a careful analysis of Erikson's algorithm yields a $O(n^3 \log n)$ running time for proper congruence and he shows that the combinatorial complexity of an explicit representation of the solution in the case of mirror congruence cannot be bounded as a function of n [7]. Rote also gives a counterexample where the algorithm fails for a polygon with holes. An $O(n^2 \log n)$ algorithm to solve the problem for properly congruent and possibly nonsimple P_1 and P_2 was presented recently [3]. It was also conjectured that the output can be restricted to simple polygons without an increase in the runtime [3]. In this paper, we present an $O(n^3)$ algorithm to solve the problem for mirror congruent and possibly nonsimple polygons P_1 and P_2. In other words, our algorithm is able to produce solutions unbounded by n in a time polynomial in n using an implicit representation of the output. Note that we can restrict the output to simple polygons if we allow an additional linear factor for intersection checking.

*Department of Conputer Science and Software Engineering, Concordia University, Montréal, Québec, Canada.

†Department of Computer and Information Science, Polytechnic Institute of New York University, 5 Metrotech Center, Brooklyn NY 11201 USA. http://john.poly.edu. Research partially supported by NSF Grants CCF-0430849 and OISE-0334653 and by an Alfred P. Sloan Research Fellowship. Research partially completed while the author was on sabbatical at the School of Computer Science, Mcgill University, Montréal, Québec, Canada.

‡Freie Universität Berlin, Institute of Computer Science, Germany

2 Preliminaries

Two polygons are *mirror congruent* (*properly congruent*) if they are equivalent up to reflection or glide reflection (rotations and translations). Note that a glide reflection is a reflection followed by a translation parallel to the reflection axis. A reflection along an axis g followed by a rotation or a translation is a reflection around an axis g'. In this paper, we focus on mirror congruent polygons. Congruence transforms involving glide reflection are denoted by $T = (g, \mathbf{v})$ where g is the axis of reflection and $\mathbf{v}$ is the vector of translation if any. Let $T^{-1} = (g, -\mathbf{v})$. We refer to the boundary of a polygon P by $\delta(P)$ and we normalize P to have unit perimeter. A polyline that is a subset of $\delta(P)$ is specified by a start point and an endpoint on $\delta(P)$ (not necessarily vertices) and is always considered to be directed clockwise around P. A polyline can be viewed as an alternating sequence of lengths and angles, which always begins and ends with a length. Two polylines are congruent if they are represented by the same sequence, two polylines are *flip congruent* if they are represented by the same sequence after replacing all of the angles α_i in one by $2\pi - \alpha_i$ and reversing the order of the sequence, and two polylines are *mirror congruent* if they are represented by the same sequence after reversing the order of the sequence. Let $\underset{P}{\angle} a$ be the interior angle of a point a on a polygon P. Let $\overline{ab}$ be the line segment with endpoints a and b and $P[a \,..\, b]$ be the polyline connecting a to b on P in clockwise order. We use $\overset{\text{FLIP}}{\cong}$ to denote flip congruence, $\overset{\text{MIRROR}}{\cong}$ to denote mirror congruence. Observe that $P[a \,..\, b] \overset{\text{FLIP}}{\cong} P[b \,..\, a]$. Let $vd(a, b)$ be the vertical distance between the two points a and b. A partitioning of P, if it exists, is a solution to Problem 1 and is denoted by $S = (P_1, P_2)$. It consists of polygons P_1 and P_2 such that there exists a transformation where $T_S(P_1) = P_2$. The *split polyline*, denoted by Split(S), partitions the polygon P into P_1 and P_2. We are interested in a split polyline that has minimum complexity. When P is symmetric, we call the partition trivial and the problem reduces to symmetry detection which has been solved in linear time in [2]. Note if T_S is a reflection it can be determined by one pair of points $(p_i, T_S(p_i))$ such that $p_i \in \delta(P_1)$ and $T_S(p_i) \in \delta(P_2)$. If T_S is glide reflection, it can be determined by two pairs of points $(p_i, T_S(p_i))$ and $(p_j, T_S(p_j))$ such that p_i and p_j belong to $\delta(P_1)$ and $T_S(p_i)$ and $T_S(p_j)$ belong to $\delta(P_2)$. We say that two subsets $s_1 \subseteq P_1$ and $s_2 \subseteq P_2$

of congruent polygons P_1 and P_2 are transformationally congruent with respect to congruence transformation T_S if $T_S(s_1) = s_2$.

3 Results

3.1 Preprocessing

Congruence of polylines is detected by string matching. Our string representation of polygons and polylines yields Lemma 2.

Lemma 2 ([5]) *Given a polygon P, with $O(n^2)$ preprocessing and space, queries of the form $P[a \,..\, b] \overset{?}{\underset{MIRROR}{\cong}} P[c \,..\, d]$ and $P[a \,..\, b] \overset{?}{\underset{FLIP}{\cong}} P[c \,..\, d]$ can be answered in constant time.*

Let the length of a polyline $P[a \,..\, b]$ (denoted $d_P(a, b)$) be the sum of the lengths of all the segments that form this polyline. Let $d_P^{-1}(a, x)$ be the point b such that $d_P(a, b) = x$. That is, it is the point on $\delta(P)$ obtained by walking x units clockwise around $\delta(P)$ from a. Note that $d_P^{-1}(a, 0.5) = b$ is equivalent to $d_P^{-1}(b, 0.5) = a$.

Lemma 3 ([1]) *Given a polygon P, with $O(n)$ preprocessing and space, the functions d_P and d_P^{-1} can be computed in constant time if the endpoints are vertices of the given polygon, and in $O(\log n)$ if they are not, using standard point location techniques.*

3.2 Algorithms

Lemma 4 *Assume that P can be nontrivially partitioned into two mirror congruent polygons where $S = (P_1, P_2)$ and let b and e denote the endpoints of the split polyline* Split(S) *then either $P_1[b \,..\, e]$ is disjoint from the polyline $T_S(P_1[b \,..\, e])$, $T_S(P_1[b \,..\, e])$ partially overlaps with $P_1[b \,..\, e]$, or $P_1[b \,..\, e]$ and $P_2[e \,..\, b]$ are line segments.*

Proof. Suppose that $T_S(P_1[b \,..\, e]) = P_2[e \,..\, b]$. We know that by definition $P_1[b \,..\, e] \overset{FLIP}{\cong} P_2[e \,..\, b]$. Therefore, the polyline $P_1[b \,..\, e]$ and its flip congruent $P_2[e \,..\, b]$ are mirror congruent which obviously cannot happen unless $P_1[b \,..\, e]$ and $P_2[e \,..\, b]$ are line segments. □

In section 3.3, we present an algorithm for the case where Split(S) is disjoint from T_S(Split(S)) (see Figure 1) and in section 3.4, we present an algorithm for the case where they partially overlap (see Figure 2). All the proofs in the following sections are omitted due to space constraints.

3.3 Disjoint split polyline

In this section, we assume that if a solution exists then the split polyline Split(S) is disjoint from its mirror image by the transformation T_S. We first show the necessary

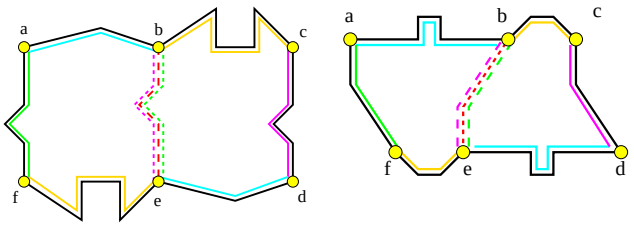


Figure 1: Polygons partitioned into two simple mirror congruent pieces with a nonoverlapping split polyline

conditions for the existence of a solution in Lemma 5, namely that a solution $S = (P_1, P_2)$ can be specified by a six-tuple of points on $\delta(P)$ satisfying some properties. In Lemma 6, we show how to verify if a given six-tuple specifies a valid solution or not. In Lemmas 7 and 8, we show how, given two points of a solution six-tuple, we can find the rest of the points in the six-tuple. In Theorem 1, given that (by Lemma 5) at least four points of a solution six-tuple are vertices, we present an $O(n^3)$ algorithm that solves Problem 1 for the case discussed in this section.

For Lemmas 5, 6, 7 and 8, assume that P can be nontrivially partitioned into two mirror congruent polygons P_1 and P_2 where $S = (P_1, P_2)$ and Split(S) is disjoint from T_S (Split(S)). Let $d = T_S(b)$, $c = T_S(e)$, $f = T_S^{-1}(b)$, and $a = T_S^{-1}(e)$.

Lemma 5 *The following facts hold (see Figure 1): (a, b, c, d, e, f) appear in clockwise order on $\delta(P)$; $P[f \,..\, a] \overset{MIRROR}{\cong} P_2[e \,..\, b]$; $P[c \,..\, d] \overset{MIRROR}{\cong} P_1[b \,..\, e]$; $P[a \,..\, b] \overset{MIRROR}{\cong} P[d \,..\, e]$; $P[b \,..\, c] \overset{MIRROR}{\cong} P[e \,..\, f]$; $P[f \,..\, a] \overset{FLIP}{\cong} P[c \,..\, d]$; $\underset{P}{\angle} a + \underset{P}{\angle} c = \underset{P_1}{\angle} e + \underset{P_2}{\angle} e$; $\underset{P}{\angle} f + \underset{P}{\angle} d = \underset{P_1}{\angle} b + \underset{P_2}{\angle} b$; at least two of the points in $\{a, c, e\}$ and two of the points in $\{b, d, f\}$ are vertices of P; $d_P^{-1}(a, 0.5) = d$; $d_P^{-1}(b, 0.5) = e$; and $d_P^{-1}(c, 0.5) = f$.*

Lemma 6 *Given the preprocessing in Lemma 2 and the positions of six points (a, b, c, d, e, f) on $\delta(P)$, it can be checked that the points specify a valid solution $S = (P_1, P_2)$ for the disjoint split polyline case of Problem 1 in constant time.*

Lemma 7 *The points (a, b, c, d, e, f) are as defined in Lemma 5. Given the position of two points of $\{a, c, e\}$ or $\{b, d, f\}$ and the preprocessing in Lemma 3, the positions of all six points (a, b, c, d, e, f) can be computed $O(\log n)$ time except in the case where both b and e are not vertices of P.*

Lemma 8 *Given the positions of $\{a, c, d, f\}$, the fact that both b and e are not vertices (equivalent to $\{a, c, d, f\}$ being all vertices by Lemma 5) and the preprocessing in Lemma 2, the positions of b and e can be computed $O(n)$ time.*

Theorem 1 *Given a simple polygon P and given that* Split(S)*, if it exists, is disjoint from T_S(Split(S)), a solution $S = (P_1, P_2)$ to Problem 1 can be found in $O(n^3)$ time if and only if P can be partitioned into two congruent polygons.*

3.4 Partially overlapping split polyline

In this section, we assume that if a solution S exists then the split polyline Split(S) is partially overlapping with its mirror image by the transformation T_S. We first show the necessary conditions for the existence of a solution in Lemma 9, namely that a solution $S = (P_1, P_2)$ can be specified by a six-tuple of points on $\delta(P)$ that obey one of two sets of properties (which we call case 1 and case 2). In Lemma 10, we show how to verify if a given six-tuple specifies a valid solution or not. In Lemmas 11 and 12, we show how, in each one of the two cases, given two points of a solution six-tuple, we can find the rest of the six-tuple points. In Theorem 2, given that (by Lemma 9) at least four points of a solution six-tuple are vertices, we present an $O(n^3)$ algorithm that solves Problem 1 for the case discussed in this section.

For Lemmas 9, 10, 11 and 12, assume that P can be nontrivially partitioned into two mirror congruent polygons P_1 and P_2 where Split(S) is partially overlapping with T_S(Split(S)). Let $T_S(e) = c$, $T_S^{-1}(b) = f$. Assume without loss of generality that the axis of glide reflection g is vertical.

Lemma 9 *The following facts hold (see Figure 2): $P[e \,..\, f] \overset{MIRROR}{\cong} P[b \,..\, c]$; $P_1[f \,..\, e] \overset{MIRROR}{\cong} P_2[c \,..\, b]$; there exists two points a and d on $\delta(P)$ such that either $P[f \,..\, a] \overset{MIRROR}{\cong} P[c \,..\, d]$, $P[a \,..\, b] \overset{FLIP}{\cong} P[d \,..\, e]$, $\angle_P(2\pi - \angle_P d) + \angle_P f = \angle_{P_1} b + \angle_{P_2} b$ and $\angle_P c + \angle_P(2\pi - \angle_P a) = \angle_{P_1} e + \angle_{P_2} e$ (this is case 1, see the left polygon in Figure 2) or $P[f \,..\, a] \overset{FLIP}{\cong} P[c \,..\, d]$, $P[a \,..\, b] \overset{MIRROR}{\cong} P[d \,..\, e]$, $\angle_P d + \angle_P f = \angle_{P_1} b + \angle_{P_2} b$ and $\angle_P c + \angle_P a = \angle_{P_1} e + \angle_{P_2} e$ (this is case 2 see the right polygon in Figure 2); at least two of the points in $\{a, c, e\}$ and two of the points in $\{b, d, f\}$ are vertices of P; if $x = (vd(b, e)/vd(f, b)) \bmod vd(c, d)$ then for case 1, x is an odd number and for case 2, x is even; (a, b, c, d, e, f) appear in clockwise order on $\delta(P)$; $d_P^{-1}(a, 0.5) = d$; $d_P^{-1}(b, 0.5) = e$ and $d_P^{-1}(c, 0.5) = f$.*

Lemma 10 *Given the preprocessing in Lemma 2 and the positions of six points (a, b, c, d, e, f) on $\delta(P)$, it can be checked that the points specify a valid solution $S = (P_1, P_2)$ for the partially overlapping split polyline case of Problem 1 in constant time.*

Lemma 11 *The points (a, b, c, d, e, f) are as defined in Lemma 9. Given the position of any two of $\{a, c, e\}$ or $\{b, d, f\}$ and the preprocessing in Lemma 3, the positions of all six points (a, b, c, d, e, f) can be computed in $O(\log n)$ time except in the cases where either both b and e or both c and f are not vertices (Figures 3 and 4).*

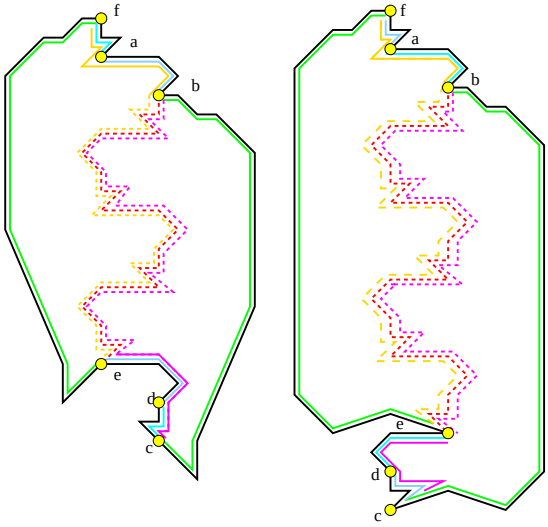


Figure 2: Polygons partitioned into two simple mirror congruent pieces with an overlapping split polyline.

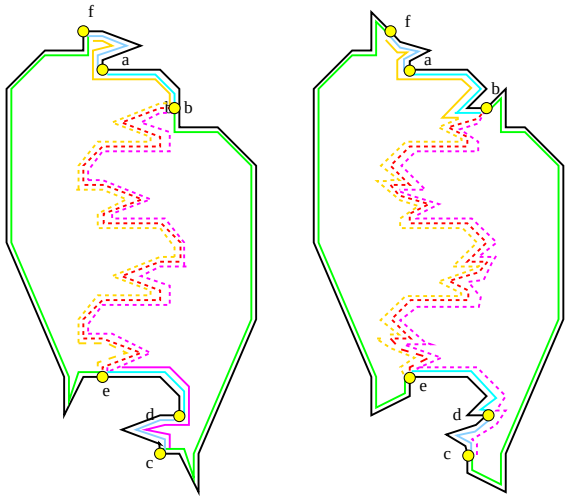


Figure 3: Case 1a (left) where $\{a, c, d, f\}$ are vertices and $\{b, e\}$ are not. Case 1b (right) where $\{a, b, e, d\}$ are vertices and $\{c, f\}$ are not.

Lemma 12 *Given the positions of $\{a, c, d, f\}$ and the preprocessing in Lemma 2, the positions of b and e can be computed in $O(n)$ time for case 1a and 1b. Similarly, given the positions of $\{a, b, d, e\}$ and the preprocessing in Lemma 2, the positions of c and f can be computed in $O(n)$ time in cases 2a and 2b.*

Theorem 2 *Given a simple polygon P and given that* Split(S)*, if it exists, is partially overlapping with T_S(Split(S)), a solution $S = (P_1, P_2)$ to Problem 1 can be found in $O(n^3)$ if and only if P can be partitioned into two congruent polygons.*

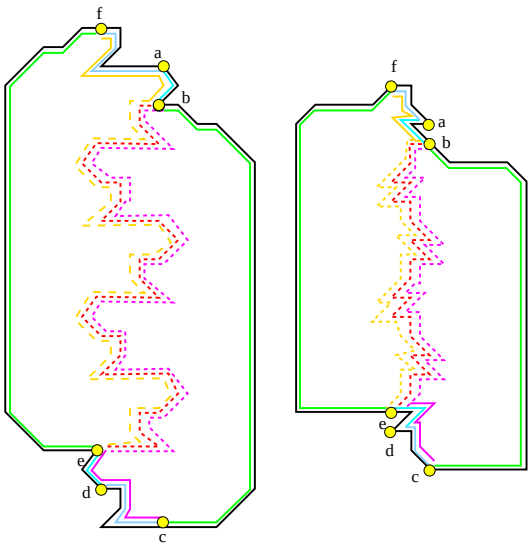


Figure 4: Case 2*a* (left) where $\{a, b, e, d\}$ are vertices and $\{c, f\}$ are not. Case 2*b* (right) where $\{a, c, d, f\}$ are vertices and $\{b, e\}$ are not.

4 Conclusion

Theorem 3 *Given a simple polygon P, we can decide if it can be partitioned into two mirror congruent polygons and find a solution $S = (P_1, P_2)$ to Problem 1, if it exists, in $O(n^3)$ time.*

References

[1] R. P. Boland and J. Urrutia. Polygon area problems. In *Canadian Conference on Computational Geometry (CCCG)*, pages 159–162, 2000.

[2] P. Eades. Symmetry finding algorithms. In G.T. Toussaint, editor, *Computational Morphology*, pages 41–51. North Holland, 1988.

[3] D. El-Khechen, T. Fevens, J. Iacono, and G. Rote. Partitioning a polygon into two congruence pieces. *Proceedings of the Kyoto International Conference on Computational Geometry and Graph Theory*, page ?, 2007.

[4] K. Erikson. Splitting a polygon into two congruent pieces. *The American Mathematical Monthly*, 103(5):393–400, 1996.

[5] M. Farach and S. Muthukrishnan. Perfect hashing for strings: formalization, algorithms and open problems. In *Symposium on Combinatorial Pattern Matching (CPM)*, 1996. `http://www.cs.rutgers.edu/~farach/pubs/tmp.html`.

[6] M. Keil. Polygon decomposition. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 11, pages 491–518. Elsevier, 2000.

[7] G. Rote. Some thoughts about decomposing a polygon into two congruent pieces. Unpublished Draft, `page.mi.fu-berlin.de/~rote/Papers/postscript/Decomposition+of+a+polytope+into+two+congruent+pieces.ps`, 1997.

The Embroidery Problem

Esther M. Arkin* George W. Hart† Joondong Kim* Irina Kostitsyna†
Joseph S. B. Mitchell* Girishkumar R. Sabhnani† Steven S. Skiena†

Abstract

We consider the problem of embroidering a design pattern, given by a graph G, using a single minimum length thread. We give an exact polynomial-time algorithm for the case that G is connected. If G has multiple connected components, then we show that the problem is *NP-hard* and give a polynomial-time 2-*approximation* algorithm. We also present results for special cases of the problem with various objective functions.

1 Introduction

An embroidery is a decorative design sewn onto a fabric using one or more threads. The artist guides the thread with a needle as it alternates between the top and the bottom of the fabric. The exposed thread on the top of the fabric is the desired design; the thread on the bottom of the design is needed only to interconnect the needle holes as the design is sewn. We study the single-thread embroidery problem in which the goal is to minimize the total length of thread.

Figure 1: Embroidery of a girl with basket.

Model. We require that the complete embroidery must be done with a single continuous piece of thread and that the thread must form a cycle, returning to the starting point (where a knot will be tied). The *embroidery problem* is graph traversal optimization problem, as we now formally state.

Problem Statement. Given a *graph* $G(V, E)$, with *vertices* V and *edges* E embedded in the Euclidean plane, find a minimum-length closed tour T with alternating edge types (*front* and *back*), such that front edges exactly cover E (without repetition) and back edges form an arbitrary subset of the edges of the complete graph on V, with possible repetitions. We assume that V is a finite set of n points in the plane and that E is a set of m straight line segments joining pairs of points in V. The *length* of an edge is its Euclidean length; the total length of a tour or a set, X, of edges is denoted $|X|$.

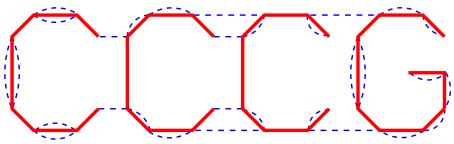


Figure 2: An embroidery graph, with red (solid) edges representing the front edges, E, of the embroidery design and blue (dashed) edges representing the back edges.

We refer to a tour T satisfying the above constraints as an *embroidery tour* for G. The front edges of T are denoted F, the back edges are denoted B. A single continuous piece of thread following T gives exactly the desired embroidery design $E = F$ (without repeating any edge) on the front of the cloth, and the back edges B of T represent "wasted" thread length. Since the edges F exactly cover E, the length of any feasible embroidery tour is simply $|T| = |E| + |B|$, so, for given E, exactly minimizing $|T|$ is equivalent to minimizing $|B|$. However, in terms of approximation ratio, the problem, OPT_T, of minimizing $|T|$ is different from the problem, OPT_B, of minimizing $|B|$.

We also consider the *Steiner* version of the embroidery problem in which we allow the set V to be augmented by a set of Steiner points that lie along edges E of the design; i.e., in the Steiner embroidery problem the set F of front edges must form an exact cover of the edges E, but each edge $e \in E$ may be (exactly) covered by a set of segments in F, with endpoints that may lie interior to e.

Related Work. The rural postman is most closely related to our problem: Given an undirected graph $G = (V, E)$ with edge weights, and a subset $E' \subseteq E$, find a closed walk of minimum weight traversing all edges of E' at least once. The stacker-crane problem is also similar, but the required edges to be traversed are directed. The main distinction between the embroidery

*Department of Applied Mathematics and Statistics, Stony Brook University, {estie,jdkim,jsbm}@ams.sunysb.edu
†Department of Computer Science, Stony Brook University, {ikost,george,gk,skiena}@cs.sunysb.edu

Graph G	OPT_T	OPT_B
Connected	poly-time Section 2.1	poly-time Section 2.1
Arbitrary	*NP-hard, 2-apx* Section 2.2	*NP-hard, 3-apx* Section 2.2
Indep Segments	*NP-hard, 1.5-apx,* Section 2.3	*NP-hard, 2-apx* Section 2.3

Table 1: Summary of results: No Steiner points allowed.

Graph G	OPT_T	OPT_B
Connected	poly-time Section 3.1	poly-time Section 3.1
Arbitrary	*NP-hard, 2-apx, PTAS* Section 3.2	*NP-hard, 3-apx* Section 3.2

Table 2: Summary of results with Steiner points.

problem and these related problems is that in the embroidery problem the tour is not allowed to traverse two of the required edges in a row; it must alternate between the front (specified) and back edges. The rural postman has a (Christofides-like) 3/2-approximation [3] and the stacker-crane has a 9/5-approximation [4]. Biedl [2] studies the special case of the embroidery problem in which only "cross-stitches" are used.

Summary of Results. Table 1 summarizes our results on the OPT_T and OPT_B problems for different types of input embroidery graphs G: (i) connected, (ii) arbitrary, with possibly many connected components, and (iii) an independent set of edges – no two edges of E share an endpoint (however, the line segments that embed E may cross arbitrarily). Table 2 lists our results for the Steiner embroidery problem.

2 Embroidery Without Steiner Points

An embroidery tour T alternates between front edges and back edges. Hence $\forall v \in V$ the number of back edges incident to v must be exactly equal to the number of front edges incident to v. See Theorem 1.

Theorem 1 *T is an embroidery tour for $G(V,E)$ if and only if $G(V,T)$ is connected and $\forall v \in V$: $d_F(v) = d_B(v)$, where $d_F(v)$ is the degree of vertex v in $G(V,F)$.*

Proof. *If:* Since T is an embroidery tour (using single continuous thread) $G(V,T)$ must be connected. If there exists a vertex v such that $d_B(v) < d_F(v)$, by the pigeon-hole-principle on entry and exit type of edges on v, T must have two consecutive front edges $e_i, e_j \in F$ sharing v, hence contradicting that T is embroiderable. A similar contradiction holds if $d_B(v) > d_F(v)$.

Only If: Since $G(V,T)$ is connected and $d_T(v)$ is even there exists an Euler tour in $G(V,T)$. We show how to construct an Euler tour that alternates edges from F and B. First, we show that $G(V,T)$ must contain an edge-alternating circuit. Start an edge-alternating walk $W = \{a, \ldots, v, x \ldots, y, v\}$ from an arbitrary vertex, until a vertex v repeats. This defines an edge-alternating circuit, unless edges (v,x) and (y,v) belong to the same side. But if so, W can be continued, as there remains at least one unused alternate side edge incident on v. Thus, we can decompose $G(V,T)$ into a set of edge-disjoint, alternating circuits. Any two such circuits (say c_1 and c_2) incident on a common vertex v' can be merged to form a larger alternating circuit, since both c_1 and c_2 contain front and back edges at v'. Repeated merging operations reduce the set of alternating circuits to a single alternating tour. □

2.1 One Connected Component

If G is connected, then the embroidery problem can be solved as follows: Find a minimum-length set of back edges B such that the degree requirement $\forall v \in V$: $d_B(v) = d_E(v)$ is satisfied. By Theorem 1 $E \cup B$ is an embroidery tour for G. Since B is minimum-length, the resulting tour is optimal. Thus, the selection of an optimal B is exactly the minimum-weight b-*matching* problem on V, with vertex weights (degrees) $b(v) = d_E(v)$, $\forall v \in V$, which is solvable in polynomial time [1].

2.2 Multiple Connected Components

Consider now an arbitrary design G, with possibly many connected components. As before, we can compute a minimum-weight b-matching, with vertices weighted by the degrees, $d_E(v)$; however, an optimal b-matching does not result in a set B of back edges that yields a complete solution, since the graph $G(V, E \cup B)$ may be disconnected.

In fact, we show that it is NP-hard to solve OPT_T or OPT_B exactly, using a simple reduction from Euclidean TSP ([6]):

Theorem 2 *The embroidery problem (either OPT_T or OPT_B) is NP-hard for arbitrary graphs G, with many connected components.*

Approximating OPT_T. We turn now to approximating OPT_T. We define a new graph $G'(V', E')$, where V' is the set of connected components in $G(V,E)$, and E' is the set of edges in the complete graph on V'. For each edge $e(i,j) \in E'$, the weight of the edge $w(i,j) = \min_{u \in V_i, w \in V_j} dist(u,w)$. Let MST be a minimum spanning tree of G'.

Now initialize B to contain a copy of front edges E and two copies of each MST edge. Note that each MST

edge is a minimum-weight edge connecting the appropriate vertices in the two different components. Let $T_{apx} = E \cup B$. Hence, $\forall v \in V, d_{T_{apx}}(v) \geq 2 \cdot d_E(v)$, implying

$$|T_{apx}| = 2 \cdot |E| \ + \ 2 \cdot |MST|.$$

Theorem 3 *T_{apx} can be converted to an embroidery tour for $G(V, E)$ with length $\leq 2 \cdot OPT_T$.*

Proof. By the definition of T_{apx}, $G(V, T_{apx})$ is connected (since it uses the MST edges to connect between different connected components) and $d_{T_{apx}}(v)$ is even. Also since $\forall v \in V, d_{T_{apx}}(v) \geq 2 \cdot d_E(v)$, we can find an Euler tour in $G(V, T_{apx})$, such that there are no consecutive front edges. We also try to "avoid" consecutive back edges, by choosing to leave a node on a front edge, if it was entered on a back edge, if possible. Note that T_{apx} may have consecutive back edges $e_i(v_i, v), e_j(v, v_j) \in B$ at a vertex v, in which case we can shortcut using $e'(v_i, v_j)$ and update $T_{apx} := T_{apx} \cup \{e'\} \setminus \{e_i, e_j\}$ without increasing $|T_{apx}|$ (by triangle inequality). Since all front edges touching v had already been used, we know that this shortcut does not disconnect v from the tour. Thus we can convert T_{apx} to a tour containing alternate front and back edges to make it an embroidery tour without increasing its cost.

Also, $OPT_T = |T_{opt}| \geq |E| + |MST|$, since T_{opt} must cover all the edges in E and must also span all the connected components and by definition of MST, it is the cheapest way to connect the disconnected components. Thus, $2 \cdot OPT_T \geq 2 \cdot (|E| + |MST|) \geq |T_{apx}|$. □

Approximating OPT_B. We start by finding a minimum-weight *b-matching* M_b of V with weight $b(v) = d_E(v), \forall v \in V$. Then for all connected components $V_1, V_2, \ldots, V_k$ in graph $G(V, E \cup M_b)$, we find the MST on graph $G'(V', E')$, as we did in Section 2.2. Now add a copy of each M_b edge and two copies of each MST edge to B. Let $T_{apx} = E \cup B$. Again, $\forall v \in V, d_{T_{apx}}(v) \geq 2 \cdot d_E(v)$. Thus,

$$|B| = |M_b| \ + \ 2 \cdot |MST|.$$

Theorem 4 *T_{apx} can be converted to an embroidery tour for $G(V, E)$ with back edges of length $\leq 3 \cdot OPT_B$.*

Proof. By similar arguments as in the proof of Theorem 3, we can convert T_{apx} to an embroidery tour with $|B| \leq |M_b| + 2 \cdot |MST|$. Now, $OPT_B = |B_{opt}| \geq |M_b|$, since in T_{opt}, B_{opt} is one b-matching satisfying $b(v) = d_E(v)$ and M_b is a minimum-weight b-matching. Also $OPT_B \geq |MST|$, since T_{opt} must span all of the connected components of $G(V, E)$. Note that MST here is a minimum spanning tree on the connected components of $G(V, E \cup M_b)$, which has smaller cost as compared to the minimum spanning tree on connected components of $G(V, E)$. Thus, $3 \cdot OPT_B \geq |M_b| + 2 \cdot |MST| \geq |B|$. □

2.3 Independent Segments

In the case that the edges E do not share endpoints (i.e., they form a set of possibly intersecting line segments), OPT_T can be approximated using the 3/2-approximation algorithm for the rural postman: If the approximating tour uses two consecutive back edges, then we simply shortcut, replacing the two edges with one shorter back edge. This results in a 3/2-approximation for OPT_T.

3 Embroidery with Steiner Points

It may be possible to use a shorter thread if we allow a front edge to be split into two or more subsegments by placing *Steiner* points judiciously along it. In fact, by placing a Steiner point arbitrarily close to an endpoint (vertex) of a front edge, we can make the length of back edges arbitrarily close to zero; see Figure 3. We say that such a Steiner point *doubles* the vertex where it is placed.

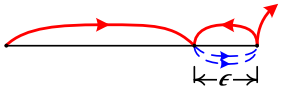


Figure 3: Placing a *Steiner* point near a vertex.

3.1 One Connected Component

Lemma 5 *There exists an optimal embroidery tour T_{opt} (allowing Steiner points) for a connected $G(V, E)$ that does not have Steiner points on edges other than those near endpoints that double vertices.*

Proof. If s is a *Steiner* point interior to an edge $e \in E$, with back edges $e(a, s), e(s, b) \in B$ incident to s, then we can simply replace these two edges with a single (back) edge $e(a, b)$ (and remove Steiner point s) without increasing the cost of tour $|T_{opt}|$. See Figure 4. □

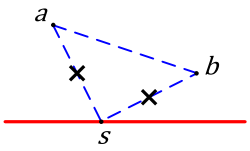


Figure 4: An optimal tour T exists without having a Steiner point s interior to an edge.

Lemma 6 *There exists an optimal tour T_{opt} (allowing Steiner points) for a connected $G(V, E)$ that does not have two back edges $e_i, e_j \in B$ incident to a common vertex $v \in V$.*

Proof. Similar to the proof of Lemma 5. □

Since an optimal embroidery tour T_{opt} is an *Euler* cycle (by definition of T), the sum of front and back edge degrees for each vertex is even. Thus, all odd-degree vertices $v \in V$ (if any present) have one back outgoing edge, and even-degree vertices do not have any outgoing back edges (using Lemma 5, 6). Therefore, an optimal solution is a union of front edges and back edges constituting minimum-weight perfect matching edges built on odd-degree vertices. The problem hence reduces to finding a minimum-weight perfect matching in a complete graph (of odd-degree vertices in this case), which can be solved in time $O(n^3)$.

3.2 Multiple Connected Components

Clearly, the same NP-hardness reduction for the non-Steiner version applies also if we allow Steiner points.

Approximating OPT_T. The idea is very similar to Section 2.2, except that the graph $G'(V', E')$ (defined over different components) has edge weights $w(i,j) = \min_{u \in G(V_i,E), w \in G(V_j,E)} dist(u,w), \forall e(i,j) \in E'$ (where u, w are edges). We refer to this minimum spanning tree on this new graph $G'(V', E')$ as MST_{St}. As before, we add a copy of front edges, F in this case (since each edge e from E that contains one or more Steiner points, gets split and is put as two ore more segments in F) and two copies of each MST_{St} edge to B. Note that $|F| = |E|$, as F exactly covers E. Let $T_{apx} = F \cup B$. Thus,

$$|T_{apx}| = 2 \cdot |E| \ + \ 2 \cdot |MST_{St}|$$

Theorem 7 *T_{apx} can be converted to an embroidery tour (allowing Steiner points) for $G(V,E)$ with length $\leq 2 \cdot OPT_T$.*

Proof. We note that every time we introduce a Steiner point, we create a new vertex v' with $d_F(v') = 2$. Since the introduction of a Steiner point is only because of some MST_{St} edge and because we double the MST_{St} edge, $d_{T_{apx}}(v') \geq 2 \cdot d_E(v')$ for all new Steiner points v'. Excluding other details (which are similar to those in Theorem 3), T_{apx} can be converted to an embroidery tour without increasing its cost.

Also, as before, $OPT_T = |T_{opt}| \geq |E| + |MST_{St}|$. Thus, $2 \cdot OPT_T \geq 2 \cdot (|E| + |MST_{St}|) \geq |T_{apx}|$. □

Approximating OPT_B. This idea is also very similar to Section 2.2, except that, instead of M_b, it uses a perfect matching M on odd-degree vertices in $G(V,E)$. It also uses MST_{St} defined in Section 3.2. We add a copy of each edge of M and two copies of each MST_{St} edge to B. Let $T_{apx} = F \cup B$, where F is the front edge cover of Steiner point split edges in E. Then,

$$|B| = |M| \ + \ 2 \cdot |MST_{St}|.$$

Theorem 8 *T_{apx} can be converted to an embroidery tour (allowing Steiner points) for $G(V,E)$ with back edges of length $\leq 3 \cdot OPT_B$.*

Proof. We apply shortenings to B as in Lemmas 5, 6. The result is a perfect matching of odd-degree vertices of $G(V,E)$. Thus, $OPT_B \geq |M|$, since the cost of any perfect matching is at least as much as the cost of the minimum weight perfect matching. Also $OPT_B \geq |MST_{St}|$, since T_{opt} must span all of the connected components of $G(V,E)$. Thus, $3 \cdot OPT_B \geq |M| + 2 \cdot |MST_{St}| \geq |B|$. □

3.3 A PTAS

By using the m-guillotine method for geometric network approximation [5], we obtain a PTAS for the problem:

Theorem 9 *The embroidery problem with Steiner points and an arbitrary input graph G has a PTAS for OPT_T.*

3.4 OPT_{St} vs OPT_{NSt}

We analyze how much one actually gains by allowing Steiner points to be inserted on front edges:

Theorem 10 $OPT_{St} \geq \frac{1}{2} OPT_{NSt}$.

Acknowledgments

We thank Bonnie Skiena for suggesting the problem. We thank the anonymous referees for their helpful suggestions. This work was partially supported by grants from the National Science Foundation (EIA-0325123, CCF-0431030, DBI-0444815, CCF-0528209, CCF-0729019), NASA Ames, and Metron Aviation.

References

[1] R. P. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

[2] T. Biedl, J. D. Horton, and A. Lopez-Ortiz. Cross-stitching using little thread. In *17th Canadian Conference of Computational Geometry*, pages 196–199, 2005.

[3] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part II: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

[4] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

[5] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28:1298–1309, 1999.

[6] C. H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

Computational Balloon Twisting: The Theory of Balloon Polyhedra

Erik D. Demaine* Martin L. Demaine* Vi Hart†

Abstract

This paper builds a general mathematical and algorithmic theory for balloon-twisting structures by modeling their underlying edge skeleta, evolving classic balloon animals into the new world of balloon polyhedra.

What if Euler were a clown?

1 Overview

Balloon twisting (or *balloon modeling*) is a form of sculpture rooted in the magic community starting in the 1930s [1]. Modern balloon twisters gather at the annual *Twist & Shout* convention[1] and are the subject of an excellent documentary [5]. In this paper, we investigate the geometric and algorithmic nature inherent in this art form, founding the new field of *computational balloon twisting*. We use this perspective to design a new class of balloon-twisted sculpture called *balloon polyhedra*.

We begin with the basics of practical balloon twisting (Section 2) and their mathematical idealizations called "bloons" (Section 3). Then we consider the mathematics of three such models in turn: simple twisting (Section 4), pop twisting (Section 5), and equalizing bloon lengths (Section 6). Finally, we find optimal constructions for Platonic and Archimedean solids (Section 7).

In addition to artistic applications, computational balloon twisting has potential applications to building architectural structures. Our results suggest that a long, low-pressure tube (called an *air beam* in architecture) enables the temporary construction of inflatable shelters, domes, and many other polyhedral structures, which can be later reconfigured into different shapes and re-used at different sites.

2 Balloon Basics

The majority of balloon twisting starts from a long, narrow balloon, the most common being the "260" which measures 2 inches in diameter and 60 inches in length when fully inflated. Normally the balloonist only partially inflates such a balloon, however, leaving one end deflated as in Figure 3(a). This deflated end leaves room for the air to spread out when *twisting* the balloon along a circular cross-section, forming a *vertex* as in Figure 3(b). The vertex holds its shape if wrapped around another vertex, as in Figure 3(c). The figure shows the vertex coming from another balloon, but it could just as well come from another part of the same balloon, as in the middle of Figure 3(d). Indeed, one theme in balloon twisting is designing complex figures (often animals) from a single balloon, and in this paper we often aim for this goal or for minimizing the number of balloons. Vertex joints can also be bent, similar to joints in a linkage, and will hold their shape if the linkage forces them to remain bent by a nontrivial angle, as on the right of Figure 3(d).

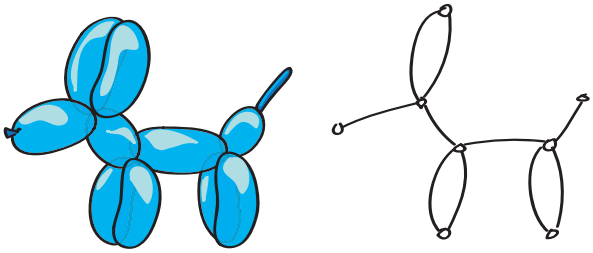


Figure 1: Classic dog (one balloon).

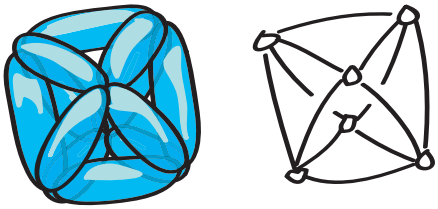


Figure 2: Octahedron (one balloon).

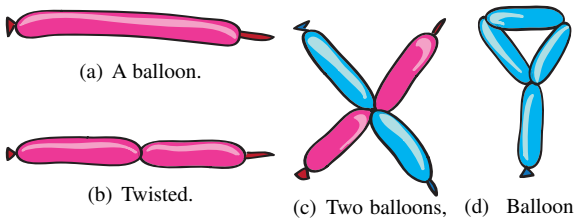
3 Twistable Tangles: Bloon Models

Inflated balloon segments and their twisted end vertices naturally form a graph. Our central problem is to determine which graphs are *twistable* under a variety of abstract models of physical balloons, which we refer to as "bloons" for contrast.

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine,mdemaine}@mit.edu

†Stony Brook University, Stony Brook, NY 11794, USA, vi@vihart.com

[1] http://www.balloonconvention.com/



(a) A balloon. (b) Twisted. (c) Two balloons, twisted. (d) Balloon twisted into a triangle.

Figure 3: Twisting balloons.

In general, a *bloon* is a segment which can be *twisted* at arbitrary points to form *vertices* at which the bloon can be bent like a hinge. The endpoints of a bloon are also vertices. Two vertices can be *tied* to form permanent point connections. A twisted bloon is *stable* if every vertex is either tied to another vertex or held at a nonzero bending angle.

We distinguish two main models of bloon twisting:

1. *(Simple) twisting*: Every subsegment of a bloon between two vertices form an edge in the associated graph, representing an inflated portion of a balloon.
2. *Pop twisting*: Some subsegments of a bloon between two vertices can be marked as *deflated*, causing them not to appear in the associated graph. Such deflated segments can be achieved with physical balloons by squeezing the air down the balloon or by popping a segment between two existing vertices (a practice common in balloon twisting, though requiring some care and skill).

Two other parameters shape the model:

3. *Number of bloons*: In general we allow structures consisting of any number k of bloons. Of particular interest are the case $k = 1$ and minimizing the number of bloons. A graph has *bloon number* k if it can be simply twisted from k bloons and no fewer.
4. *Bloon lengths*: For multibloon structures, we prefer the bloons to have the same or similar lengths. In particular, this constraint helps us avoid the need for extremely long balloons (which are difficult to obtain). An *ℓ-bloon* is a bloon of length ℓ. We often consider graphs whose edges have unit length, and hence particular cases like *doubloons* ($\ell = 2$) and *demidoubloons* ($\ell = 1$) are of interest.

4 Euler Outgrowth: Bloon Number

Simple twisting of a single bloon naturally forms an Eulerian tour of the constructed graph. Thus single-bloon graphs must have vertices of even degree, except possibly for two odd degrees, and must be connected. Indeed, such graphs are always twistable:

Theorem 1 *A graph has bloon number* 1 *if and only if the graph is Eulerian.*

More interesting from a technical standpoint is the case of k bloons (of arbitrary lengths). Here we can exactly characterize bloon number:

Theorem 2 *A graph with $o > 0$ odd vertices has bloon number $o/2$.*

Proof. Every odd-degree vertex must have an odd number of bloon ends, and each bloon has only two ends, so $o/2$ bloons are necessary. To see that $o/2$ bloons suffice, consider adding $o/2$ edges connecting the odd-degree vertices in pairs. (Recall that every graph has an even number of odd-degree vertices.) The resulting graph has all even degrees and hence an Euler tour. Removing the $o/2$ added edges from the tour results in $o/2$ paths, which are the desired bloons. □

5 Chinese Connection: Pop Twisting

Pop twisting is of course the more general model: it allows building any graph (without straight degree-2 vertices) from a single bloon. In this context, the natural objective is to minimize the total deflated length of the bloon, or equivalently, the total length of the bloon.

This problem is similar to the *Chinese Postman Problem*: given a graph, find a tour of minimum length that visits all edges. This problem has a classic polynomial-time solution based on adding to the graph a minimum-cost perfect matching of the complete graph K_o on the o odd-degree vertices, resulting in the cheapest Eulerian supergraph. The costs in the complete graph can be defined by shortest paths in the graph (for hiding deflated segments against inflated segments), or to include shortcuts available to the bloon in 3D.

The difference is that a pop twisting of a polyhedron requires a path, while the Chinese postman finds the optimal tour (cycle). To find the optimal path, we instead add the minimum-cost $(o/2 - 1)$-edge matching in K_o, leaving exactly two odd vertices. More generally, if we are given k bloons instead of one, we can add a minimum-cost $(o/2 - k)$-edge matching, leaving exactly $2k$ odd vertices; by Theorem 2, the resulting graph can be traversed by k paths. Such a matching can be computed as a minimum-cost maximum flow in the complete bipartite graph $K_{o,o}$, with edge costs defined as in K_o, together with a source of capacity $o/2 - k$ attached to one side of the bipartition via edges of capacity 1, and a sink attached to the other side of the bipartition via edges of capacity 1.

Theorem 3 *There is a polynomial-time algorithm that, given a graph and a desired $k \geq 1$, finds the k bloons of minimum total length that pop-twist the graph.*

6 Length Limitations: Holyer's Problem

Given a graph simply twistable from k bloons, how similar in length can the k bloons be? In particular, when can the lengths all be identical? We can specialize further to obtain a clean combinatorial problem by supposing graph edges all have unit length, as in regular polyhedra, and the bloons have integer length ℓ. What graphs can be simply twisted from ℓ-bloons?

This problem is closely related to *Holyer's problem*: decide whether the edges of a graph can be decomposed into copies of a fixed graph H. In 1981, Holyer [8] conjectured that this problem is NP-complete if H has at least three edges. This conjecture turns out to be correct when H is connected. In fact, the problem is NP-complete if H has a connected component consisting of at least three edges [3], and otherwise it can be solved in polynomial time [2]. Of particular relevance is an old result that every graph with an even number of edges can be decomposed into length-2 paths [11]:

Theorem 4 *Every graph with unit edge lengths can be twisted from doubloons and possibly one demidoubloon (when the graph has an odd number of edges).*

For $\ell > 2$, however, there is a discrepancy between Holyer's problem and simply twisting from ℓ-bloons. On the one hand, each ℓ-bloon can be twisted into any Eulerian graph on ℓ edges. On the other hand, Holyer's problem assumes all bloons form the same such graph, e.g., a path of ℓ edges or a cycle of ℓ edges. Therefore the known NP-hardness for Holyer's problem beyond two edges does not immediately imply NP-hardness for simple twisting beyond doubloons. Fortunately, one NP-hardness proof for Holyer's problem also establishes NP-hardness of simple twisting:

Theorem 5 *It is NP-complete to decide whether a planar bipartite graph with unit edge lengths can be simply twisted from ℓ-bloons.*

Proof. Dyer and Frieze [4, Theorem 3.4] prove NP-hardness of Holyer's problem when the graph to decompose is planar and bipartite and the pattern graph H is a path of length $\ell > 2$. Their reduction has the additional feature that all cycles have length larger than ℓ, and hence no ℓ-bloon could form a structure other than a path of length ℓ. □

We can specialize even further and still obtain NP-hardness. Theorem 2 characterizes the fewest bloons required for simple twisting. When can these fewest bloons have the same length?

Theorem 6 *It is strongly NP-complete to decide whether a planar 3-connected graph with o odd vertices can be simply twisted from $o/2$ equal-length bloons.*

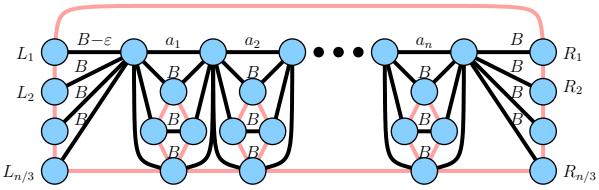


Figure 4: NP-hardness of using the fewest possible equal balloons.

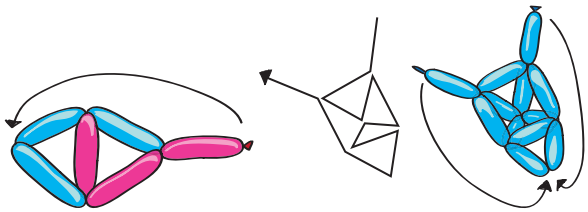
Proof. Figure 4 shows a reduction from 3-partition: given integers $a_1, a_2, \ldots, a_n$, partition into triples of equal sum. The light portion of the graph just makes the graph 3-connected. The dark portion consists of $n/3$ odd-degree vertices on the left, $L_1, L_2, \ldots, L_{n/3}$, and $n/3$ odd-degree vertices on the right, $R_1, R_2, \ldots, R_{n/3}$. All other vertices will have even degree. We can imagine building $n/3$ paths between corresponding L_i and R_i, for $1 \le i \le n/3$, and then pinching these paths together at $n+1$ meeting points. Then a left-to-right path has a choice at each meeting point of which path to follow. Exactly one path can follow an edge of length a_i; the others follows paths of total length B. Here $B > a_1 + a_2 + \cdots + a_n$. Thus each path must visit an equal number of B's, i.e., $n - n/3 + 2$ of them. The path including L_1 has a special edge of length ε less. Here $\varepsilon < \min\{a_1, a_2, \ldots, a_n\}$ and the total length of the light portion of the graph is ε. Thus only this path can visit light edges, and must visit all. Therefore the graph can be twisted by $n/3$ equal-length bloons if and only if the 3-partition instance has a solution. □

By suitable scaling, we can make all edge lengths integers, and then subdivide edges into unit lengths. It seems somewhat difficult, however, to make the graph 3-connected by adding a suitable light Eulerian graph.

Some positive results are known for special cases of Holyer's problem. For example, every 4-regular connected graph whose number of edges is divisible by 3 can be decomposed into paths of length 3, and hence simply twisted from tribloons [7]. The same decomposition and twisting results hold for triangulated (maximal) planar graphs with at least four vertices [6]. It is conjectured that every simple planar 2-edge-connected graph whose number of edges is divisible by 3 can be decomposed into paths and cycles of length 3, and hence simply twisted from tribloons [9]. See also [10]. But relatively few results are known for sizes larger than 3.

7 Polyhedral Projects: Balloon Polyhedra

In contrast to the hardness result of Theorem 6, we show that every Platonic and Archimedean solid can be twisted using the bloon number of bloons, $o/2$, all of equal length. Furthermore, these solids can be twisted



(a) Tetrahedron construction. (b) Octahedron construction.

Figure 5: Constructing two Platonic solids.

so that the component bloon units are all isomorphic and arranged in a symmetric manner. This property makes these polyhedra particularly easy to construct, and lends itself well to color patterns. See Figure 6.

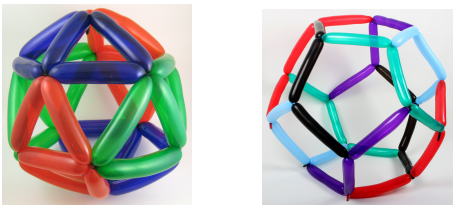
Figure 5 shows how to construct two Platonic solids: the tetrahedron and octahedron. We consider the icosahedron below because it can also be viewed as a snub tetrahedron. The cube and dodecahedron are both possible with tribloons, and together with the tetrahedron are special in that the bloon units can have only dihedral symmetry. In contrast, the icosahedron construction has pyrite symmetry, while the Archimedean constructions below (and the octahedron) have the same symmetry group as the original polyhedron.

The Archimedean solids can be categorized into three different groups for our purposes: Eulerian, truncated, and snub. The Eulerian case is of course trivial. For truncated polyhedra, the optimal bloons are tribloons, because each original edge truncates to create two vertices and three edges, yielding a $2:3$ vertex-edge ratio. The tribloons can be embedded as Zs (or Ss, as the result is chiral), where each center edge aligns with an edge of the original (untruncated) polyhedron, with the arms bending to form the truncated faces. The snub polyhedra (including the icosahedron) can be made from a common unit, namely, a quintibloon twisting into the shape of two triangles sharing an edge.

Of course, not all polyhedra can be made from the bloon number of bloons, $o/2$, of equal length. The pentagonal pyramid is a simple example. It has ten edges and six vertices, all of odd degree, yielding a bloon number of 3. Unfortunately, 10 is not divisible by 3, so one bloon must have length 4. In the realm of polyhedra with icosahedral symmetry, the simplest counterexample is the rhombic triacontahedron, with 60 edges and 32 vertices, which again do not divide evenly. It remains open whether any polyhedron fails to have a twisting from a bloon number of equal-length bloons but not by virtue of indivisibility. It also remains open whether some symmetric polyhedron can be twisted only from nonidentical units or only from units arranged asymmetrically.

Acknowledgments. We thank the anonymous referees for helpful comments.

(a) Tetrahedron (two balloons). (b) Cube (four balloons). (c) Octahedron (one balloon).



(d) Icosahedron (six balloons). (e) Dodecahedron (six balloons).

Figure 6: Balloon polyhedra.

References

[1] Balloon HQ. The history of balloons. http://balloonhq.com/faq/history.html, 2002. See also http://www.tmyers.com/marr.html.

[2] K. Bryś and Z. Lonc. Polynomial cases of graph decomposition: a complete solution of Holyer's problem. Submitted. Announced at *4th Twente Workshop on Graphs and Combinatorial Optimization*, June 1995, Enschede, The Netherlands.

[3] Dorit Dor and Michael Tarsi. Graph decomposition is NP-complete: A complete proof of Holyer's conjecture. *SIAM Journal on Computing*, 26(4):1166–1187, 1997.

[4] M. E. Dyer and A. M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, February 1985.

[5] Naomi Greenfield and Sara Taksler. Twisted: A balloonamentary. 79-minute film, 2007. http://twistedballoondoc.com/.

[6] Roland Häggkvist and Robert Johansson. A note on edge-decompositions of planar graphs. *Discrete Mathematics*, 283(1–3):263–266, June 2004.

[7] Katherine Heinrich, Jiping Liu, and Minli Yu. P_4-decompositions of regular graphs. *Journal of Graph Theory*, 31(2):135–143, 1999.

[8] Ian Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, November 1981.

[9] M. Jünger, G. Reinelt, and W. R. Pulleyblank. On partitioning the edges of graphs into connected subgraphs. *Journal of Graph Theory*, 9(4):539–549, December 1985.

[10] Alexandr Kostochka and Vladimir Tashkinov. Decomposing graphs into long paths. *Order*, 20(3):239–253, September 2003.

[11] A. Kotzig. From the theory of finite regular graphs of degree three and four (in Slovak). *Časopis Pěstov. Mat.*, 82:76–92, 1957.

On the Complexity of Point Recolouring in Geometric Graphs

Henk Meijer
Roosevelt Academy
Middelburg, The Netherlands
h.meijer@roac.nl

Yurai Núñez-Rodríguez
Queen's University
Kingston, ON Canada
yurai@cs.queensu.ca

David Rappaport *
Queen's University
Kingston, ON Canada
daver@cs.queensu.ca

Abstract

Given a collection of points representing geographic data we consider the task of delineating boundaries based on the features of the points. Assuming that the features are binary, for example, red or blue, this can be viewed as determining red and blue regions, or states. Due to regional anomalies or sampling error, we may find that reclassifying, or recolouring, some points may lead to a more rational delineation of boundaries. In this note we study the maximal length of recolouring sequences where recolouring rules are based on neighbour relations and neighbours are defined by a geometric graph. We show that the difference in the maximal length of recolouring sequences is striking, as it can range from a linear bound for all trees, to an infinite sequence for some planar graphs.

1 Introduction

Given a set of planar points partitioned into red and blue subsets, a red-blue separator is a boundary that separates the red points from the blue ones. There has been considerable investigation of methods for obtaining such red-blue separating boundaries. In his PhD thesis, Seara [8], examines various means for red-blue separation. For the case of red-blue separation with the minimum perimeter polygon the problem is known to be NP-hard [3, 1]. A somewhat related topic is to obtain a balanced subdivision of red and blue points, that is, faces of the subdivision contain a prescribed ratio of red and blue points. Kaneko and Kano [4] give a comprehensive survey of results pertaining to red and blue points in the plane, including results on balanced subdivisions.

For some applications one is willing to reclassify points by recolouring them so as to obtain a more reasonable boundary. For example Chan [2] shows that finding a red-blue separating line with the minimum number of reclassified points takes $O((n+k^2)\log k)$ expected time, where k is the number of recoloured points.

In Reinbacher et al. [7] a heuristic algorithm is presented for obtaining a better delineating boundary that recolours points. The input is a triangulated set of n planar red-blue points. For a point p to be recoloured it needs to be "surrounded" by points of the opposite colour. Reinbacher et al. show experimental results on delineating boundaries after recolouring.

A *surrounded point* is realized when there is a contiguous set of oppositely coloured neighbours of p, in the triangulation, that span a radial angle greater than 180°. As the recolouring occurs in an iterative sequence it is not clear that the process will ever come to an end. However, Reinbacher et al. show that no sequence that iteratively recolours surrounded points will ever visit the exact same colouring of the points more than once. Thus the maximum number of recolourings is bounded by the total number of possible colourings which is $2^n - 1$. This bound was improved by Núñez-Rodríguez and Rappaport [5] by proving that any recolouring sequence has $O(n^2)$ length. This bound is, in fact, tight.

In this note, we present bounds on recolouring for other types of geometric graphs. Our main results include bounds on the number of recolourings for graphs of maximum degree 3, bounds on the number of recolourings of trees, and examples of infinite recolouring sequences on planar and non-planar graphs. Some of our proofs have been omitted because of space limitations.

In the next section we precisely describe the recolouring problem. We follow, in the subsequent section, with our new results on the length of recolouring sequences of geometric graphs, such as planar graphs, non-planar graphs, and trees. The last section discusses some extensions of our results.

2 Preliminaries

We are given as input a drawing of a graph, $D = (S, E)$, where S is a set of points in the plane partitioned into blue points and red points and $E \subseteq S \times S$ is the set of edges of the graph. The edges are represented by straight line segments. An edge incident to points p and q is denoted as $\overline{pq}$. We assume throughout, for simplicity of exposition, that the points are in general position. We colour the edges of D red if its two incident points are red, and blue if its two incident points are

*Supported by an NSERC of Canada Discovery Grant

blue. If one of the incident points is red and the other is blue we mix the colours to obtain a magenta edge.

Definition 1 *Let the edges of D be coloured as above. Then the* ***magenta angle*** *of a point $p \in S$ is:*

- $0°$*, if p has at most one radially consecutive incident magenta edge,*
- $360°$*, if p has degree greater than one and is only incident to magenta edges,*
- *the maximum angle between two or more radially consecutive incident magenta edges, otherwise.*

Notice that, according to the previous definition, a point with only one neighbour in D has magenta angle $0°$ regardless of the colour of its neighbour (See Figure 1). A *surrounded* point is one with magenta angle larger than $180°$. Therefore, a point of degree zero or one is never surrounded nor recoloured. We say an edge *is in* the magenta angle of a point if it is incident to the point and falls within the span of the magenta angle.

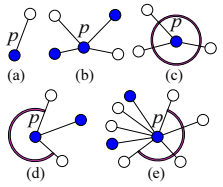


Figure 1: Examples of magenta angles α of point p. Magenta angles larger than $0°$ are represented by arcs: (a), (b) $\alpha = 0°$, (c) $\alpha = 360°$, (d), (e) $180° < \alpha < 360°$.

The strategy of reclassification by recolouring, recolours a surrounded point p at a time. The sequence in which surrounded points are recoloured can be driven by a mixed criterion, such as recolouring the surrounded point with the largest magenta angle, or with the largest number of edges in the magenta angle. According to Reinbacher et al. [7], there exist mixed criteria that always produce recolouring sequences of linear size. In the sequel, we assume surrounded points are recoloured in an arbitrary manner, in order to find bounds for any, and all, possible recolouring strategies. The recolouring process stops when there are no more surrounded points.

3 Bounded and Unbounded Recolouring Sequences

At all times the graphs are assumed to be connected because, in general, each connected component can be considered independently. We use the term *drawing*, to refer to the drawing of a graph as defined in the previous section. There are families of drawings for which every recolouring sequence is finite and others that allow infinite recolouring sequences. We characterize some of these families. A family of drawings with finite recolouring sequence comes from graphs with maximum degree 3.

Theorem 1 *Let D be a drawing with n bi-chromatic points. If D has maximum point degree 3, then the length of any recolouring sequence of D is $O(n)$.*

The proof of Theorem 1 only relies on the number of point neighbours and the fact that the number of magenta edges decreases with every recolouring. Thus, the result also holds for non-planar and more general drawings with maximum degree 3.

3.1 Planar Drawings

As opposed to triangulations, planar drawings may have non-convex faces and points of degree one. One may think of obtaining bounds for planar graphs based on the fact that a planar graph is a subgraph of a triangulation. However, this does not seem to help since there are simple examples where a subgraph can have either a larger or a smaller number of recolourings in the worst case over all initial colourings.

In fact, a recolouring sequence of a planar graph can be infinite. Figure 2 shows an example of a graph and a colour configuration that lead to an infinite recolouring sequence. Observe from Figure 2 that the initial colouring repeats after a number of steps (recolourings). Notice that only certain recolouring sequences are infinite in this example. The drawing in Figure 2 can be made 2-connected and the minimum point degree can be increased by carefully adding more edges incidents to the points of degree one, without affecting the recolouring sequence.

3.2 Non-Planar Drawings

At this point, it is obvious that one can also construct non-planar drawings with infinite recolouring sequences since planar drawings allow so. Nevertheless, the examples shown for infinite recolouring sequences on planar drawings include points that never change colour. We show an example of a non-planar drawing with infinite recolouring sequence where every point changes colour infinitely many times (see Figure 3). If similar examples can be built for planar drawings, these have not yet been found.

3.3 Trees

In this subsection we use the term *tree drawing* to refer to a straight line drawing of a tree (not necessarily planar). A trivial example of a tree drawing that has $O(n)$ recolouring sequence is a "jigsaw" path with points alternately coloured. In such example, all blue points

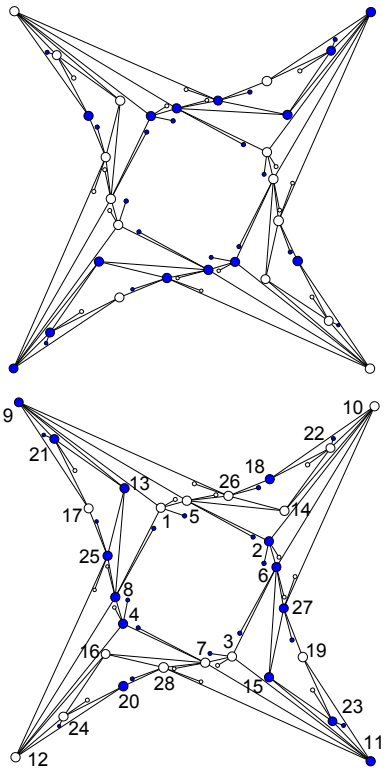


Figure 2: Planar drawing D with infinite recolouring sequence. Points represented by smaller circles never change colour. Top: initial colouring of D. Bottom: D after 28 recolourings. The labels indicate the order in which points are recoloured. Notice that the bottom drawing is a rotation of the top drawing. This indicates that the recolourings can repeat infinitely many times.

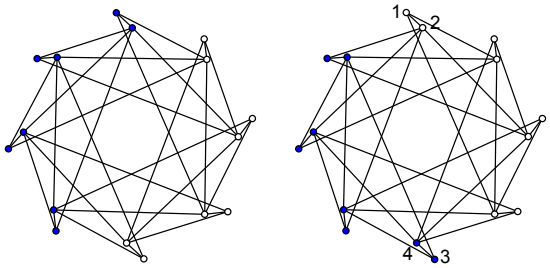


Figure 3: Non-planar drawing D with infinite recolouring sequence. Left: initial colouring of D. Right: D after 4 recolourings. The labels indicate the order in which points are recoloured. Notice that the right drawing is a rotation of the left drawing. This indicates that the recolourings can repeat infinitely many times.

can be coloured to red, leading to approximately $n/2$ recolourings.

It is not hard to prove that the number of recolourings of tree drawings is $O(n^2)$ by the same arguments used for triangulations (Theorem 9 [5]), with minor modifications. However, this bound is not tight. As a corollary of Theorem 1 we have that binary tree drawings have a linear number of recolourings. In the remainder of this section we prove that the number of recolourings of general tree drawings is also linear.

In order to prove a tight bound (Theorem 4) for the recolouring of tree drawings, we define a partial order on the recolourings involved in a recolouring sequence. Then we bound the total number of recolourings based on the number of minimal elements (sinks) of such partial order. This idea is explained and formalized in what follows.

We denote a recolouring event r, or simply a recolouring, as the event of a certain point p changing colour. Let $R = (r_1, \ldots, r_k)$ be a recolouring sequence where r_i denotes the recolouring at step i, $1 \leq i \leq k$, $k > 0$. We also denote $p(r)$ as the point that changes colour at recolouring r, and $N(r)$ the number of times that $p(r)$ has changed colour in R prior to event r.

Definition 2 *Let T be a tree drawing and let R be a recolouring sequence of T. The **history graph** of R is a directed graph $H = (R, I)$, $I \subseteq R \times R$ such that $(r_j, r_i) \in I$ if and only if $p(r_i)p(r_j)$ is in the magenta angle associated to r_j.*

Observation 1 *By the definition of history graph all the edges are directed from later recolourings to earlier ones. Therefore, a history graph is a directed acyclic graph (DAG) and defines a partial order on the elements of the recolouring sequence.*

The following lemma formally states that for two consecutive recolourings of a point p to occur, it is required that at least two neighbours of p change colour in between.

Lemma 2 *Let T be a tree drawing with n bi-chromatic points, let R be a recolouring sequence of T, and let $H = (R, I)$ be the history graph of R. Consider a recolouring $r \in R$ with $N(r) > 0$. Then the outdegree of r is at least 2. Moreover, there exist two distinct neighbours of $p(r)$, $p_1, p_2 \in T$, with recolourings $s_1, s_2 \in R$, respectively, such that $(r, s_1) \in I$ and $(r, s_2) \in I$.*

Proof. Obviously, if a point is recoloured red (similarly blue) and was recoloured earlier in the sequence, the previous recolouring was to blue (red). The intersection between the magenta angles at the time it is surrounded by red (blue) and previously by blue (red) contains at least two edges since the corresponding magenta angles are greater than 180°. Therefore, there are at least two neighbours of p, $p_1, p_2 \in T$ that are recoloured at least once between two consecutive recolourings of p. □

In the light of Lemma 2, we can state the following definition.

Definition 3 *Let R be a recolouring sequence of a tree drawing T and $H = (R, I)$ be the corresponding history graph. The* ***binary history graph*** *of R, $BH = (R, BI)$, $BI \subseteq I$, is a subgraph of the history graph where nodes have outdegrees 2 or 0: nodes with outdegree 0 correspond to first time recolourings; nodes with outdegree 2 correspond to subsequent recolourings. Consider a node r_k of degree 2 in the binary history tree. From Lemma 2 we know that there are two distinct neighbours of $p(r_k)$ that have been previously recoloured. Thus we choose the two outgoing edges of r_k $\overline{(r_k, r_i)}$, $\overline{(r_k, r_j)}$, such that i and j are the largest indices smaller than k for neighbours of r_k in the history graph where $p(r_i) \neq p(r_j)$.*

The motivation to define the binary history graph is to obtain a cycle-free subgraph of the history graph that involves all the recolourings. This is formalized in the next lemma.

Lemma 3 *Let T be a tree drawing, and R a recolouring sequence of T with binary history graph BH. BH has no directed or undirected cycles. Therefore, BH is a forest of trees.*

To obtain a bound on the size of binary history trees we show that the number of nodes is linear in the size of the corresponding tree drawing. This will lead us to conclude the results of the following theorem.

Theorem 4 *Let T be a tree drawing with n bichromatic points. The length of any recolouring sequence of T is $O(n)$.*

4 Extensions

4.1 Surrounded Threshold Greater Than 180°

Thus far we have assumed that a point is surrounded when its magenta angle is any value greater than 180°. Reinbacher et al. [7] show that a threshold value smaller than 180° allows for infinite recolouring sequences on very simple graphs –trees included. Some of our results hold for threshold values $\alpha > 180°$. Trees, for example, have linear recolouring sequences for any threshold $180° < \alpha < 360°$. Also Theorem 1 holds for any threshold $\alpha > 0°$. Other results do not seem to hold for any threshold value. For instance, it is not clear how large the value of α can be such that infinite recolouring sequences exist on planar graphs.

4.2 More than two colours

Suppose that the points come in more than two colours. We define the colour of an edge as the mixture of the colours of its endpoints. In a multi-coloured scenario we say that p is *surrounded* by a set of edges of a single mixed colour if the edges define a continuous angle greater than 180°. As we may intuitively observe, increasing the number of colours only lowers the chances of a point being surrounded without changing the fundamental nature of the problem. In fact, inspection shows that all of our previous results hold in a multi-coloured scenario. Thus, our recolouring bounds for a bi-chromatic set of points carry over to multi-coloured point sets.

5 Conclusions

We have re-examined a point recolouring method useful for reclassifying points to obtain reasonable subdividing boundaries. We show tight (linear) bounds on trees and graphs of maximum degree 3 for the longest possible sequence of recolourings. Planar and non-planar graphs have been shown to have infinitely many recolourings.

Some interesting questions remain open. First, can planar drawings have sequences of recolourings where all the points change colour infinitely many times? Also, what is the complexity of point recolouring in planar drawings and other geometric graphs when the threshold to consider a point as surrounded is greater than 180°?

References

[1] E. M. Arkin, S. Khuller, and J. S. B. Mitchell. Geometric knapsack problems. *Algorithmica*, 10:399–427, 1993.

[2] T. M. Chan. Low-dimensional linear programming with violations. in *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, 570–579, 2002.

[3] P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recogn. Lett.*, 14:715–718, 1993.

[4] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane —a survey. in *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*, Algorithms and Combinatorics, Springer, vol. 25, 551–570, 2003.

[5] Y. Núñez-Rodríguez, and D. Rappaport. Tight bounds for point recolouring. *Proc. 18th Canadian Conf. in Comp. Geometry (CCCG'06)*, Kingston, ON, Canada, 67–70, 2006.

[6] Joseph O'Rourke. Computational Geometry in C. *Cambridge University Press*, 1998.

[7] Iris Reinbacher, Marc Benkert, Marc van Kreveld, Joseph S.B. Mitchell, Jack Snoeyink, and Alexander Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, vol. 50:3, 386–414, 2008.

[8] Carlos Seara. On Geometric Separability. *PhD. Thesis. Universitat Politècnica De Catalunya*, 2002.

Improved Bounds on the Average Distance to the Fermat-Weber Center of a Convex Object

A. Karim Abu-Affash* Matthew J. Katz*

Abstract

We show that for any convex object Q in the plane, the average distance between the Fermat-Weber center of Q and the points in Q is at least $4\Delta(Q)/25$, and at most $2\Delta(Q)/(3\sqrt{3})$, where $\Delta(Q)$ is the diameter of Q. We use the former bound to improve the approximation ratio of a load-balancing algorithm of Aronov et al. [1].

1 Introduction

The Fermat-Weber center of an object Q in the plane is a point in the plane, such that the average distance from it to the points in Q is minimal. For an object Q and a point y, let $\mu_Q(y)$ be the average distance between y and the points in Q, that is, $\mu_Q(y) = \int_{x \in Q} \|xy\|\, dx/\text{area}(Q)$, where $\|xy\|$ is the Euclidean distance between x and y. Let $\mathcal{FW}_Q$ be a point for which this average distance is minimal, that is, $\mu_Q(\mathcal{FW}_Q) = \min_y \mu_Q(y)$, and put $\mu^*_Q = \mu_Q(\mathcal{FW}_Q)$. The point $\mathcal{FW}_Q$ is a Fermat-Weber center of Q.

It is easy to verify, for example, that the Fermat-Weber center of a disk D coincides with the center o of D, and that the average distance between o and the points in D is $\Delta(D)/3$, where $\Delta(D)$ is the diameter of D. Carmi, Har-Peled, and Katz [3] studied the relation between μ^*_Q and the diameter of Q, denoted $\Delta(Q)$. They proved that there exists a constant c_1, such that, for any *convex* object Q, the average distance between a Fermat-Weber center of Q and the points in Q is at least $c_1\Delta(Q)$, and that the largest such constant c^*_1 lies in the range $[1/7..1/6]$.

In this paper, we both improve the above bound on c^*_1, and tightly bound a new constant c^*_2; see below. More precisely, we first significantly narrow the range in which c^*_1 must lie, by proving (in Section 2) that $4/25 \leq c^*_1 \leq 1/6$. Next, we consider the question what is the smallest constant c^*_2, such that, for any convex object Q, $\mu^*_Q \leq c^*_2\Delta(Q)$. We prove (in Section 3) that $1/3 \leq c^*_2 \leq 2/(3\sqrt{3})$. A useful corollary obtained from these results is that the average distance to the center of the smallest enclosing circle of a convex n-gon P is less than 2.41 times μ^*_P.

The Fermat-Weber center of an object Q is a very significant point. The classical Fermat-Weber problem is: Find a point in a set F of feasible facility locations, that minimizes the average distance to the points in a set D of (possibly weighted) demand locations. If D is a finite set of points, F is the entire plane, and distances are measured using the L_2 metric, then it is known that the solution is algebraic [2]. See Wesolowsky [8] for a survey of the Fermat-Weber problem.

Only a few papers deal with the continuous version of the Fermat-Weber problem, where the set of demand locations is continuous. Fekete, Mitchell and Weinbrecht [4] presented algorithms for computing an optimal solution for $D = F = P$ where P is a simple polygon or a polygon with holes, and the distance between two points in P is the L_1 geodesic distance between them. Carmi, Har-Peled and Katz [3] presented a linear-time approximation scheme for the case where P is a convex polygon.

Aronov et al. [1] considered the following load balancing problem. Let D be a convex region and let $\mathcal{P} = \{p_1, \ldots, p_m\}$ be a set of m points representing m facilities. One would like to divide D into m equal-area subregions $R_1, \ldots, R_m$, so that region R_i is associated with point p_i, and the total cost of the subdivision is minimized. Given a subdivision, the cost $\kappa(p_i)$ associated with facility p_i is the average distance between p_i and the points in R_i, and the total cost of the subdivision is $\sum_i \kappa(p_i)$.

Aronov, et al. discussed the structure of an optimal subdivision, and also presented an $(8 + \sqrt{2\pi})$-approximation algorithm, under the assumption that the regions $R_1, \ldots, R_m$ must be convex and that D is a rectangle. Our improved bound on the constant c^*_1, allows us (in Section 4) to improve the above approximation ratio.

2 $4/25 \leq c^*_1 \leq 1/6$

Carmi, Har-Peled and Katz [3] showed that there exists a convex polygon P such that $\mu^*_P \leq \Delta(P)/6$. This immediately implies that $c^*_1 \leq 1/6$. We prove below that $c^*_1 \geq 4/25$. Our proof is similar in its structure to the proof of [3].

*Department of Computer Science, Ben-Gurion University, Israel; {abuaffas, matya}@cs.bgu.ac.il. A.K. Abu-Affash was partially supported by the Lynn and William Frankel Center for Computer Sciences.

Theorem 2.1. *Let P be a convex object. Then $\mu_P^* \geq 4\Delta(P)/25$.*

Proof: Let $\mathcal{FW}_P$ be a Fermat-Weber center of P. We need to show that $\int_{x\in P} \|x\mathcal{FW}_P\|\,dx \geq \frac{4\Delta(P)}{25}\text{area}(P)$. We do this in two stages. In the first stage we show that for a certain subset P' of P, $\int_{x\in P'} \|x\mathcal{FW}_P\|\,dx \geq \frac{4\Delta(P)}{27}\text{area}(P)$. This implies that for any convex object Q, $\mu_Q^* \geq 4\Delta(Q)/27$. In the second stage we apply this intermediate result to a collection of convex subsets of $P - P'$ that are pairwise disjoint to obtain the claimed result. This latter stage is essentially identical to the second stage in the proof of [3]; it is included here for the reader's convenience.

We now describe the first stage. Let s be a line segment of length $\Delta(P)$ connecting two points p and q on the boundary of P. We may assume that s is horizontal and that p is its left endpoint, since one can always rotate P around, say, p until this is the case.

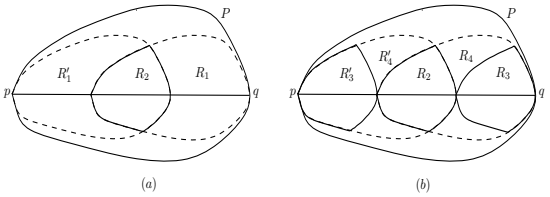


Figure 1: Proof of intermediate result.

Let P^α be the polygon obtained from P by shrinking it by a factor of α, that is, by applying the transformation $f(a,b) = (a/\alpha, b/\alpha)$ to the points (a,b) in P. We place a copy R_1 of $P^{3/2}$, such that R_1 is contained in P and has a common tangent with P at q. Similarly, we place a copy R_1' of $P^{3/2}$, such that R_1' is contained in P and has a common tangent with P at p; see Figure 1(a). Clearly, $\text{area}(R_1) = \text{area}(R_1') = \frac{4}{9}\text{area}(P)$.

Let $R_2 = R_1 \cap R_1'$. We place a copy R_3 of R_2, such that R_3 is contained in R_1 and has a common tangent with R_1 at q. Similarly, we place a copy R_3' of R_2, such that R_3' is contained in R_1' and has a common tangent with R_1' at p. Let $R_4 = R_1 - (R_2 \cup R_3)$ and $R_4' = R_1' - (R_2 \cup R_3')$; see Figure 1(b).

We know that, regardless of the exact location of $\mathcal{FW}_P$, the distance between $\mathcal{FW}_P$ and the points in R_3 plus the distance between $\mathcal{FW}_P$ and the points in R_3' is greater than $\frac{2\Delta(P)}{3}\text{area}(R_3)$, and the distance between $\mathcal{FW}_P$ and the points in R_4 plus the distance between $\mathcal{FW}_P$ and the points in R_4' is greater than $\frac{\Delta(P)}{3}\text{area}(R_4)$. More precisely,

$$\int_{x\in R_3} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R_3'} \|x\mathcal{FW}_P\|\,dx \geq \frac{2\Delta(P)}{3}\text{area}(R_3)$$

and

$$\int_{x\in R_4} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R_4'} \|x\mathcal{FW}_P\|\,dx \geq \frac{\Delta(P)}{3}\text{area}(R_4)\,.$$

Since $\text{area}(R_4) = \text{area}(R_1) - (\text{area}(R_2) \cup \text{area}(R_3)) = \frac{4}{9}\text{area}(P) - 2\text{area}(R_3)$, we obtain our intermediate result

$$\begin{aligned}\int_{x\in P} \|x\mathcal{FW}_P\|\,dx &\geq \int_{x\in R_3} \|x\mathcal{FW}_P\|\,dx + \\ &+ \int_{x\in R_3'} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R_4} \|x\mathcal{FW}_P\|\,dx + \\ &+ \int_{x\in R_4'} \|x\mathcal{FW}_P\|\,dx \geq \frac{2\Delta(P)}{3}\text{area}(R_3) + \\ &+ \frac{\Delta(P)}{3}\left(\frac{4}{9}\text{area}(P) - 2\text{area}(R_3)\right) = \frac{4\Delta(P)}{27}\text{area}(P)\,.\end{aligned}$$

This intermediate result immediately implies that for any convex object Q, $\mu_Q^* \geq 4\Delta(Q)/27$. In the second stage we show that the 27 in the denominator can be replaced by 25.

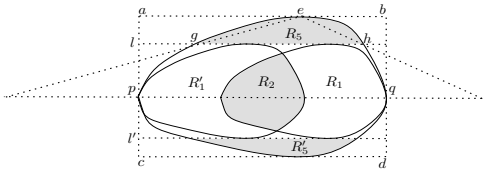


Figure 2: Proof of improved result.

Consider Figure 2. We draw the axis-aligned bounding box of P. The line segment s (whose length is $\Delta(P)$) divides the bounding box of P into two rectangles, $abqp$ above s and $pqdc$ below s. We divide each of these rectangles into two parts (a lower part and an upper part), by drawing the two horizontal lines l and l'. Let R_5 denote the intersection of P with the upper part of the upper rectangle, and let R_5' denote the intersection of P with the lower part of the lower rectangle.

Let e be any point on the segment ab that also lies on the boundary of R_5. We mention several facts concerning R_5 and R_5'. $R_5 \cap R_5' = \phi$, $R_5 \cap R_1 = \phi$, $R_5 \cap R_1' = \phi$, $R_5' \cap R_1 = \phi$, and $R_5' \cap R_1' = \phi$. Notice also that $\Delta(R_5)$, $\Delta(R_5') \geq \Delta(P)/3$, since, e.g., the line segment $l \cap R_5$ contains the base of the triangle that is obtained by intersecting the triangle peq with R_5, and the length of this base is $\Delta(P)/3$.

We observe that $\text{area}(R_5) + \text{area}(R_5') \geq \text{area}(P)/9$ by showing that $\text{area}(R_5) \geq \text{area}(P \cap abqp)/9$ (and that $\text{area}(R_5') \geq \text{area}(P \cap pqdc)/9$). Let g, h be the two points on the line l that also lie on the boundary of R_5. Let $l(s)$ be the line containing s, and let T be the triangle defined by $l(s)$ and the two line segments connecting e to $l(s)$ and passing through g and through h, respectively. Let T_2 denote the triangle geh.

Clearly $T_2 \subseteq R_5$. Put $Q = R_5 - T_2$. Then, $\text{area}(R_5) = \text{area}(T_2) + \text{area}(Q) = \text{area}(T)/9 + \text{area}(Q)$. Therefore, $\text{area}(R_5) \geq (\text{area}(T) + \text{area}(Q))/9 \geq \text{area}(P \cap abqp)/9$. We show that $\text{area}(R'_5) \geq \text{area}(P \cap pqdc)/9$ using the "symmetric" construction. Since $(P \cap abqp) \cup (P \cap pqdc) = P$ we obtain that $\text{area}(R_5) + \text{area}(R'_5) \geq \text{area}(P)/9$.

It is also easy to see that $\Delta(R_2) = \Delta(P)/3$ and $\text{area}(R_2) \geq \text{area}(P)/9$. This is because $P^3 \subseteq R_2$ and $\text{area}(P^3) = \text{area}(P)/9$, where P^3 is the polygon obtained from P by shrinking it by a factor of 3.

Now using the implication of our intermediate result we have

$$\begin{aligned}\int_{x\in R_5} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R'_5} \|x\mathcal{FW}_P\|\,dx + \\ + \int_{x\in R_2} \|x\mathcal{FW}_P\|\,dx \geq \frac{4\Delta(R_5)}{27}\text{area}(R_5) + \\ + \frac{4\Delta(R'_5)}{27}\text{area}(R'_5) + \frac{4\Delta(R_2)}{27}\text{area}(R_2) \geq \\ \geq \frac{4\Delta(P)}{81}(\text{area}(R_5) + \text{area}(R'_5) + \text{area}(R_2)) \geq \\ \geq \frac{8\Delta(P)}{729}\text{area}(P)\,.\end{aligned}$$

Therefore

$$\begin{aligned}\int_{x\in P} \|x\mathcal{FW}_P\|\,dx \geq \int_{x\in R_3} \|x\mathcal{FW}_P\|\,dx + \\ + \int_{x\in R'_3} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R_4} \|x\mathcal{FW}_P\|\,dx + \\ + \int_{x\in R'_4} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R_5} \|x\mathcal{FW}_P\|\,dx + \\ + \int_{x\in R'_5} \|x\mathcal{FW}_P\|\,dx + \int_{x\in R_2} \|x\mathcal{FW}_P\|\,dx \geq \\ \geq \frac{4\Delta(P)}{27}\text{area}(P) + \frac{8\Delta(P)}{729}\text{area}(P) = \\ = \frac{116\Delta(P)}{729}\text{area}(P)\,.\end{aligned}$$

At this point we may conclude that for any convex object Q, $\mu^*_Q \geq 116\Delta(Q)/729$. So we repeat the calculation above using this result for the regions R_5, R'_5 and R_2 (instead of using the slightly weaker result, i.e., $\mu^*_Q \geq 4\Delta(Q)/27$). This calculation will yield a slightly stronger result, etc. In general, the result after the k-th iteration is $\mu^*_Q \geq c_k\Delta(Q)$, where $c_k = 4/27 + 2c_{k-1}/27$ and $c_0 = 4/27$. It is easy to verify that this sequence of results converges to $\mu^*_Q \geq 4\Delta(Q)/25$. ∎

Corollary 2.2. *Let P be a non-convex simple polygon, such that the ratio between the area of a minimum-area enclosing ellipse of P and the area of a maximum-area enclosed ellipse is at most β, for some constant $\beta \geq 1$. Then $\mu^*_P \geq 4\Delta(P)/(25\beta^2)$.*

Proof: As in [3], except that we apply the improved bound of Theorem 2.1. ∎

3 $1/3 \leq c^*_2 \leq 2/(3\sqrt{3})$

As mentioned in the introduction, the average distance between the Fermat-Weber center of a disk D (i.e., D's center) and the points in D is $\Delta(D)/3$, where $\Delta(D)$ is the diameter of D. This immediately implies that $c^*_2 \geq 1/3$. We prove below that $c^*_2 \leq 2/(3\sqrt{3})$.

We first state a simple lemma and a theorem of Jung that are needed for our proof.

Lemma 3.1. *Let R, Q be two (not-necessarily convex) disjoint objects, and let p be a point in the plane. Then, $\mu_{(R\cup Q)}(p) \leq \max\{\mu_R(p), \mu_Q(p)\}$.*

Proof:

$$\begin{aligned}\mu_{(R\cup Q)}(p) = \frac{\int_{x\in R\cup Q}\|px\|\,dx}{\text{area}(R\cup Q)} = \\ = \frac{\int_{x\in R}\|px\|\,dx + \int_{x\in Q}\|px\|\,dx}{\text{area}(R) + \text{area}(Q)} = \\ = \frac{\text{area}(R)\cdot\mu_R(p) + \text{area}(Q)\cdot\mu_Q(p)}{\text{area}(R) + \text{area}(Q)} \leq \\ \leq \frac{(\text{area}(R) + \text{area}(Q))\max\{\mu_R(p), \mu_Q(p)\}}{\text{area}(R) + \text{area}(Q)} \leq \\ \leq \max\{\mu_R(p), \mu_Q(p)\}\,.\end{aligned}$$ ∎

Theorem 3.2 (Jung's Theorem [5, 6]). *Every set of diameter d in $\mathbb{R}^n$ is contained in a closed ball of radius $r \leq d\sqrt{\frac{n}{2(n+1)}}$. In particular, if R is a convex object in the plane, then the radius of the smallest enclosing circle C of R is at most $\Delta(R)/\sqrt{3}$, where $\Delta(R)$ is the diameter of R.*

Theorem 3.3. *For any convex object R, $\mu^*_R \leq 2\Delta(R)/(3\sqrt{3})$.*

Proof: Let R be a convex polygon. Let C be the smallest enclosing circle of R, and let o and r denote R's center point and radius, respectively. Notice that $o \in R$, since R is convex. We divide R into 8 regions $R_1, \ldots, R_8$ by drawing four line segments through o, such that each of the 8 angles formed around o is of 45°; see Figure 3(a). Clearly, for each R_i, $o \in R_i$ and $\Delta(R_i) \leq r$.

We first prove that for each region R_i, $\mu_{R_i}(o) \leq 2\Delta(R_i)/3$. (This is done by adapting the proof of Lemma 3.1 of Aronov et al. [1].) Consider Figure 3(b). Let $p \in R_i$ be the farthest point from o. Draw the circular sector ocd centered at o of radius $\|op\|$. Let a and b be as in Figure 3(b). Let f be the point on the arc cd, such that the regions Q_1 and Q_2 obtained by drawing the segment of are of equal area. (Q_1 is the region oxb and Q_2 is the difference between the sector opf and the

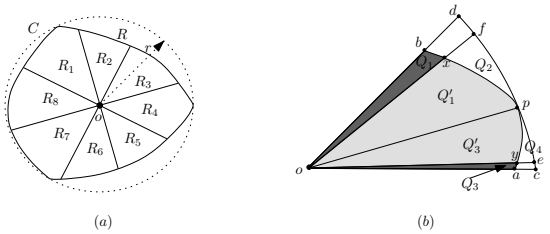


Figure 3: Illustrating the proof of Theorem 3.3.

region opx, where x is the intersection point between of and the boundary piece pb.) Similarly, let e be the point on the arc cd, such that the regions Q_3 and Q_4 obtained by drawing the segment oe are of equal area. (Q_3 is the region oay and Q_4 is the difference between the sector oep and the region oyp, where y is the intersection point between oe and the boundary piece ap.)

Now, on the one hand, since opb is convex, x is the farthest point from o in Q_1, and, on the other hand, x is the closest point to o in Q_2. Hence, any point in Q_2 is farther from o than any point in Q_1. Thus we get that $\mu_{opb}(o) = \mu_{(Q'_1 \cup Q_1)}(o) \leq \mu_{(Q'_1 \cup Q_2)}(o) = \mu_{opf}(o) = 2\,\|op\|/3 = 2\Delta R_i/3$. We show that $\mu_{oap}(o) \leq 2\Delta R_i/3$ using the "symmetric" analysis. Since opb and oap are disjoint convex objects, then, by Lemma 3.1, $\mu_{R_i}(o) = \mu_{(opb \cup oap)}(o) \leq 2\Delta R_i/3$.

We now show that $\mu_R(o) \leq 2\Delta(R)/(3\sqrt{3})$, immediately implying that $\mu^*_R \leq 2\Delta(R)/(3\sqrt{3})$. By Theorem 3.2, we know that $r \leq \Delta(R)/\sqrt{3}$. We also know that for each R_i, $\Delta(R_i) \leq r$. Thus, $\mu_{R_i}(o) \leq 2\Delta(R_i)/3 \leq 2r/3 \leq 2\Delta(R)/(3\sqrt{3})$.

We now apply Lemma 3.1 to obtain that

$$\begin{aligned}
\mu_R(o) &\leq \max\left\{\mu_{(R_1 \cup R_2 \cup R_3 \cup R_4)}(o), \mu_{(R_5 \cup R_6 \cup R_7 \cup R_8)}(o)\right\} \leq \\
&\leq \max\Big\{\max\left\{\mu_{(R_1 \cup R_2)}(o), \mu_{(R_3 \cup R_4)}(o)\right\}, \\
&\qquad \max\left\{\mu_{(R_5 \cup R_6)}(o), \mu_{(R_7 \cup R_8)}(o)\right\}\Big\} \leq \\
&\quad \vdots \\
&\leq \max\{\mu_{R_1}(o), \mu_{R_2}(o), \mu_{R_3}(o), \mu_{R_4}(o), \\
&\qquad \mu_{R_5}(o), \mu_{R_6}(o), \mu_{R_7}(o), \mu_{R_8}(o)\} \leq \\
&\leq 2\Delta(R)/(3\sqrt{3})\,.
\end{aligned}$$

■

Corollary 3.4. *Let P be a convex n-gon. Then one can compute in linear time a point p, such that $\mu_P(p) \leq \frac{25}{6\sqrt{3}}\mu^*_P$.*

Proof: We apply Megiddo's linear-time algorithm for computing the smallest enclosing circle C of P [7]. Let p denote the center of C, then, by Theorem 2.1

$$\frac{\mu_P(p)}{\mu^*_P} \leq \frac{2\Delta(P)/(3\sqrt{3})}{4\Delta(P)/25} = \frac{25}{6\sqrt{3}}\,.$$

■

Corollary 3.4 gives us a very simple linear-time constant-factor approximation algorithm for finding an approximate Fermat-Weber center in a convex polygon. A less practical linear approximation scheme for finding such a point was presented by Carmi et al. [3].

4 Application

We consider the load balancing problem studied by Aronov et al. [1]. Let D be a convex region and let $\mathcal{P} = \{p_1, \ldots, p_m\}$ be a set of m points representing m facilities. The goal is to divide D into m equal-area *convex* regions $R_1, \ldots, R_m$, so that region R_i is associated with point p_i, and the total cost of the subdivision is minimized. The cost $\kappa(p_i)$ associated with facility p_i is the average distance between p_i and the points in R_i, and the total cost of the subdivision is $\sum_i \kappa(p_i)$.

Assuming D is a rectangle that can be divided into m squares of equal size, Aronov et al. present an $O(m^3)$-time algorithm for computing a subdivision of cost at most $(8+\sqrt{2\pi})$ times the cost of an optimal subdivision. By applying Theorem 2.1 in the analysis of their algorithm, we obtain a better approximation ratio, namely, $(\frac{29}{4} + \sqrt{2\pi})$. For further details, see the full version of this paper.

References

[1] B. Aronov, P. Carmi, and M.J. Katz. Minimum-cost load-balancing partitions. *Proc. 22nd ACM Symp. on Computational Geometry,* pages 301–308, 2006.

[2] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete and Comput. Geom.,* 3 (1988), 177–191.

[3] P. Carmi, S. Har-Peled, and M.J. Katz. On the Fermat-Weber center of a convex object. *Comput. Geom. Theory and Appls,* 32(3) (2005), 188–195.

[4] S.P. Fekete, J.S.B. Mitchell, and K. Weinbrecht. On the continuous Weber and k-median problems. *Proc. 16th ACM Symp. on Computational Geometry,* pages 70–79, 2000.

[5] H.W.E. Jung. Über die kleinste Kugel, die eine räumliche Figur einschließt. *J. Angew. Math.* 123 (1901), 241–257.

[6] H.W.E. Jung. Über den kleinsten Kreis, der eine ebene Figur einschließt. *J. Angew. Math.* 137 (1910), 310–313.

[7] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing,* 12(4) (1983), 759–776.

[8] G. Wesolowsky. The Weber problem: History and perspectives. *Location Science,* 1(1) (1993), 5–23.

On the nonexistence of dimension reduction for ℓ_2^2 metrics

Avner Magen* Mohammad Moharrami *

Abstract

An ℓ_2^2 metric is a metric ρ such that $\sqrt{\rho}$ can be embedded isometrically into $\mathbb{R}^d$ endowed with Euclidean norm, and the minimal possible d is the dimension associated with ρ. A dimension reduction of an ℓ_2^2 metric ρ is an embedding of ρ into another ℓ_2^2 metric μ so that distances in μ are similar to those in ρ and moreover, the dimension associated with μ is small. Much of the motivation in investigating dimension reductions in ℓ_2^2 comes from a result of Goemans which shows that if such metrics have good dimension reductions, then they embed well into ℓ_1 spaces. This in turn yields a rounding procedure to a host of semidefinite programming with good approximation guarantees.

In this work we show that there is no dimension reduction ℓ_2^2 metrics in the following strong sense: for every function $D(n)$ and for every n there exists an n point ℓ_2^2 metric ρ such that for all embeddings of ρ into an ℓ_2^2 metric μ with distortion at most $D(n)$, the associated dimension of μ is at least $n-1$. This stands in striking contrast to the Johnson Lindenstrauss lemma which provides a logarithmic dimension reduction for ℓ_2 metrics.

1 Introduction

The theory of finite metric spaces has attracted a lot of attention from algorithm designers in recent years. In fact, many substantial steps in approximation algorithms were achieved using *embeddings* of one metric space into another and estimating the *distortion* of the embedding.

We quickly review the needed background. Let $f : (X,d) \to (X',d')$ be a mapping from metric space (X,d) into metric space (X',d'). The distortion of f is the minimum D such that $\alpha \cdot d(x,y) \le d'(f(x),f(y)) \le D \cdot \alpha d(x,y)$ holds for some $\alpha \ge 0$ and for any $x,y \in X$.

One of the most useful ways embedding results are applied is in the context of Linear Program and Semidefinite Programming relaxations for combinatorial problems. By viewing optimal solutions of such relaxations as finite metric spaces and then embedding these metric spaces with low distortion into ℓ_1 one effectively obtains a rounding procedure (see [12, 7, 14]), namely, a procedure that maps a set of vectors into a $\{0,1\}$ assignment. The groundbreaking work of Arora, Rao and Vazirani [4] used this idea to provide an improved approximation of $O(\sqrt{\log n})$ to Sparsest Cut.

Metric spaces emerging from semidefinite relaxations can be typically described as follows. Consider a finite set of point in $\mathbb{R}^p$ endowed with the *square* Euclidean distance, that is, for vectors $\mathbf{v}_1,\ldots,\mathbf{v}_n \in \mathbb{R}^p$ the resulting distance function is $d_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|^2$. A distance function d obtained this is called an ℓ_2^2 distance functions. If in addition d satisfies triangle inequalities, we say that d is an ℓ_2^2 *metric*, or a *Negative Type Metric*. Notice that semidefinite relaxations may enforce such (linear) constraints.

Unlike ℓ_p metrics, i.e., metrics that embed in ℓ_p space with no distortion, the class of ℓ_2^2 metrics does not inherit the structure of a host normed space. This, to a great extent, explains why analyzing such metrics proved to be notoriously hard. The aforementioned result of Arora et al. [4], while not directly about metric spaces, shows that ℓ_2^2 metrics are well embeddable into ℓ_1 and ℓ_2 in some appropriately defined average sense. The result was later extended to show that every ℓ_2^2 metric is embeddable into ℓ_1 with distortion $O(\sqrt{\log n}\cdot\log\log n)$ [3]. Finding the smallest distortion needed to embed such metrics in ℓ_1 has become an intriguing open question, attracting attention from both geometers as well as complexity theorists. The best lower bound known so far is due to Khot and Vishnoi [9] that show that $\Omega(\log\log n)$ distortion is required.

Another theme of interest in the theory of metric spaces is dimension reduction: to what extent can the dimension associated with a metric be reduced without changing the distances by much? Such reductions are well understood in Euclidean space. While representing the metric of n points in Euclidean space isometrically requires dimension $n-1$, much less is sufficient for near isometries. In a seminal paper [8], Johnson and Lindenstrauss show that every n-point set in ℓ_2 can be embedded into $O(\log n)$-dimensional Euclidean space with constant distortion. Alon [2] recently showed that this dimension is essentially the best possible. Exploring the possibility of a similar phenomenon in ℓ_1 spaces, Brinkman and Charikar [5] and later Lee and Naor [11] showed that there is no dimension reduction in these spaces: they exhibited an n point metric in ℓ_1 such that embedding it with constant distortion in m-dimensional ℓ_1 space is only possible for $m = n^{\Omega(1)}$.

The following result due to M. Goemans presented

*Department of Computer Science, University of Toronto

in a workshop on methods in discrete mathematics [6] relates the themes distortion and dimension, with respect to ℓ_2^2 metrics. As such, it suggested an alternative avenue to low-distortion embeddings for such metrics. First, we need a slight restriction in the definition of the metrics spaces in question.

Definition 1 *A distance function on n points is called NEG_{sym} if there are vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathbb{R}^p$ such that $d_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|^2$, and $\|x-y\|^2 + \|z-y\|^2 \geq \|x-z\|^2$ for all $x, y, z \in \{\pm \mathbf{v}_i\}_i$. The smallest possible p above is the dimension associated with d.*

Notice that $\mathrm{NEG_{sym}}$ metrics are special cases of ℓ_2^2 metrics. We also observe that since $\|x-y\|^2 + \|z-y\|^2 - \|x-z\|^2 = 2(x-y)\cdot(z-y)$, the condition above says that $(x-y)\cdot(z-y) \geq 0$, i.e., no three points among $\{\pm \mathbf{v}_i\}_i$ span an obtuse angle. Similarly to ℓ_2^2 metrics, it is possible to optimize a linear objective functions in the distances over $\mathrm{NEG_{sym}}$ metrics.

Theorem 1 (Goemans, 2000) *Every NEG_{sym} metric on p dimension can be embedded into ℓ_2 with distortion $O(\sqrt{p})$.*

Notice that if one could get a logarithmic dimension reduction for $\mathrm{NEG_{sym}}$ metrics á la Johnson-Lindenstrauss, then Theorem 1 would imply that such metrics are embeddable into ℓ_2 with distortion $O(\sqrt{\log n})$. In fact, it follows from [13] that applying Johnson Lindenstrauss lemma for an ℓ_2^2 metric would result in a low-dimensional ℓ_2^2 metric that cannot violate triangle inequality by a large margin. So it would seem reasonable to expect that such dimension reductions are possible. An $O(\sqrt{\log n})$ distortion achieved this way would improve the results of Arora, Lee and Naor [3], and would greatly simplify [4, 3], being based on purely geometrical principles rather than combinatorial and geometrical ones. This question of whether such dimension reductions exist was raised in a workshop on metric geometry at Texas A&M, Summer 2006 [1].

In this work we show that there is no dimension reduction in ℓ_2^2 (or even for $\mathrm{NEG_{sym}}$) in a strong sense: whenever the distortion depends only on the number of points, one cannot reduce the dimension below the trivial $n-1$. Specifically, we show

Theorem 2 *For any real function $D(n)$, there exists an n-point metric space X in ℓ_2^2 (or NEG_{sym}) such that for every metric space Y in ℓ_2^2 that is associated with less than $n-1$ dimension, the distortion required for embedding X into Y is greater than $D(n)$.*

2 The construction

Recall that if ρ is an ℓ_2^2 metric then $\sqrt{\rho}$ is a metric that is obtained by taking points in Euclidean space where no three of them spans an obtuse angle. Therefore, dimensionality reduction in ℓ_2^2 metrics amounts to dimensionality reduction of a set of points in Euclidean space that span no obtuse angle, with the additional requirement that the image of the points do not span such angles as well.

The previous results on the minimum required dimension of a metric were due to Brinkman and Charikar [5] and to Lee and Naor [11] in the ℓ_1 case.

An obvious first attempt would be to use random projections as in the Johnson Lindenstrauss lemma. Doing so certainly preserve distances approximately, and in fact allows for only small changes in angles: the sine of the angles change by an arbitrary small factor [13]; but that is not strong enough when the angles in question are close to $\pi/2$. Indeed, it is easy to see that under a random projection a right angle will become obtuse with probability $1/2$. In particular, the Johnson-Lindenstrauss Lemma itself is not a good approach to the question of dimension reduction for ℓ_2^2 metrics. With this in mind, it is not surprising that the bad example we exhibit contains many right angles.

Let $c > 1$ be a constant, let $\mathbf{p_0} = -\frac{1}{2}\sum_{i=1}^{n} c^{i-1}\mathbf{e_i}$, and recursively define

$$\mathbf{p_j} = \mathbf{p_{j-1}} + c^{j-1}\mathbf{e_j}.$$

Let $X(n,c)$ be the $(n+1)$-point metric space given by the squared Euclidean distances between the points $\mathbf{p_i}$. Notice that $X(n,c)$ is simply the line metric on points spaced at intervals with lengths that increase exponentially (as power of c). Further, since all the $\mathbf{p_i}$s lie on vertices of a box centred at the origin, no three points among $\pm\mathbf{p_i}$ span an obtuse angle. Therefore $X(n,c)$ is a $\mathrm{NEG_{sym}}$ metric. We will show that there is a large enough constant c depending on D, such that in every embedding of $X(n,c)$ with distortion $\leq \sqrt{D(n)}$ into ℓ_2 the vectors $f(\mathbf{p_i}) - f(\mathbf{p_{i-1}})$ are arbitrarily close to being orthogonal. Since a set of vectors that are sufficiently close to being orthogonal must have full dimension, the dimension lower bound then follows.

Let f be an embedding from $\{\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_n}\}$ into $\mathbb{R}^d$, and let α be a positive number such that for every i, j, k

$$\|\mathbf{p_i} - \mathbf{p_j}\| \leq \|f(\mathbf{p_i}) - f(\mathbf{p_j})\| \leq \alpha\|\mathbf{p_i} - \mathbf{p_j}\|,$$

and

$$(f(\mathbf{p_i}) - f(\mathbf{p_j}))\cdot(f(\mathbf{p_k}) - f(\mathbf{p_j})) \geq 0.$$

Denote the vector $f(\mathbf{p_i}) - f(\mathbf{p_{i-1}})$ by $\mathbf{w_i}$. Much of the argument we need is captured by the following lemma bounding the angles between the vectors $\mathbf{w_i}$s.

Lemma 3 *For $i \neq j$, $|\mathbf{w_i}\cdot\mathbf{w_j}| \leq \frac{\alpha}{c-1}\|\mathbf{w_i}\|\|\mathbf{w_j}\|$*

Proof. Assume without loss of generality that $i < j$. Focusing on the points $\mathbf{p_{i-1}}$, $\mathbf{p_i}$ and $\mathbf{p_j}$ we get that

$$0 \le (f(\mathbf{p_{i-1}}) - f(\mathbf{p_i})) \cdot (f(\mathbf{p_j}) - f(\mathbf{p_i})) =$$

$$-\mathbf{w_i} \cdot \Big(\sum_{k=i+1}^{j} \mathbf{w_k} \Big),$$

therefore

$$\mathbf{w_i} \cdot \mathbf{w_j} \le -\mathbf{w_i} \cdot \Big(\sum_{k=i+1}^{j-1} \mathbf{w_k} \Big).$$

The distortion condition implies that $\|\mathbf{w_j}\| \ge c^j$ and that $\|\mathbf{w_k}\| \le \alpha c^k$ for every k. Therefore

$$\mathbf{w_i} \cdot \mathbf{w_j} \le -\mathbf{w_i} \cdot \Big(\sum_{k=i+1}^{j-1} \mathbf{w_k} \Big) \le \|\mathbf{w_i}\| \sum_{k=i+1}^{j-1} \alpha c^{k-1} \le$$

$$\alpha \|\mathbf{w_i}\| c^j/(c-1) \le \alpha \|\mathbf{w_i}\| \|\mathbf{w_j}\|/(c-1). \qquad (1)$$

To lower bound $\mathbf{w_i} \cdot \mathbf{w_j}$ we consider the angle between the same three points, with $i-1$ as the center point. We get

$$0 \le (f(\mathbf{p_i}) - f(\mathbf{p_{i-1}})) \cdot (f(\mathbf{p_j}) - f(\mathbf{p_{i-1}})) =$$

$$\mathbf{w_i} \cdot \Big(\sum_{k=i}^{j} \mathbf{w_k} \Big).$$

Now we have

$$\mathbf{w_i} \cdot \mathbf{w_j} \ge -\mathbf{w_i} \cdot \Big(\sum_{k=i}^{j-1} \mathbf{w_k} \Big)$$

and similarly to (1)

$$\mathbf{w_i} \cdot \mathbf{w_j} \ge -\mathbf{w_i} \cdot \Big(\sum_{k=i}^{j-1} \mathbf{w_k} \Big) \ge -\|\mathbf{w_i}\| \sum_{k=i}^{j-1} \alpha c^{k-1} \ge$$

$$-\alpha \|\mathbf{w_i}\| c^j/(c-1) \ge -\alpha \|\mathbf{w_i}\| \|\mathbf{w_j}\|/(c-1).$$

We refer the reader to Figure 1 which illustrates the geometrical intuition of the lemma.

□

Given any function $D(n)$, we set $c = n\sqrt{D(n)} + 1$. Assume that $X(n,c)$ can be embedded into an ℓ_2^2 metric with dimension less than n and distortion at most $D(n)$. Then there must be a function f from $\{\mathbf{p_0}, \ldots, \mathbf{p_n}\}$ that satisfies the conditions of Lemma 3 with $\alpha = \sqrt{D(n)}$ and the vectors $\mathbf{w_i}$s are in $\mathbb{R}^d$ with $d < n$. Let $\mathbf{w_i}' = \mathbf{w_i}/\|\mathbf{w_i}\|$ be the normalized vector $\mathbf{w_i}$. Then

$$|\mathbf{w_i}' \cdot \mathbf{w_j}'| = |\mathbf{w_i} \cdot \mathbf{w_j}|/\|\mathbf{w_i}\|\|\mathbf{w_j}\| \le$$

$$\frac{\alpha}{c-1} = \frac{\alpha}{n\sqrt{D(n)}} = 1/n.$$

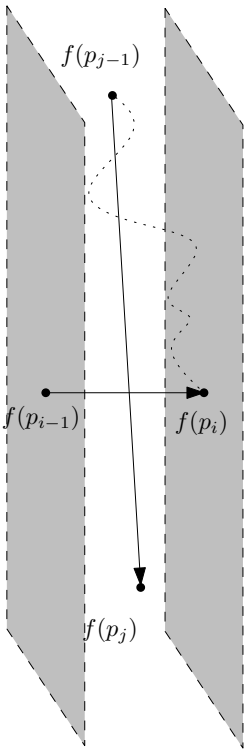


Figure 1: $f(\mathbf{p_{j-1}})$ and $f(\mathbf{p_j})$ must lie in the slab between $f(\mathbf{p_{i-1}})$ and $f(\mathbf{p_i})$; since $\mathbf{w_j} = f(\mathbf{p_j}) - f(\mathbf{p_{j-1}})$ has norm much larger than the width of the slab, $\mathbf{w_i}$ and $\mathbf{w_j}$ must be almost orthogonal.

But now we have a set of n unit vectors which are almost orthogonal. It is a well known fact that such a set must have full rank; for completeness we show it here. Let A be the gram matrix of $\mathbf{w_i}$'s that is $A_{i,j} = \mathbf{w_i}'\mathbf{w_j}'$. Then $A_{ii} = 1$ and $|A_{i,j}| \le 1/n$ for $i \ne j$. Thus, $\|A - I\|_\infty < \frac{1}{n}$, and for any vector $\mathbf{x} \ne \mathbf{0}$,

$$\|A\mathbf{x}\|_\infty \ge \|\mathbf{x}\|_\infty - \|(A-I)\mathbf{x}\|_\infty \ge$$

$$\|\mathbf{x}\|_\infty - (n-1)(1/n)\|\mathbf{x}\|_\infty > 0.$$

So A is nonsingular and therefore the dimension spanned by the $\mathbf{w_i}'$ (and so by their original counterparts $\mathbf{w_i}$) is n. We have therefore shown that

$$d \ge n = |X(n,c)| - 1$$

which proves Theorem 2.

References

[1] Open problems raised at the concentration week on metric geometry and geometric embeddings of discrete metric spaces. http://www.math.tamu.edu/research/workshops/linanalysis/problems.html, 2006.

[2] N. Alon. Problems and results in extremal combinatorics, part i. *Discrete Math.*, 273:31–53, 2003.

[3] S. Arora, J. R. Lee, and A. Naor. Euclidean distortion and the sparsest cut [extended abstract]. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 553–562, New York, 2005. ACM.

[4] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 222–231 (electronic), New York, 2004. ACM.

[5] B. Brinkman and M. Charikar. On the impossibility of dimension reduction in l1. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.

[6] M. Goemans. Strengthening of negative type metrics. In *New Methods in Discrete Mathematics*, Alpe d'Huez, March 2000.

[7] Piotr Indyk. Algorithmic aspects of geometric embeddings. In *FOCS*, volume 42, pages 10–33, 2001.

[8] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemp. Math.*, pages 189–206. Amer. Math. Soc., Providence, RI, 1984.

[9] S. Khot and N. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *Proceedings of The 46-th Annual Symposium on Foundations of Computer Science*, 2005.

[10] P. Kumar and E. A. Yildirim. Minimum-volume enclosing ellipsoids and core sets. *Journal of Optimization Theory and Applications*, 126(1):1–21, July 2005.

[11] J. R. Lee and A. Naor. Embedding the diamond graph in lp and dimension reduction in l1. *Geometric & Functional Analysis GAFA*, 14(4):745–747, August 2004.

[12] N. Linial, E. London, and Yu. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[13] A. Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Disc. Comput. Geom.*, 38(1):139–153, 2007.

[14] J. Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.

The Steiner Ratio for Obstacle-Avoiding Rectilinear Steiner Trees

Anna Lubiw* Mina Razaghpour†

Abstract

We consider the problem of finding a shortest rectilinear Steiner tree for a given set of points in the plane in the presence of rectilinear obstacles that must be avoided. We extend the *Steiner ratio* to the obstacle-avoiding case and show that it is equal to the Steiner ratio for the obstacle-free case.

1 Introduction

Given a set of points (also called *terminals*) and a set of obstacles in the plane, an obstacle-avoiding rectilinear Steiner minimum tree (OAR-SMT) is a tree of shortest length, composed solely of vertical and horizontal line segments, connecting the points and avoiding the interior of the obstacles. The OAR-SMT problem has important applications in VLSI design. For extensive surveys of Steiner tree problems, refer to [5] and [7].

A Steiner tree may contain vertices different from the points to be connected, namely Steiner points. If we do not allow Steiner points, then the problem becomes the minimum spanning tree problem. Whereas the Steiner problem has been proven to be NP-hard in both Euclidean and rectilinear metrics [3, 2], it is easy to determine the minimum spanning tree. Consequently, we are interested in the quality of a minimum spanning tree as an approximation to the minimum Steiner tree for various versions of the problem. The *Steiner ratio* is defined to be the maximum, over all instances, of the ratio of the length of a minimum spanning tree to the length of a Steiner minimum tree (SMT). For every metric space, the Steiner ratio is between 1 and 2 [4]. For the Euclidean Steiner tree problem (obstacle-free case), the Steiner ratio is $\frac{2}{\sqrt{3}}$. This was conjectured by Gilbert and Pollack in 1966 [4] and was proved in 1992 by Du and Hwang [1]. For the rectilinear Steiner tree problem, Hwang [6] proved earlier that the Steiner ratio is $\frac{3}{2}$.

What is the most natural generalization of this to the case of obstacles? An edge of a spanning tree must walk around the obstacles in this case. It seems natural to allow Steiner points at corners of obstacles. This does not lead to a polynomially solvable problem but, as we show here, does lead to interesting Steiner ratio results. We call an Steiner point *anchored* if it is at the corner of an obstacle. An *anchored-OAR-SMT* is then defined as an obstacle-avoiding rectilinear SMT in which all Steiner points are anchored. Note that when there are no obstacles, an anchored-OAR-SMT is a minimum spanning tree. We define the obstacle-avoiding Steiner ratio as the worst case ratio of the length of an anchored-OAR-SMT to the length of an OAR-SMT. We show that this ratio is $\frac{3}{2}$, which is the same as the Steiner ratio for the obstacle-free case.

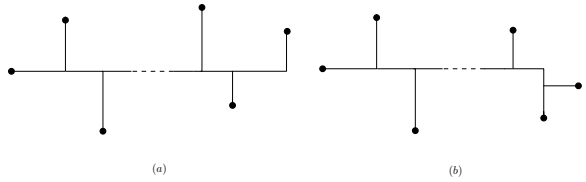


Figure 1: Two canonical forms of full Steiner trees.

Note that if we do not allow Steiner points, we do not get an interesting ratio in the case of obstacles. The worst case ratio between the lengths of an obstacle-avoiding minimum spanning tree and a Steiner minimum tree is 2, which is equal to the Steiner ratio in a generic metric space.

In the remainder of this paper, we use Steiner tree to mean a rectilinear obstacle-avoiding Steiner tree, unless otherwise specified.

2 Canonical Trees

In this section we show that it suffices to prove our Steiner ratio result for *canonical* Steiner trees, which have the forms shown in Figure 1. This was proved for the case without obstacles by Hwang [6] and we follow his approach. A *canonical* Steiner tree is defined as follows:

Definition 1 (Canonical Trees) *A rectilinear minimum Steiner tree is* canonical *if it has one of the following forms, possibly after a rotation:*

i. All Steiner points and the leftmost terminal lie on a horizontal line. All Steiner points are connected to exactly one terminal by a vertical edge. These vertical edges alternatingly extend up and down. The rightmost and leftmost Steiner points are connected to a second terminal by a horizontal edge or a corner (Figure 1(a)).

*School of Computer Science, University of Waterloo, alubiw@uwaterloo.ca

†School of Computer Science, University of Waterloo, mrazaghp@uwaterloo.ca

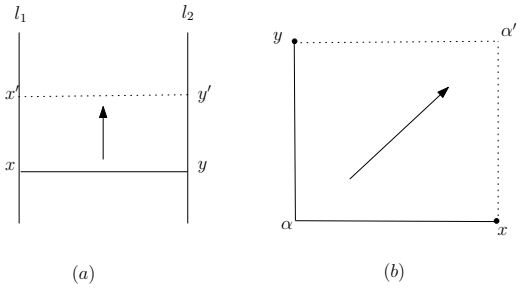


Figure 2: (a) Shifting. (b) Flipping.

ii. As above except that the two rightmost Steiner points are connected together by a corner (Figure 1(b)).

Hwang's first step is to reduce to *full* Steiner trees, which are Steiner trees in which terminals appear only as leaves. To justify this, note that we can cut a Steiner tree at any non-leaf terminal to obtain full Steiner subtrees. Similarly, we can cut an obstacle-avoiding Steiner tree at non-leaf terminals and at obstacle corners to obtain *full* obstacle-avoiding Steiner trees, in which terminals and obstacle corners appear only as leaves. It is sufficient to prove the ratio result for full subtrees (this is justified more formally below). Note that obstacle corners are then regarded as terminals in the full subtrees.

Hwang's next step is to apply *shifting* and *flipping* operations (Figure 2) to transform any minimum Steiner tree to one whose full subtrees are canonical. In a shift, a segment xy incident to two parallel lines l_1 and l_2 and containing no terminals or Steiner points other than possibly x and y is replaced by segment $x'y'$ also incident to l_1 and l_2 and parallel to xy. In a flip, two segments $x\alpha$ and $y\alpha$ meeting at the corner α and containing no terminals or Steiner points other than possibly x and y are replaced by segments $x\alpha'$ and $y\alpha'$, such that $x\alpha y\alpha'$ is a rectangle. These operations do not increase the length of a Steiner tree, and therefore map an SMT to another SMT. Two Steiner trees are said to be *equivalent* if one can be transformed to the other by shifting and flipping. To deal with obstacles, we first perform shifts and flips to bump into as many obstacles as possible. We then claim that further shifts and flips, as performed in Hwang's reduction, can ignore the obstacles. More formally:

Lemma 1 *Let T be an OAR-SMT such that, among all equivalent OAR-SMTs, T has the maximum number of full subtrees. Then shift and flip transformations can be done on T as if there were no obstacles, without violating the obstacle-avoiding property.*

Proof. Assume by contradiction that there exists a transformation H mapping T to T' such that T' violates the obstacle-avoiding property. Let H be a shift from segment xy to $x'y'$ such that $x'y'$ intersects an obstacle (Figure 3). Then, there must exist at least one obstacle corner inside the rectangle $xyy'x'$. Let c be the closest such obstacle corner to xy. We can shift xy to c to increase the number of full subtrees by at least one, thus contradicting the assumption that T has the maximum number of full subtrees. Next, let H be a flip from the corner $x\alpha y$ to the corner $x\alpha' y$ such that either $x\alpha'$ or $y\alpha'$ intersects with an obstacle. Since the obstacles are rectilinear, there must exist an obstacle corner inside the rectangle $x\alpha y\alpha'$. Let c be the closest such obstacle to α. We can flip α to c and increase the number of full subtrees by at least one, again leading to a contradiction.

□

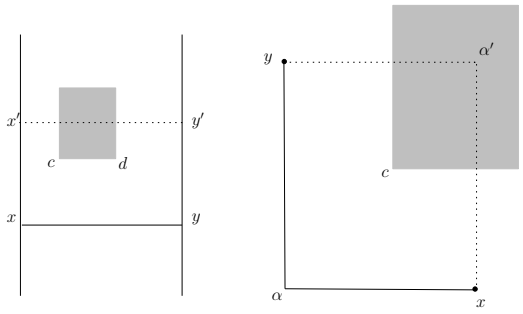


Figure 3: Shifting and flipping in the presence of obstacles.

Starting with an OAR-SMT with the maximum number of full subtrees, we can therefore apply Hwang's proof steps and reductions to get a canonical OAR-SMT:

Theorem 1 *For any terminal set P and obstacle set O there is an OAR-SMT whose full subtrees are in canonical form.*

Notation We call the horizontal line connecting the Steiner points the *spine* and denote it by E. We call the vertical lines connecting the terminals to the spine the *ribs*. Let n_u and n_l be the number of the ribs above and below the spine, respectively ($n_l = n_u$ or $n_l = n_u \pm 1$). Let $R_1, \ldots, R_{n_u}$ and $r_1, \ldots, r_{n_l}$ denote the ribs above and below the spine, in the order of x coordinate, respectively. Let T_i and t_i denote the terminals located on R_i and r_i, respectively. Denote by S_i and s_i the Steiner points at which R_i and r_i meet the spine, respectively. The rightmost rib and the leftmost rib (of length zero) meet the the spine at a corner point or a terminal, which for convenience of notation, we also denote by S_i and s_i for $i = 1, n_l, n_u$. We define a *pocket* as a subtree connecting three terminals consecutive in x-ordering. Without loss of generality, we assume that

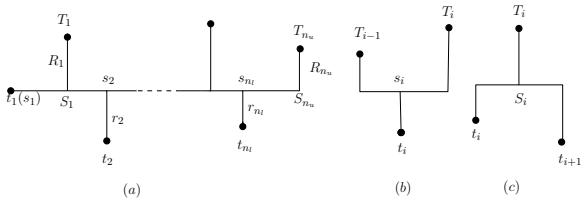


Figure 4: (a) A canonical Steiner tree. (b) Upper pocket P_i. (c) Lower pocket p_i.

the second leftmost rib is an upper rib. Then the ith upper pocket, denoted by P_i, connects T_{i-1}, t_i and T_i via s_i and the ith lower pocket, denoted by p_i, connects t_i, T_i and t_{i+1} via S_i. These notations are illustrated in Figure 4.

3 Rectilinear Steiner Ratio With Obstacles

Let T^* and AT^* denote an OAR-SMT and an anchored-OAR-SMT for a terminal set P, respectively.

Theorem 2

$$\frac{|AT^*|}{|T^*|} \leq \frac{3}{2}$$

Proof. The proof only needs to be established for canonical Steiner trees. To see why, consider an OAR-SMT T' equivalent to T^* whose full subtrees are canonical. Such a tree exists by Theorem 1. Assume that Theorem 2 holds for each full subtree F_i. Therefore, there exists an anchored OAR-SMT G_i for the terminal set spanned by F_i such that $|G_i| \leq \frac{3}{2}|F_i|$. The union of all G_i's, denoted by G, is an anchored Steiner tree spanning P that is no longer than $\frac{3}{2}$ times the length of T^*.

We therefore assume that T^* is a canonical Steiner minimal tree. We will build a pair of anchored Steiner trees on P whose lengths add up to $3|T^*|$. The smaller tree will therefore have length at most $\frac{3}{2}|T^*|$.

We use the notation given at the end of Section 2. First, we assume that T^* is a type (i) canonical tree. We identify a subset of obstacle corners, called *critical corners*, and use them as Steiner points in the construction of the two trees. All other obstacle corners can be ignored.

For each upper (lower) pocket consider the set of all obstacle corners located above (below) the spine and between the two upper (lower) ribs. The height of such an obstacle corner is defined as its distance from the spine. We can restrict attention to the obstacles whose heights are less than the length of the shorter rib of their pocket. We define a *U-critical (L-critical) corner* as the obstacle corner with the minimum height in this set, breaking ties arbitrarily. If there is no such obstacle corner, the critical corner is the terminal located on the shorter upper (lower) rib and we refer to it as a *virtual* critical corner. Note that the length of the shortest path between a critical corner and a terminal in its pocket is equal to their rectilinear distance. Let O_i (o_i) denote the U-critical (L-critical) corner in the upper pocket P_i (lower pocket p_i), and let I_i^O (I_i^o) be its image projected on the spine. Let H_i^O (H_i^o) be the height of O_i (o_i).

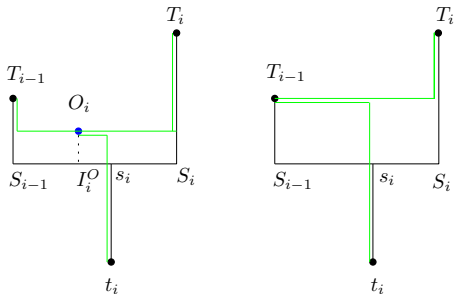


Figure 5: The Green Tree.

3.1 The green tree

For each upper pocket, we connect the three terminals t_i, T_{i-1} and T_i to the U-critical corner O_i. See Figure 5. The length of the subtree is:

$$|T_{P_i}| = |R_{i-1}| + |R_i| - |H_i^o| + |r_i| + |S_{i-1}S_i| + |s_i I_i^O|$$

We connect the boundary terminals, if not included in any upper pocket, directly to the next terminal in the x-ordering. Summing over all upper pockets, the length of the green tree is:

$$|T_{green}| = 2\sum_{i=1}^{n_u}|R_i| + \sum_{i=1}^{n_l}|r_i| - \sum_{i=1}^{n_u}|H_i^O| + |E| + \sum_{i=1}^{n_u}|s_i I_i^O|$$

3.2 The red tree

A U-critical corner O_j is *involved* in a lower pocket p_i, if O_j's image on the spine, I_j^O, is located between the boundary Steiner points s_i and s_{i+1}. O_j can be either involved in the pocket p_j or p_{j-1}.

For each lower pocket p_i, there can be 0, 1 or 2 U-critical corners involved in the pocket. We consider three cases:

Case 1: There is one U-critical corner, O_j ($j = i$ or $i+1$), involved in p_i. We connect t_i, T_{i-1} and T_i to O_j. See Figure 6 (a) and (b). In this case, the length of the subtree is:

$$\begin{aligned} |T_{p_i}| &= |R_i| + |r_i| + |r_{i+1}| + |H_i^O| + |s_i s_{i+1}| + |S_i I_j^O| \\ &\leq |R_i| + |r_i| + |r_{i+1}| + |H_i^O| + 2|s_i s_{i+1}| - |s_j I_j^O| \end{aligned}$$

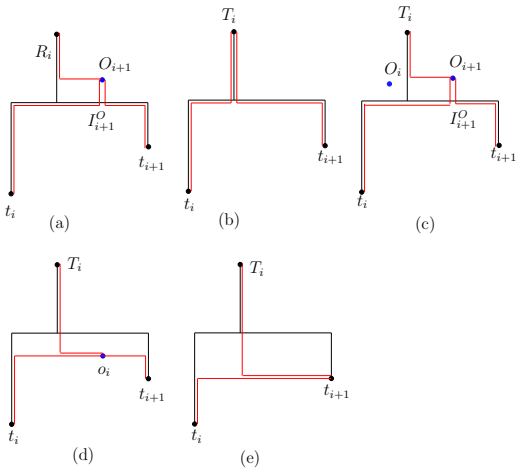


Figure 6: The Red Tree: (a) one (non-virtual) upper critical corner. (b)Virtual upper critical corner. (c)Two upper critical corners. (d) No upper critical corner, at least one (non-virtual) lower critical corner. (e) No upper critical corner, virtual or no lower critical obstacle.

Case 2: There are two U-critical corners, O_i and O_{i+1}, involved in p_i. We connect t_i, T_{i-1} and T_i to O_{i+1}. See Figure 6(c). The length of the subtree is:

$$|T_{p_i}| = |R_i| + |r_i| + |r_{i+1}| + |H^O_{i+1}| + |s_i s_{i+1}| + |S_i I^O_{i+1}|$$

We want the term $|H^O_i| - |s_i I^O_i|$ to appear exactly once for each U-critical corner O_i, so that it is canceled by the term $|s_i I^O_i| - |H^O_i|$ in the green tree's length. Therefore, we re-write the length of the subtree as:

$$\begin{aligned} |T_{p_i}| &\leq |R_i| + |r_i| + |r_{i+1}| + |H^O_i| + |H^O_{i+1}| \\ &+ 2|s_i s_{i+1}| - |s_i I^O_i| - |s_{i+1} I^O_{i+1}| \end{aligned}$$

Case 3: There are no U-critical corners involved in p_i. In this case, we use the L-critical corner in the pocket, o_i, as a Steiner point and connect the three terminals t_i, T_{i-1} and T_i to o_i. See Figure 6 (d) and (e).

The length of the subtree is:

$$|T_{p_i}| \leq |R_i| + |r_i| + |r_{i+1}| + 2|s_i s_{i+1}|$$

We connect the boundary terminals, if not included in any lower pocket, directly to the next terminal in x-ordering. Summing over all lower pockets, the length of the red tree is :

$$|T_{red}| \leq \sum_{i=1}^{n_u} |R_i| + 2\sum_{i=1}^{n_l} |r_i| + \sum_{i=1}^{n_u} |H^O_i| + 2|E| - \sum_{i=1}^{n_u} |s_i I^O_i|$$

Now we add up the lengths of the two trees together:

$$|T_{green}| + |T_{red}| \leq 3\sum_{i=1}^{n_u} R_i + 3\sum_{i=1}^{n_l} r_i + 3|E| = 3|T^*|$$

This proves that the length of the shorter tree is at most $\frac{3}{2}$ times the length of T^*.

Now assume that T^* is a type (ii) canonical tree. First, we ignore the exceptional terminal, call it u, and build the red and green trees as above. Then, we modify the red tree so that the path between t_{n_l} and T_{n_u} passes through u, and in the green tree, we connect u to T_{n_u}. It is easy to see that the lengths of the two trees still add up to less than 3 times the length of T^*.

Finally, the $\frac{3}{2}$ bound for the obstacle-avoiding Steiner ratio is tight, since it clearly cannot be less than the Steiner ratio for the obstacle-free case.

□

4 Future work

We are working on an approximation algorithm to compute the anchored-OAR-SMT. We conjecture that in the Euclidean case, the obstacle-avoiding Steiner ratio is $\frac{2}{\sqrt{3}}$, the same as the Steiner ratio for the Euclidean obstacle-free case.

References

[1] D. Du and F. K. Hwang. A Proof of the Gilbert-Pollak conjecture on the Steiner ratio. *Algorithmica*, 7(2&3):121–135, 1992.

[2] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977.

[3] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[4] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[5] F. Hwang, D. Richards, and P. Winter. *Steiner tree problem*. North-Holland, Amsterdam ; New York, 1992.

[6] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics*, 30(1):104–114, 1976.

[7] H. J. Promel and A. Steger. *The Steiner tree problem: a tour through graphs, algorithms, and complexity (Vieweg advanced lectures in mathematics)*. Friedrick Vieweg & Son, 2002.

Core-Preserving Algorithms

Hamid Zarrabi-Zadeh*

Abstract

We define a class of algorithms for constructing coresets of (geometric) data sets, and show that algorithms in this class can be dynamized efficiently in the insertion-only (data stream) model. As a result, we show that for a set of points in fixed dimensions, additive and multiplicative ε-coresets for the k-center problem can be maintained in $O(1)$ and $O(k)$ time respectively, using a data structure whose size is independent of the size of the input. We also provide a faster streaming algorithm for maintaining ε-coresets for fat extent-related problems such as diameter and minimum enclosing ball.

1 Introduction

The data stream model of computation has recently attracted considerable interest due to growing applications involving massive data sets. In this model, data is presented to the algorithm one by one as a stream over time, and the algorithm must compute a function over the stream in only one pass, using a limited amount of storage.

The coreset framework is a fundamental tool for designing algorithms in the data stream model as it allows to compute a function approximately over the data stream by keeping only a small-size "sketch" of the input, called a coreset. Roughly speaking, a subset Q of the input set P is called an *ε-coreset* of P with respect to an optimization problem, if solving the optimization problem on Q gives an ε-approximate solution to the problem on the whole input set, P.

Several streaming algorithms have been developed over the past few years for various geometric problems using the notion of coresets [1, 6, 9, 14]. For all these problems, coresets defined satisfy the following two properties:

a) If Q is an ε-coreset of P and Q' is an ε-coreset of P', then $Q \cup Q'$ is an ε-coreset of $P \cup P'$;

b) If Q is an ε-coreset of S and S is an δ-coreset of P, then Q is an $(\varepsilon + \delta)$-coreset of P.

Using the above two properties and based on the general dynamization technique of Bentley and Saxe [5], Agarwal *et al.* [2] obtained the following result in the data stream model: If there is an ε-coreset of size $f(\varepsilon)$ for a problem, then one can solve the problem in the data stream model using $O(f(\varepsilon/\log^2 n)\log n)$ overall space, where n is the number of elements received so far in the stream.

In this paper, we show that for a special class of algorithms which we call core-preserving, the space complexity of the corresponding streaming algorithms can be reduced to $f(\varepsilon)$, using a simple bucketing scheme. The importance of this result is that the dependency of the space complexity to the input size, n, is removed. (Such a result was previously known only for the ε-coresets with respect to the extent measure [6, 4].) This independency to the input size is very important as the input size in the data streams is usually huge.

Our framework leads to improved algorithms for a number of problems in the data stream model, some of which are listed below. In the following, the input is assumed to be a stream of points in $\mathbb{R}^d$, where d is a constant.

- **(Additive) coreset for k-center**: We show that an additive ε-coreset for the k-center problem can be maintained in $O(k/\varepsilon^d)$ space and $O(1)$ amortized update time, improving the previous algorithm attributed to Har-peled [12] which requires $O(\text{poly}(k, 1/\varepsilon, \log n))$ space and similar time. This is indeed the first streaming algorithm maintaining an ε-coreset for this problem using a total space independent of n.

- **Multiplicative coreset for k-center**: For the k-center problem, we show that a multiplicative ε-coreset (as defined in Section 2) can be maintained in $O(k!/\varepsilon^{kd})$ space and $O(k)$ amortized update time. This is again the first streaming algorithm for this problem whose space is independent of the input size. This result immediately extends to a variant of the k-clustering problem in which the objective is to minimize the sum of the clusters radii [7, 10].

- **Coreset for fat measures**: For "fat" measures such as diameter and radius of the minimum enclosing ball, one can easily maintain an ε-coreset by just keeping the extreme points along $O(1/\varepsilon^{(d-1)/2})$ directions. The time and space complexity of this naïve algorithm is $O(1/\varepsilon^{(d-1)/2})$. In two-dimensions, using the recent algorithm of Agarwal

*School of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada; hzarrabi@uwaterloo.ca

and Yu [4], one can improve the update time from $O(\sqrt{1/\varepsilon})$ to $O(\log(1/\varepsilon))$. We show that the update time in 2D can be further reduced to $O(1)$ using our framework. Moreover, the update time in three dimensions is reduced from $O(1/\varepsilon)$ to $O(\log(1/\varepsilon))$ using our algorithm. A slight improvement in higher dimensions is implied as well.

2 Preliminaries

Let P be a set of points in $\mathbb{R}^d$. A *k-clustering* of P is a set $\mathcal{B}$ of k balls that completely cover P. We denote by $\mathrm{rad}(b)$ the radius of a ball b, and define $\mathrm{rad}(\mathcal{B}) = \max_{b\in\mathcal{B}} \mathrm{rad}(b)$. A *$\delta$-expansion* of $\mathcal{B}$ is obtained by increasing the radius of each ball of $\mathcal{B}$ by an additive factor of δ.

Definition 1 A set $Q \subseteq P$ is called an *additive ε-coreset* of P for the k-center problem, if for every k-clustering $\mathcal{B}$ of Q, P is covered by an $(\varepsilon \cdot \mathrm{rad}(\mathcal{B}))$-expansion of $\mathcal{B}$.

We denote by $(1+\varepsilon)\mathcal{B}$ a clustering obtained from $\mathcal{B}$ by expanding each ball $b \in \mathcal{B}$ by a factor of $\varepsilon \cdot \mathrm{rad}(b)$.

Definition 2 A set $Q \subseteq P$ is called a *multiplicative ε-coreset* of P for the k-center problem, if for every k-clustering $\mathcal{B}$ of Q, P is covered by $(1+\varepsilon)\mathcal{B}$.

Given two points $p, q \in \mathbb{R}^d$, we say that p is *smaller* than q, if p lies before q in the lexicographical order of their coordinates. Throughout this paper, we denote by $\lfloor x \rfloor_2$ the largest (integer) power of 2 which is less than or equal to x.

3 Core-Preserving Algorithms

In this section, we formally define the notion of core-preserving algorithms, and show how it can be used to efficiently maintain coresets in data streams.

Definition 3 *Let $\mathcal{A}$ be an (offline) algorithm that for every input set P, computes an ε-coreset $\mathcal{A}(P)$ of P. We call $\mathcal{A}$* core-preserving*, if for every two sets R and S, $\mathcal{A}(R \cup \mathcal{A}(S))$ is an ε-coreset of $R \cup S$.*

For $R = \emptyset$, the above property implies that $\mathcal{A}(\mathcal{A}(S))$ is an ε-coreset of S. It means that repeated calls to a core-preserving algorithm on a set S always returns an ε-coreset of S. This is why the algorithm is called "core-preserving".

Theorem 1 *Let $\mathcal{A}$ be a core-preserving algorithm that for any set S, computes an ε-kernel of S of size $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$ in time $O(\alpha|S| + \mathcal{T}_{\mathcal{A}}(\varepsilon))$. Then for every stream P, we can maintain an ε-coreset of P of size $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$ using $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$ total space and $O(\alpha + \mathcal{T}_{\mathcal{A}}(\varepsilon)/\mathcal{S}_{\mathcal{A}}(\varepsilon))$ amortized time per update.*

Proof. The function INSERT described below inserts a date item p into the stream P and returns an ε-kernel of P. Initially, Q and R are empty sets.

INSERT(p):

1: $R \leftarrow R \cup \{p\}$
2: **if** $|R| > \mathcal{S}_{\mathcal{A}}(\varepsilon)$ **then**
3: $\quad Q \leftarrow \mathcal{A}(R \cup Q)$
4: $\quad R \leftarrow \emptyset$
5: return $Q \cup R$

The algorithm divides the input stream P into buckets of size $\lceil \mathcal{S}_{\mathcal{A}}(\varepsilon) \rceil$. At any time, only the last bucket is active which is maintained in the set R. Let $S = P \setminus R$. The algorithm maintains an ε-coreset of S in Q. Upon arrival of a new item p, it is first added to the active bucket R, and if R is full, algorithm $\mathcal{A}$ is invoked to compute an ε-coreset of $R \cup Q$. The correctness of the algorithm immediately follows from the facts that $\mathcal{A}$ is core-preserving and Q is an ε-coreset of S; thus, $\mathcal{A}(R \cup Q)$ is an ε-coreset of $R \cup S = P$.

The total space used by the algorithm is bounded by $|Q| + |R| = O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$. Algorithm $\mathcal{A}$ is invoked once per $\lceil \mathcal{S}_{\mathcal{A}}(\varepsilon) \rceil$ inserts. Since each call to $\mathcal{A}$ requires $O(\alpha|S| + \mathcal{T}_{\mathcal{A}}(\varepsilon))$ time, the amortized update time per input is $O(\alpha + \mathcal{T}_{\mathcal{A}}(\varepsilon)/\mathcal{S}_{\mathcal{A}}(\varepsilon))$. □

Theorem 1 yields two major improvements over the general Bentley-Saxe method used in [2]: First of all, the total space required is reduced from $O(\mathcal{S}_{\mathcal{A}}(\varepsilon/\log^2 n)\log n)$ to $O(\mathcal{S}_{\mathcal{A}}(\varepsilon))$, which is independent of n. Secondly, the running time in the worst case is reduced from $O([\alpha\mathcal{S}_{\mathcal{A}}(\varepsilon/\log^2 n) + \mathcal{T}_{\mathcal{A}}(\varepsilon/\log^2 n)]\log n)$ to only $O(\alpha|P| + \mathcal{T}_{\mathcal{A}}(\varepsilon))$, again independent of n.

4 Additive Coreset for k-Center

In this section, we provide an efficient streaming algorithm for maintaining an additive ε-coreset for the k-center problem in fixed dimensions.

Lemma 2 *There is a core-preserving algorithm that for any given point set $P \subseteq \mathbb{R}^d$, computes an additive ε-coreset for the k-center problem of size $O(k/\varepsilon^d)$ in time $O(|P| + k/\varepsilon^d)$.*

Proof. Let $r^*(P)$ be the radius of the optimal k-clustering of P, and $\tilde{r}(P)$ be a 2-approximation of $r^*(P)$, i.e., $r^*(P) \leqslant \tilde{r}(P) \leqslant 2r^*(P)$.

We first define some notations: Let $\mathcal{G}_\alpha$ be a uniform grid of side length α, and $X_\alpha(P)$ be the set of all $p \in P$, such that p is the smallest point in a non-empty grid cell of $\mathcal{G}_\alpha$. Let $\delta(P) = \lfloor \varepsilon\tilde{r}(P)/(4d^{1/2}) \rfloor_2$. Our core-preserving algorithm is as follows: given a point set P,

we first compute $\delta = \delta(P)$, and return $X_\delta(P)$ as the output. It is easy to observe that any k-clustering of $X_\delta(P)$, when expanded by a factor of $\varepsilon r^*(P)$, covers all the grid cells containing at least one point from P, and therefore, $X_\delta(P)$ is an ε-coreset of P [3, 13].

Let R and S be two arbitrary point sets in $\mathbb{R}^d$, and let Q be an ε-coreset of S computed by our algorithm. To show that our algorithm is core-preserving, we need to prove that for any input of the form $P = R \cup Q$, the algorithm returns an ε-coreset of $R \cup S$.

Let $\delta = \delta(P)$, $\sigma = \delta(S)$, and $\rho = \max\{\delta(P), \delta(S)\}$. Obviously, $X_\rho(R \cup S)$ is an ε-coreset of $R \cup S$, because both P and S are subsets of $R \cup S$, and hence, $\max\{\tilde{r}(P), \tilde{r}(S)\} \leqslant 2r^*(R \cup S)$. We claim that $X_\rho(R \cup S) \subseteq X_\delta(R \cup Q)$. Since ρ/δ (resp., ρ/σ) is a non-negative power of 2, every grid cell of $\mathcal{G}_\delta$ (resp., $\mathcal{G}_\sigma$) is completely contained in a grid cell of $\mathcal{G}_\rho$ (see Figure 1). Let p be the smallest point of $R \cup S$ in a grid cell c of $\mathcal{G}_\rho$. Two cases arise:

- $p \in R$: in this case, p is the smallest point of a cell $c' \in \mathcal{G}_\delta$ (otherwise, there is a point p' smaller than p in c', which is smaller than p in c as well, a contradiction). Therefore, $p \in X_\delta(R \cup Q)$.
- $p \in S$: here, p is simultaneously the smallest point of a cell $c' \in \mathcal{G}_\sigma$ and a cell $c'' \in \mathcal{G}_\delta$ (otherwise, if there is a smaller point p' in either c' or c'', it would be picked instead of p as the smallest point of c, a contradiction). Since p is the smallest point in c', we have $p \in Q$, and since p is the smallest point of c'', we conclude that $p \in X_\delta(R \cup Q)$.

Therefore, any $p \in X_\rho(R \cup S)$ is contained in $X_\delta(R \cup Q) = X_\delta(P)$, which completes the proof.

For the space complexity, note that every ball of an optimal k-clustering of P intersects $O(1/\varepsilon^d)$ grid cells of $\mathcal{G}_\delta$. Therefore, the size of the resulting ε-coreset is $O(k/\varepsilon^d)$. We can use a linear-time implementation of Gonzalez's algorithm [11, 12] to compute a 2-approximation of $r^*(P)$, and therefore, the total running time required is $O(|P| + k/\varepsilon^d)$. □

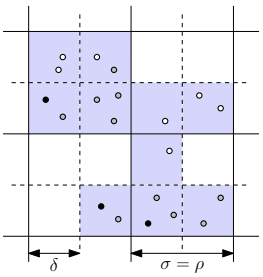


Figure 1: Additive coreset for k-center. The points of R, Q, and $S \setminus Q$ are shown in white, black, and gray, respectively.

Plugging Lemma 2 into the general framework provided in Theorem 1, we immediately get the following result.

Theorem 3 *Given a stream of points P in $\mathbb{R}^d$, an additive ε-coreset for the k-center problem of size $O(k/\varepsilon^d)$ can be maintained using $O(k/\varepsilon^d)$ total space and $O(1)$ amortized time per update.*

The above results also hold for any L_p metric: it just suffices to replace $d^{1/2}$ by $d^{1/p}$ in the definition of $\delta(P)$. The algorithm for multiplicative ε-coresets is omitted in this extended abstract.

5 Coresets for Fat Extent-Related Problems

Given a point set $P \subseteq \mathbb{R}^d$, let $\mathbb{B}(P)$ denote the minimum axis-parallel hyperbox enclosing P. We denote by $\ell(P)$ the length of the longest side of $\mathbb{B}(P)$. A subset $Q \subseteq P$ is called an *additive ε-kernel of P*, if for all $u \in \mathbb{S}^{d-1}$,

$$w(Q, u) \geqslant w(P, u) - \varepsilon\ell(P),$$

where $w(P, u) = \max_{p,q \in P} \langle p - q, u \rangle$.

A function $\mu(\cdot)$ defined over subsets of $\mathbb{R}^d$ is called a *fat measure*, if there exists a constant $\alpha > 0$ such that for any additive ε-kernel Q of P, $\alpha\mu(P) \leqslant \mu(Q) \leqslant \mu(P)$. Examples of fat measures are diameter, radius of the minimum enclosing ball, and width of the smallest enclosing hypercube. Obviously, if Q is an additive ε-kernel of P and μ is a fat measure, then Q is an (ε/α)-coreset of P with respect to μ.

Given a point set $P \subseteq \mathbb{R}^d$, an additive ε-kernel of P can be computed efficiently using an adaptation of the simple grid-rounding method proposed in [6, 15] based on Dudley's construction [8]. The algorithm is described in the following lemma.

Lemma 4 *There is a core-preserving algorithm that for every point set $P \subseteq \mathbb{R}^d$, computes an additive ε-kernel of P of size $O(1/\varepsilon^{(d-1)/2})$ in $O(|P| + 1/\varepsilon^{d-(3/2)})$ time for $d \geqslant 2$, or in $O((|P| + 1/\varepsilon^{d-2}) \log(1/\varepsilon))$ time for $d \geqslant 3$.*

Proof. We assume w.l.o.g. that $\mathrm{conv}(P)$ contains the origin. Let $\mathcal{B}(P)$ be the smallest hypercube centered at the origin containing P. If $\ell'(P)$ denotes the side length of $\mathcal{B}(P)$, then obviously $\ell(P) \leqslant \ell'(P) \leqslant 2\ell(P)$.

Let $B = \mathcal{B}(P)$. By a simple scaling, we may assume that $B = [-1, 1]^d$. Let $\mathcal{R}$ be the set of points of a $\sqrt{\varepsilon}$-grid over the boundary of the cube $[-2, 2]^d$, and let p_r denote the nearest neighbor of a point $r \in \mathcal{R}$ in the set P (see Figure 2). Let $\mathcal{Q} = \{p_r \mid r \in \mathcal{R}\}$. Obviously, $|\mathcal{Q}| \leqslant |\mathcal{R}| = O(1/\varepsilon^{(d-1)/2})$. Moreover, $\mathcal{Q}$ is an additive ε-kernel of P with the argument provided below. The running time follows immediately from the fast implementation of Chan using the discrete nearest neighbor queries [6].

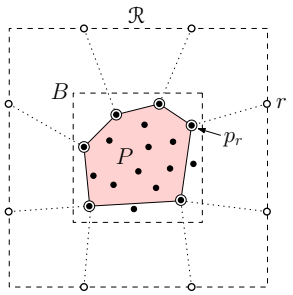


Figure 2: Construction of additive ε-kernel.

Consider two arbitrary point sets R and S in $\mathbb{R}^d$, and let Q be an additive ε-kernel of S computed by our algorithm. In order for our algorithm to be core-preserving, we need to show that for any input of the form $P = R \cup Q$, the algorithm returns an additive ε-kernel of $R \cup S$.

We adapt the proof from [6]. Fix a unit vector $u \in \mathbb{S}^{d-1}$ and a point $p \in R \cup S$. Obviously, there is a point $r \in \mathcal{R}$ such that $\angle(r-p, u) \leqslant \arccos(1-\varepsilon/8)$ (See [6], Observation 2.3). If $p_r \in S$, then by our construction there is a point $q \in Q$ such that $\|r-q\| \leqslant (1+c\varepsilon)\|r-p_r\|$ (details omitted). If $p_r \in R$, we simply set $q = p_r$. Therefore,

$$\begin{aligned}
& \|r-q\| \leqslant (1+c\varepsilon)\|r-p\| \\
\Rightarrow\ & (1-\varepsilon/8)\,\langle r-q, u\rangle \leqslant (1+c\varepsilon)\,\langle r-p, u\rangle \\
\Rightarrow\ & \langle r-q, u\rangle - 3\sqrt{d}\varepsilon/8 \leqslant \langle r-p, u\rangle + 3c\sqrt{d}\varepsilon \\
& \quad \text{(since } \|r-p\| \leqslant 3\sqrt{d} \text{ and } \|r-q\| \leqslant 3\sqrt{d}\text{)} \\
\Rightarrow\ & \langle p, u\rangle \leqslant \langle q, u\rangle + 3\sqrt{d}(c+1/8).
\end{aligned}$$

It means that the projections of p and q in direction u differ by at most $O(\varepsilon)$. Since $\ell(P) \geqslant 1/2$, we conclude that $\langle p-q, u\rangle = O(\varepsilon)\ell(P)$ in every direction u, which completes the proof. □

Combining Lemma 4 with Theorem 1, we get the following result:

Theorem 5 *Given a stream of points P in $\mathbb{R}^d$ and a fat measure μ, an ε-coreset of P with respect to μ can be maintained using $O(1/\varepsilon^{(d-1)/2})$ total space and $\max\{O(1), O((1/\varepsilon^{(d-3)/2})\log(1/\varepsilon))\}$ amortized time per update.*

Remark. Using our framework to maintain ε-coresets of fat sets as a subroutine, we have recently succeeded to obtain a streaming algorithm for maintaining ε-coresets with respect to the general extent measure using near optimal space [16]. This leads to improved streaming algorithms for a wide variety of geometric optimization problems, including width, minimum enclosing cylinder, minimum-width enclosing annulus, minimum-width enclosing cylindrical shell, etc.

Acknowledgements The author would like to thank Timothy M. Chan for his helpful comments.

References

[1] P. K. Agarwal and S. Har-Peled. Maintaining approximate extent measures of moving points. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 148–157, 2001.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

[3] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

[4] P. K. Agarwal and H. Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proc. 23rd Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 2007.

[5] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformations. *J. Algorithms*, 1:301–358, 1980.

[6] T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.*, 35(1–2):20–35, 2006.

[7] M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Systems Sci.*, 68:417–441, Mar. 2004.

[8] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10:227–236, 1974.

[9] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. In *Proc. 37th Annu. ACM Sympos. Theory Comput.*, pages 209–217, 2005.

[10] M. Gibson, G. Kanade, E. Krohn, I. A. Pirwani, and K. Varadarajan. On clustering to minimize the sum of radii. In *Proc. 19th ACM-SIAM Sympos. Discrete Algorithms*, pages 819–825, 2008.

[11] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[12] S. Har-Peled. Clustering motion. *Discrete Comput. Geom.*, 31(4):545–565, 2004.

[13] S. Har-Peled. No Coreset, No Cry. In *Proc. 24th Conf. Found. Soft. Tech. and Theoret. Comput. Sci.*, pages 324–335, 2004.

[14] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, pages 291–300, 2004.

[15] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using core sets. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 263–272, 2004.

[16] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. In *Proc. 16th Annu. European Sympos. Algorithms*, 2008, to appear.

Achieving Spatial Adaptivity while Finding Approximate Nearest Neighbors

Jonathan Derryberry Don Sheehy Daniel D. Sleator Maverick Woo*

Abstract

We present the first spatially adaptive data structure that answers approximate nearest neighbor (ANN) queries to points that reside in a geometric space of any constant dimension d. The L_t-norm approximation ratio is $O(d^{1+1/t})$, and the running time for a query q is $O(d^2 \lg \delta(p,q))$, where p is the result of the preceding query and $\delta(p,q)$ is the number of input points in a suitably-sized box containing p and q. Our data structure has $O(dn)$ size and requires $O(d^2 n \lg n)$ preprocessing time, where n is the number of points in the data structure. The size of the bounding box for δ depends on d, and our results rely on the Random Access Machine (RAM) model with word size $\Theta(\lg n)$.

1 Introduction

The problem of finding the nearest neighbor to a query point is a fundamental data structure problem with numerous applications in areas such as computational geometry and machine learning. Unfortunately, finding the *exact* nearest neighbor seems difficult in dimensions 3 or higher as there is no known data structure that achieves nearly-linear preprocessing time and nearly-logarithmic query time. Hence, researchers turned to the approximate version of the problem, achieving significant performance gains by permitting the data structure to return merely a *near* neighbor, a point whose distance to the query is at most a constant times the distance from the nearest neighbor.

A large number of papers have sought to improve the performance of ANN data structures (see references in [2]), but none has shown how nonrandom patterns in query sequences might be exploited to improve running time. Examples of exploiting such nonrandomness abound in the 1D version of the exact nearest neighbor problem, for which data structures whose query performance depends upon the locality of queries in space and/or time have long been known (see references in [4]). Results in 2D, however, have only started to appear in recent years. For example, [12, 3] have shown how to exploit temporal locality in a random query sequence if the distribution is known, whereas [10, 13] have shown how to achieve dynamic-finger-like bounds in the 2D point search and point location problems.

Contribution. We extend ideas from [14, 9] to present the first ANN data structure to achieve, in any constant dimension, a provable speedup according to the degree of spatial locality in the query sequence. More specifically, in the RAM model with word size $w = \Theta(\lg n)$, we are given a set P of n points in d-dimensional space, each represented as a tuple of d words. We show how to preprocess P in $O(d^2 n \lg n)$ time to build an $O(dn)$-sized data structure that serves a sequence of queries for which each query q costs $O(d^2 \lg \delta(p,q))$, where p is the result of the preceding query and $\delta(p,q)$ is the number of input points in a suitably-sized box containing p and q. The L_t-norm approximation ratio is $O(d^{1+1/t})$, and the bounding box size for δ depends on d. While our description of the data structure does not include insertions or deletions, they are straightforward to support as long as spatially adaptive time bounds are not required.

Outline. Section 2 briefly discusses related work, including results in 1D on which we rely as a black box and Section 4 discusses the notion of "distance" we use in this paper in the context of other notions of distance that have been used by previous spatially adaptive data structures. We describe and analyze the data structure and the search algorithm in Sections 5 and 6 and we conclude in Section 7.

2 Related Work

Finger Search in 1D. There exists a variety of 1D nearest neighbor data structures that exploit spatial locality in a query sequence. Here we merely highlight two optimal results and we refer the reader to these two papers for references to previous works. Let q be the number of points between the previous and current queries. For Pointer Machines, Brodal *et al.* [6] have designed finger search trees with $O(\lg q)$ query time and $O(1)$ update time. With the added power of the RAM model, Andersson and Thorup [1] have shown how to achieve a query time of $O(\sqrt{\lg q / \lg \lg q})$. The running times above are all worst-case.

In this paper, we will be making use of a 1D finger search data structure as a black box. Any finger search data structure can be used as long as it does *not* restructure during a search. (This requirement will be explained in Section 5.) When updates are not required, we can simply use a sorted array as our black box and

*Department of Computer Science, Carnegie Mellon University, jonderry,dsheehy,sleator,maverick@cs.cmu.edu. This research was sponsored by the National Science Foundation under contract no. EIA-9706572, CCR-0122581, CNS-0435382 and CCF-0635257 and Appalachian Regional Commission under contract no. CO-14574.

perform finger search in the obvious manner. Only when updates are required do we need to employ more sophisticated data structures such as level-linked 2-3 trees by Brown and Tarjan [7] or several other finger search data structures that were designed later.

Finger Search in 2D. Though most of the work on spatially adaptive data structures is restricted to 1D, there has been some recent work on developing such distance-sensitive data structures in 2D. In particular, Demaine *et al.* [10] have shown how to preprocess a set of points P to permit a sequence of *membership* queries to points in P with a time bound of $O(\lg \delta_{PPS}(p,q))$, where $\delta_{PPS}(p,q)$ represents the distance between the previous query p and the current query q as measured by counting the number of points in a triangle-shaped region that contains both p and q. Subsequently, Iacono and Langerman [13] have shown how to achieve a similar distance-sensitive bound for point location in 2D.

Previous Work on ANN. The literature on the ANN problem is rich and we refer the reader to [11, Chapter 11] for numerous references. However, in the next section we will expand on the two previous works [14, 9] that are most relevant to this paper.

3 ANN and Space Filling Curves

Space filling curves (SFCs) provide a natural mapping from a high-dimensional space to an 1D curve and the ordering of points on SFCs has been used extensively as a meaningful order of points. In particular, the problem of finding ANNs and related proximity problems can be solved by SFC methods [14, 9]. Here we will describe a well-known algorithm for computing ANNs using SFCs from Liao *et al.* [14]. This algorithm is based on a similar algorithm that uses quadtrees. The quadtree version is due to Chan [8] and can be seen as a derandomization of an algorithm by Bern [5].

Let us consider a particular SFC known as the Z-order curve. Points are easily mapped onto this curve by a simple bit shuffling operation. Let $p_{i:j}$ represent the jth bit of the ith coordinate of point $p \in \mathbb{Z}^d$, assuming that each coordinate can be represented in a w-bit word. The shuffle operation $\sigma : \mathbb{Z}^d \to \mathbb{Z}$ is defined as the binary number $\sigma(p) = p_{1:w} \cdots p_{d:w} \cdots p_{1:1} \cdots p_{d:1}$, which we call the "shuffled value of p". For any pair of points p, q, we can order p and q on the curve by comparing their shuffled values. For a set of points $P = \{p_1, \ldots, p_n\}$, their Z-order is exactly their order in an in-order traversal of a quadtree constructed from the points in P. Figure 1 depicts this relationship and gives some intuition for the name "Z-order".

The algorithm for ANN is as follows. Observe that the Z-order depends on the placement of the origin and that for a particular Z-order, the nearest neighbor to a query is not necessarily the predecessor or the successor.

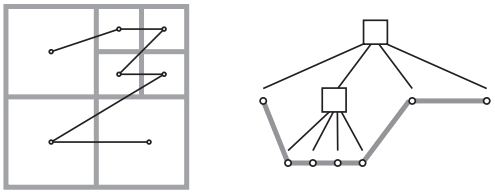


Figure 1: The in-order traversal of the quadtree leaves corresponds to the ordering of the points in Z-order.

Fortunately, one can show that there is a shift of the origin such that either the predecessor or the successor in the resulting Z-order is an ANN. In particular, consider a set of s shifts $v^{(j)} = (j/s, \ldots, j/s)$ for $j = 0, 1, \ldots, (s-1)$ and let s be $(d+1)$. Construct a set of search structures, one for each of the $(d+1)$ shifts. We compare p to q under the shift $v^{(j)}$ by comparing $\sigma(p+v^{(j)})$ to $\sigma(q+v^{(j)})$ and insert each input point into each of the structures. For a query q, do all $(d+1)$ searches for q and return the closest of the results. This algorithm gives an $O(d^{\frac{3}{2}})$ approximation in L_2 as shown in [8].

Chan's Comparison Procedure. We remark that in both the algorithm above as well as our algorithm in Section 5, we merely need the ability to compare the shuffled values. This can be done using a clever comparison procedure by Chan [9] that, given two points, compares their shuffled values using $O(d)$ exclusive-or word operations. This technique allows us to avoid computing and storing shuffled values at a cost of $O(d)$ slowdown and also mitigates the concern that a shuffled value may not fit inside a word.

4 Combinatorial Distance Measures for Point Sets

The goal of a geometric data structure supporting the dynamic finger property is to be distribution-sensitive so that sequences of geometrically close queries can be answered quickly. The ideal guarantee is that a query for a point q following a query for a point p takes time $O(\lg \mathsf{dist}(p,q))$ for some distance measure dist. For 1D problems, the distance between two points is simply the number of points between them. Unfortunately, such a combinatorial distance measure has no ready analogue in geometric spaces of dimension 2 or higher.

As the purpose of the finger p is to limit the search space to points that are geometrically close to p and q, a natural way to define a distance measure is to count the number of points in a suitable restriction of the search space. This intuition guided previous works in geometric finger search to use the notion of a region counting distance, in which $\mathsf{dist}(p,q)$ is defined as the number of input points in some carefully defined region containing both p and q [10, 13]. Formally, a region counting distance is defined by a triple (x, y, R) where x and y are points and R is a region whose membership

can be decided in $O(1)$ time. Given this triple, $\mathsf{dist}(p, q)$ is the number of points in the image of R under the affine transformation that takes x to p and y to q. The two previous works [10, 13] only applied to 2D where this transformation is unique.

Here we propose a new combinatorial distance measure similar to a region counting distance. Let c be a constant to be chosen later and let U be some axis-aligned box containing p and q with side length $c|p-q|_\infty$. The distance is defined as $\delta(p, q) = |P \cap U^*|$, where U^* is the choice among all such U that maximizes the distance measure. It should be clear that the points counted all have the desired property that their distances from p and q are bounded by a constant times the distance between p and q and therefore we have a distance measure that captures a notion of geometric locality in any dimension.

5 The Data Structure and Search Algorithm

Compared to the data structure in Section 3, our data structure uses $(2d+1)$ shifts versus $(d+1)$, and therefore consists of $(2d+1)$ 1D structures. Each of the 1D structures stores pointers to the input points in its nodes using the Z-ordering defined by the corresponding shift. Note that we do not store the keys, which are the shuffled values of the shifted points. Instead, we use Chan's comparison procedure on the points directly. For each input point x_i, we also maintain a circularly linked list comprising the $(2d+1)$ nodes that represent x_i in the 1D structures. The preprocessing time is $O(d^2 n \lg n)$ since there are $(2d+1)$ 1D structures to build, each requiring $O(n \lg n)$ comparisons that use $O(d)$ word operations each. The size of the data structure is $O(dn)$.

Given a query q, a search for an ANN is straightforward. Let p be the result of the previous query. We perform $(2d+1)$ finger searches from p in parallel, one for each shift. Let $x_1, \ldots, x_{d+1}$ be the results found by the first $(d+1)$ searches that complete. We return the x_i that is closest to q as an ANN and abandon the other d searches. Finally, we update the 1D structures to prepare for the next query by re-establishing their finger pointers to point at x_i using the circularly linked list associated with x_i. (The reason why we do not allow restructuring during a search in the 1D structures is to support the abandon and the re-establish steps.)

6 Algorithmic Guarantees

6.1 Centering Points in Quadtree Boxes

Chan [8] proved that $(d+1)$ shifts of the quadtree suffice to guarantee that for any point p and scale r, there is a shift that puts p roughly in the center of the quadtree square corresponding to that shift at scale r. This is the key lemma to prove that some quadtree will return an $O(d^{\frac{3}{2}})$-ANN.

For finger search to work, we need it to be true that for two different points, p, p' and two different scales r, r', there is a shift that puts p near the center of a square at scale r and p' near the center of a square at scale r'. Two guarantees rather than one are needed so that both the finger search will run quickly *and* the result will be a good approximation. The usual $(d+1)$ shifts would suffice if we were willing to accept only one of these guarantees, but we will show that $(2d+1)$ shifts suffice to get both.

To maintain consistency with the work we are extending, we assume the input points are scaled to finite precision real numbers in $[0, 1)^d$.

Say that p is α-*central* at scale r if for all $i = 1, \ldots, d$, we have $(p_i + \alpha) \bmod r \geq 2r\alpha$. The following is a slight extension of [8, Lemma 3.3] and its proof follows the same pattern as the original.

Lemma 1 *Let $s > d$ be an odd integer representing the number of shifts $v^{(j)} = (\frac{j}{s}, \ldots, \frac{j}{s})$, $j = 0, \ldots, (s-1)$. For a point $p \in [0, 1)^d$, and scale $r = 2^{-\ell}$, there are at most d shifts $v^{(j)}$ such that $p + v^{(j)}$ is not $\frac{1}{2s}$-central at scale r.*

Proof. We will prove that at most one shift is bad for each dimension. Formally, we prove that for each $i \in \{1, \ldots, d\}$, there is at most one shift $v^{(j)}$ such that

$$\left(p_i + \frac{j}{s} + \frac{1}{2s}\right) \bmod r < \frac{r}{s}, \tag{1}$$

or equivalently, by multiplying through by s/r,

$$\left(2^\ell s p_i + 2^\ell j + 2^{\ell-1}\right) \bmod s < 1. \tag{2}$$

Suppose on the contrary that we have distinct $j, j' \in \{0, \ldots, s-1\}$ for which (2) holds. Letting $z = 2^\ell s p_i + 2^{\ell-1}$, we have $(z + 2^\ell j) \bmod s < 1$ and also $(z + 2^\ell j') \bmod s < 1$. So, for integers q, q' and remainders $0 \leq x, x' < 1$, the above inequalities imply $z + 2^\ell j = qs + x$, and $z + 2^\ell j' = q's + x'$. It follows that $2^\ell(j - j') - (q - q')s = x - x'$. Since the LHS here is an integer and $0 \leq x, x' < 1$, it must be that in fact $x = x'$ and thus $2^\ell j \equiv 2^\ell j' \pmod{s}$. We can divide both sides of this congruence by 2^ℓ because 2^ℓ and s are relatively prime (s is odd). The result is $j = j'$, a contradiction. □

6.2 Query Time and Approximation Ratio

To analyze query time we must first choose the constant for our distance measure. Say $\delta(p, q) = |\{x \in P : |x - p|_\infty \leq (8d+4)|p-q|_\infty\}|$.

Using Lemma 1 for a scale r, we know that of the $(2d+1)$ shifts, p is $\frac{1}{4d+2}$-central in at least $(d+1)$ shifts. In particular, we are interested in the smallest scale $r \geq (4d+2)|p-q|_\infty$. At this scale, p has distance at least $|p-q|_\infty$ from the boundary of any box B_i for which it is $\frac{1}{4d+2}$-central. So q is also in each of these $(d+1)$ boxes B_i. The SFC touches each point in a box B before leaving, so each finger search will take time $O(d \lg |P \cap B_i|)$. As all points in $P \cap B_i$ are counted in

$\delta(p,q)$, we see that $(d+1)$ different shifts are guaranteed to finish in $O(d^2 \lg \delta(p,q))$ time. Choosing the best of these $(d+1)$ answers can be done in $O(d^2)$ time. Thus, the total running time is $O(d^2 \lg \delta(p,q))$.

Second, we need to show that the returned point is indeed a good ANN and this also follows from Lemma 1. Let q^* be the nearest neighbor of q. The lemma implies that at the smallest scale $r' \geq (4d+2)|q-q^*|_\infty$, there can be at most d shifts for which q is not $\frac{1}{4d+2}$-central. Therefore one of the $(d+1)$ shifts that finished searching found q in a box for which it is central at scale r'. The search returned a point x in that box, and thus $|q-x|_\infty < (8d+4)|q-q^*|_\infty$. So, x is an $O(d)$-ANN in the L_∞ norm and therefore an $O(d^{1+1/t})$-ANN in the L_t norm.

7 Concluding Remarks

In this paper, we showed how to achieve spatial adaptivity for the ANN problem in any constant dimension by extending prior work based on SFCs. Here we describe an enhancement and discuss some future research.

Using the Quadtree to Speed Up Search. Recall that the Z-ordering of the input points corresponds to the in-order traversal of the leaves in a quadtree. For any two points p, q in a quadtree, there is a unique path along the link structure. Let us call the length of this path the *quadtree distance*. The quadtree distance approximates the log of the Euclidean distance after normalization by the empty space around p and q. One can imagine building a quadtree that supports finger search in time proportional to the quadtree distance by walking up and down within the quadtree.

However, observe that even when using a *compressed quadtree*, in which paths of degree two nodes of the tree are collapsed, the quadtree distance may still be linear in the number of input points. Furthermore, even with a good shift, this distance could still be significantly worse than the distance computed by the measure in Section 4. On the other hand, it is also not hard to construct examples in which the quadtree distance is $o(\lg \delta(p,q))$. As an example, consider the effect of adding a dense set of points between p and q while p and q each remains in its own, relatively sparse region. This example not only shows that there is no strict ordering of geometric and combinatorial distance measures, but it also suggests that one can exploit using *both* the structure of the quadtree and our data structure to speed up searches.

Future Work. A good enhancement to make in the future would be to improve the approximation ratio to $(1+\epsilon)$, though it is not clear how to do this without the exponential blowup incurred by analogous enhancements to other SFC-based data structures for ANN. A more modest enhancement would be to shrink the distance measure so that the distance between two successive queries to p and q would be the number of points inside a smaller box that more tightly bounds p and q. We can achieve this to a degree by using a constant number of shifts *independently* in each dimension at the expense of an exponential factor in d to space usage, but perhaps there other methods. We may also try to more thoroughly exploit the power of RAM. For example, if we use the finger search structure by Andersson and Thorup [1] as the 1D structure, then we get an improved running time. However, we do not know how to avoid computing the shuffled values explicitly when using this. Finally, a general direction for future work would be to extend other ANN techniques—or even algorithms for serving exact nearest neighbor queries in high dimensions—to allow spatial adaptivity, temporal adaptivity, or a combination of the two.

References

[1] A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):Article 13, 2007.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[3] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Transactions on Algorithms*, 3(2):Article 17, 2007.

[4] M. Badoiu, R. Cole, E. D. Demaine, and J. Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theoretical Computer Science*, 382(2):86–96, 2007.

[5] M. Bern. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45(2):95–99, 1993.

[6] G. S. Brodal, G. Lagogiannis, C. Makris, A. K. Tsakalidis, and K. Tsichlas. Optimal finger search trees in the pointer machine. *Journal of Computer and System Sciences*, 67(2):381–418, 2003.

[7] M. R. Brown and R. E. Tarjan. Design and analysis of a data structure for representing sorted lists. *SIAM Journal of Computing*, 9(3):594–614, 1980.

[8] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, 1998.

[9] T. M. Chan. Closest-point problems simplified on the RAM. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 472–473, 2002.

[10] E. D. Demaine, J. Iacono, and S. Langerman. Proximate point searching. *Computational Geometry*, 28(1):29–40, 2008.

[11] S. Har-Peled. *Geometric Approximation Algorithms*. (working draft), 2008.

[12] J. Iacono. Optimal planar point location. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 340–341, 2001.

[13] J. Iacono and S. Langerman. Proximate planar point location. In *Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 220–226, 2003.

[14] S. Liao, M. A. Lopez, and S. T. Leutenegger. High dimensional similarity search with space filling curves. In *17th International Conference on Data Engineering*, pages 615–622, 2001.

Smallest enclosing circle centered on a query line segment

Prosenjit Bose [*], Stefan Langerman[†], and Sasanka Roy [‡]

Abstract

Given a set of n points $P = \{p_1, p_2, \ldots, p_n\}$ in the plane, we show how to preprocess P such that for any query line segment L we can report in $O(\log n)$ time the smallest enclosing circle whose center is constrained to lie on L . The preprocessing time and space complexity are $O(n \log n)$ and $O(n)$ respectively. We then show how to use this data structure in order to compute the smallest enclosing circle of P whose center is restricted to lie in one of several polygons having a total of m edges, in $O((m + n) \log n)$ time, a significant improvement over previous known algorithms.

1 Introduction

The problem of computing the smallest enclosing circle of a set P of n points in the plane was originally posed in 1857 by Sylvester [11]. Many solutions have appeared in the literature (see [9] or [8] for a brief history of the problem) culminating in the optimal linear time algorithm by Megiddo [7].

In recent years, several constrained variants of this problem have been studied, where restrictions are placed on the location of the center of the smallest enclosing disk. Already in his original paper and as a step towards the general solution, Megiddo [7] studied the situation where the center of the smallest enclosing circle is restricted to lie on a given straight line. Hurtado et al. [4] generalized Megiddo's technique to provide an $O(n + m)$ time algorithm for finding smallest enclosing circle whose center is constrained to lie in the intersection of m linear inequalities.

Bose et al. [1] considered a generalized setting of the problem where the center of the smallest enclosing circle of P is constrained to lie inside a simple polygon of size m. Their algorithm runs in $O((n + m) \log(n + m) + k)$ time, where k is the number of intersections of the boundary of the polygon with the farthest point Voronoi diagram of P. In the worst case, k may be $O(n^2)$. This result was later improved to $O((n + m) \log m)$ by Bose and Wang [2]. In a further generalization of this problem, where r (≥ 1) simple polygons with a total of m vertices are given, locating the center of the smallest enclosing circle of P with its center inside one of the given polygons was studied by Bose and Wang [2]. The time complexity of this version of the problem is $O((m + n) \log n + (n\sqrt{r} + m) \log m + m\sqrt{r} + r^{\frac{3}{2}} \log r)$ and which has further improved to $O(n \log n + m \log^2 n)$ by Roy et al. [10].

The query version of the smallest enclosing circle ($QSEC$) where the center is constrained to lie on a query line was originally posed by Roy et al. [10]. The preprocessing time and space complexity of their algorithm is $O(n \log n)$ and $O(n)$ respectively. The center of the minimum enclosing circle can be reported in $O(\log^2 n)$ time. Very recently, Karmakar et al. [5] proposed an optimal $O(\log n)$ query time algorithm for the query version of the problem. However, the improved query time comes at an increased cost in both preprocessing time and space. The preprocessing time and space complexity for their algorithm is $O(n^2)$.

In this paper we show how to achieve an optimal query time of $O(\log n)$ for the query version of the problem with $O(n \log n)$ preprocessing time and $O(n)$ space. Using our result, we show how to find the smallest enclosing circle where the center is restricted to lie in a set of polygons with a total of m vertices, in $O((m+n) \log n)$ time. This is a significant improvement over the previous best algorithm.

2 Preliminaries

Given a set P of n points in the plane, the only points that can be on the boundary of any enclosing circle lie on the convex hull of P (because a disk is convex and by the definition of the convex hull). Thus, we will assume that the points $P = \{p_1, p_2, \ldots, p_n\}$ are in convex position.

Let $V(P)$ denote the furthest point Voronoi diagram of P (see [8] for a survey on Voronoi diagrams and their furthest point counter-part). The diagram $V(P)$ partitions the plane into n unbounded convex regions, denoted $R(p_1), R(p_2), \ldots, R(p_n)$, such that for any point $p \in R(p_j)$, $d(p, p_j) \geq d(p, p_k)$ for all $k = 1, 2, \ldots, n$, and $k \neq j$. Here, $d(.,.)$ denotes the Euclidean distance between a pair of points. This structure can be computed in $O(n\log n)$ time and $O(n)$ space (see [9]). Furthest point Voronoi diagrams play an important role when studying enclosing disks, because of the following.

[*]Carleton University, Ottawa, Canada, jit@scs.carleton.ca

[†]Chercheur qualifié du FRS-F.N.R.S., Université Libre de Bruxelles, Brussels, Belgium, stefan.langerman@ulb.ac.be

[‡]Tata Consultancy Services Ltd., Pune, India, sasanka.roy@tcs.com

Lemma 1 *[6] The smallest (unconstrained) enclosing circle of a set P of points in the plane always has at least two points of P on its boundary.*

Lemma 1 implies that the center c of the smallest enclosing circle of P always lies on an edge e of $V(P)$. It is this property that allows for the discretization of the problem.

In this paper, for simplicity of exposition, we first assume that the points are in general position (i.e. no four points in P are co-circular). We then show how a minor modification to our solution allows the removal of this general position assumption. We begin by describing how our data structure answers queries when the query object is a line. Then, we show how the algorithm works for query line segments.

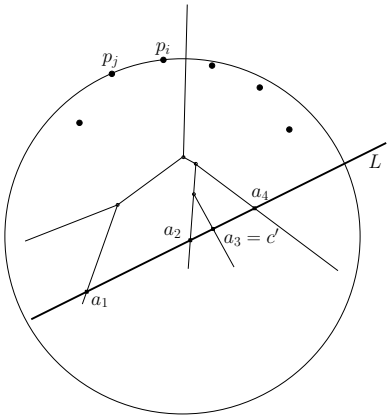


Figure 1: $\rho(a_i)$ *values are unimodal*

It has been shown in [10] how to solve the query problem in $O(\log n)$ time with $O(n \log n)$ preprocessing time and $O(n)$ space when the solution has exactly one point of P on its boundary. If the smallest enclosing circle with center c' constrained to lie on a line segment L has exactly one point of P on its boundary, then c' is either the orthogonal projection of the furthest point to L onto L, or c' must lie on an endpoint of L. Thus, the solution can be found in $O(\log n)$ time by building a point location structure on top of the furthest point Voronoi diagram. The main difficulty is to solve the query problem when the smallest enclosing circle has more than one point of P on its boundary. We present a solution to this problem in the next section.

Lemma 2 *[10] If the smallest enclosing circle with center c' constrained to lie on a segment L has more than one point of P on its boundary, then c' lies on an intersection point of L with an edge of $V(P)$.*

Note that in a degenerate case, c' may be a vertex of $V(P)$ and in this case we can say that c' coincides with the end points of the edge of $V(P)$.

Let $\rho(q)$ denote the radius of the smallest enclosing circle of P with center at point q. Note that if $q \in R(p_i)$ then p_i is on the boundary of the smallest enclosing circle centered at q. Our algorithms will heavily rely on the following:

Lemma 3 *The function $\rho(q)$ is convex.*

Proof. The value of $\rho(q)$ is the maximum distance of q to the points of P, thus $\rho(q)$ is the upper envelope of a set of cones. So it is convex. □

In particular this implies that the restriction of $\rho(q)$ to some line L is a convex function as well.

3 The Data Structure

We now have all the tools to describe our method. Recall our initial assumption that no four points in P are co-circular. Given this assumption, we note that $V(P)$ is a binary tree, denoted T. Let $|T|$ denote the number of vertices of T. Each edge e of T separates two unbounded Voronoi cells. Note that from any point inside an unbounded cell, any ray in an unbounded direction will be entirely contained in the cell. We augment the tree T by associating with each edge such a ray from the midpoint of e for each of the two adjacent cells.

The removal of a Voronoi edge $e = (a, b)$ would split T into two subtrees which we denote by T_a and T_b where T_a is the subtree that contains a and T_b the subtree containing b. The edge e is called a *centroid edge* if T_a and T_b each contain no more than $(2/3)|T|$ vertices. For any binary tree, a centroid edge is known to exist and can be found in linear time [3].

A *centroid decomposition* of T is a binary tree whose nodes are associated with edges of T, whose root is a centroid edge $e = (a, b)$ and whose two subtrees are recursively defined as centroid decompositions of T_a and T_b. It is known that a centroid decomposition of any binary tree T with n vertices has depth $O(\log n)$ and can be constructed in $O(n)$ time [3].

The data structure will be composed of a centroid decomposition of T (see Figure 2) augmented with the rays as described above, and a point location data structure for $V(P)$.

Lemma 4 *The above preprocessing algorithm requires $O(n)$ time and space for a given $V(P)$.*

Lemma 5 *Using the above structure, given a query line L and an edge $e = (a, b)$ of T, we can determine in $O(1)$ time whether the smallest enclosing circle with center constrained to lie on L has its center in e, T_a or T_b.*

Proof. Assume the Voronoi edge $e = (a, b)$ separates the two Voronoi regions $R(p_i)$ and $R(p_j)$. Let ℓ_1 and ℓ_2 be the two rays associated with e. Note that for any

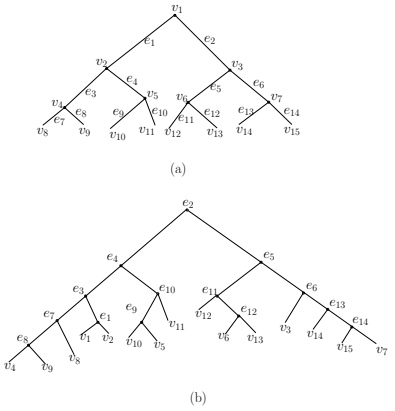


Figure 2: *(a) Binary tree T and (b) Centroid decomposition of T*

point q on ℓ_1 or ℓ_2, we know the cell that contains q, and so which is the furthest point to q. Therefore we can compute the value and the gradient of $\rho(q)$ in $O(1)$ time.

The two rays ℓ_1 and ℓ_2 divide the plane into two regions and the two subtrees T_a and T_b are each wholly contained in one of these regions. Let A be the region containing T_a and B the region containing T_b. The query line L may have three different types of intersections with ℓ_1 and ℓ_2, which form the basis of our case analysis:

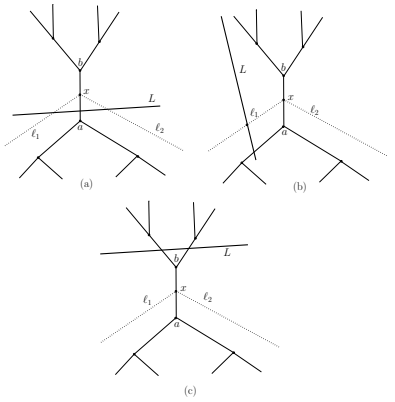


Figure 3: *Illustration of pruning*

Case 1. L intersects both rays ℓ_1 and ℓ_2 (See Figure 3(a))

Case 2. L intersects only one ray, say ℓ_1 (See Figure 3(b))

Case 3. L intersects neither of the two rays (See Figure 3(c))

For each of the three cases, we show how to eliminate one of the subtrees T_a or T_b from the search.

Case 1: Let y and z be the intersection of L with ℓ_1 and ℓ_2, respectively (See Figure 3). By determining the value and the gradient of ρ at y and at z, we can determine if the answer lies on the segment $[yz]$ or if it lies on L outside $[yz]$, since by Lemma 5, we know that the function is convex. If the solution lies on $[yz]$, then the solution lies in $T \backslash T_b$, otherwise it lies on $T \setminus T_a$.

Case 2. Without loss of generality let L intersect ℓ_1 at y (See Figure 3(b)). Again, by finding the value and the gradient of ρ at y, and by Lemma 5, we can determine if the solution lies in A or B. If the solution lies in A then we know it lies in $T \setminus T_b$. Otherwise, it lies in $T \setminus T_a$.

Case 3. If L does not intersect ℓ_1 and ℓ_2 (See Figure 3(c)), then L lies completely inside A or B. We can discard the sub-tree of the region that does not intersect the line L. In the Figure 3(c) the solution lies in $T \setminus T_a$.

□

Lemma 6 *Using the above preprocessed data structure the QSEC problem can be solved in $O(\log n)$ time.*

Proof. In the worst case we may have traverse the worst case depth of centroid decomposition tree which is $O(\log n)$. Each step in this traversal costs $O(1)$ time. Hence the query time complexity result follows. □

4 $QSEC$ for Query Line Segment

Here, the query object $L = [f, g]$ is a line segment. We first solve $QSEC$ for the query line $\overline{L}$ that contains the line segment L. Let α be the center of the $QSEC$ for line $\overline{L}$. If α lies inside $[f, g]$, then report α. Otherwise, by Lemma 5, the center c' of the desired constrained smallest enclosing circle is one of the endpoints f or g, which is closest to α. Let f be the closest point of α. Then $\alpha = c'$ and the radius of the desired smallest enclosing circle is $d(p, c')$, where $p \in P$ is the point whose corresponding Voronoi cell contains c'. The Voronoi cell that contains α can be found in $O(\log n)$ time. Thus $QSEC$ for the query line segment can be solved in $O(\log n)$ time.

5 Solution when P is in general position

When P is in general position then T may not be a binary tree. So, Lemma 4 is not true anymore. Now we will describe a method to split the nodes of T whose degree are greater than 3 by adding some virtual edges and construct a virtual binary tree VT. We will show that the number of edges thus inserted is no more than $O(n)$. Then it is easy to observe that solving $QSEC$ problem in VT is same as solving this problem for T and $QSEC$ of VT is $QSEC$ of T. Let us consider a

vertex v of T with degree greater than 3. For simplicity, the degree of v is $k > 3$.

We will add $k-1$ virtual edges as follows to make it a binary tree of size $O(k)$:

Let $(e_1, e_2, \ldots, e_k)$ (see Figure 4(a)) be the Voronoi edges adjacent to vertex v. We will insert $k-1$ virtual edges $(ve_1, ve_2, \ldots, ve_{k-1})$ between the pair of edges $(e_1, e_2, \ldots, e_k)$ (see Figure 4(b)). The edge ve_j keep the coordinates of the vertex v and the equation of Voronoi edges e_j and e_{j+1}. Intuitively, the implication of virtual edge ve_j is that we can always draw two rays such that one half-plane, say HP_1, contains the sub-tree that has edges $e_1, e_2, \ldots, e_j$ and has root at v. Other half-plane HP_2 contains the sub-tree that contains the edges $e_{j+1}, e_{j+2}, \ldots, e_k$ and has root at v.

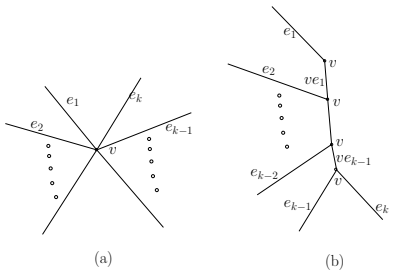


Figure 4: *Splitting a vertex of degree greater than* 3

6 Constrained Smallest Enclosing Circle Problem with Center in a given Set of Polygons

Now for the problem where we have to find the center inside r simple polygons with a total of m edges. Compute the farthest point Voronoi diagram $V(P)$ and identify the center c' of the unconstrained smallest enclosing circle. If it is inside one of these polygons, we report the answer. Otherwise, the center will be on the boundary of one of these polygons. For each edge (line segment), we compute the center of the constrained smallest enclosing circle with center on that edge, and report the radius of the smallest one. Thus, the overall time complexity becomes $O((n+m)\log n)$, where $|P| = n$. Thus it improves the previous running time proposed Roy et al. [10]. There may exist more than one such circle attaining the smallest radius [1]. We can report all these circles with the same time complexity.

References

[1] P. Bose and G. Toussaint, *Computing the constrained euclidean, geodesic and link center of a simple polygon with applications*, Proc. of the Pacific Graphics International, 1996, pp. 102–112.

[2] P. Bose and Q. Wang, *Facility location constrained to a polygonal domain*, Proc. of the Latin American Theoretical Informatics Symposium, 2002, pp. 153–164.

[3] G. N. Frederickson and D.B. Johnson, *Generating and searching sets induced by networks*, Proc. of the 7th International Colloquium on Automata, Languages and Programming, 1980, pp. 221–233.

[4] F. Hurtado, V. Sacristan, and G. Toussaint, *Facility location problems with constraints*, Studies in Locational Analysis **15** (2000), 17–35.

[5] A. Karmakar, S. Roy, and S. Das, *Fast computation of smallest enclosing circle with center on a query line segment*, Proc. of the Canadian conference on Computational Geometry, 2007, pp. 273–276.

[6] D. T. Lee, *Furthest neighbor voronoi diagrams and applications*, Report 80-11-FC-04, Dept. Elect. Engrg. Comput. Sci., Northwestern Univ., Evanston, IL, 1980.

[7] N. Megiddo, *Linear-time algorithms for linear programming in r^3 and related problems*, SIAM Journal on Computing **12** (1983), 759–776.

[8] A. Okabe, B. Boots, K. Sugihara, and S. Chiu, *Spatial tessellations: Concepts and applications of voronoi diagrams*, 2000.

[9] F. P. Preparata and M. I. Shamos, *Computational geometry: An introduction*, 1990.

[10] S. Roy, A. Karmakar, S. Das, and S. C. Nandy, *Constrained minimum enclosing circle with center on a query line segment*, Proc. Mathematical Foundation of Computer Science, 2006, pp. 765–776.

[11] J. J. Sylvester, *A question in the geometry of situation*, Quarterly Journal of Mathematics (1857), 1–79.

On a Cone Covering Problem

Khaled Elbassioni* elbassio@mpi-sb.mpg.de

Hans Raj Tiwary† hansraj@cs.uni-sb.de

Abstract

Given a set of polyhedral cones $\mathcal{C}_1, \cdots, \mathcal{C}_k \subset \mathbb{R}^d$, and a convex set D, does the union of these cones cover the set D? In this paper we consider the computational complexity of this problem for various cases such as whether the cones are defined by extreme rays or facets, and whether D is entire $\mathbb{R}^d$ or an affine subspace $\mathbb{R}^t$. As a consequence, we show that the problem of checking if the union of a given set of convex polytopes is convex is coNP-complete, thus answering a question of Bemporad et al. [3].

1 Introduction

Let $S \subseteq \mathbb{R}^d$ be a set of points in $\mathbb{R}^d$. The *conic hull* of S, denoted by cone(S), is the set of all non-negative linear combinations of points in S, i.e., cone(S) $= \{\sum_{p\in S} \mu_p p : \ \mu_p \geq 0 \text{ for all } p \in S\}$. It is well-known that any polyhedral cone cone(S) can be written equivalently as the intersection of finitely many half-spaces, i.e., cone(S) $= \{x \in \mathbb{R}^d : \ Ax \leq 0\}$, where $A \in \mathbb{R}^{m\times d}$. The two representations are called the $\mathcal{V}$- and the $\mathcal{H}$-representations, respectively.

In this note we are interested in the complexity of covering problems of the following form:

ConeCover($\mathcal{C}, D$): Given a collection of cones $\mathcal{C} = C_1, \ldots, C_k$, and a convex set D, does $\bigcup_{i=1}^k C_i \not\supseteq D$?

A polytope P is the convex hull of a finite set S of points in $\mathbb{R}^d$, and it can also be written in one of two equivalent forms: $P = \text{conv}(S) = \{\sum_{p\in S} \mu_p p : \ \mu_p \geq 0 \text{ for all } p \in S, \ \sum_{p\in S} \mu_p = 1\}$ ($\mathcal{V}$-representation), or $P = \{x \in \mathbb{R}^d | Ax \leq \mathbf{1}\}$, where $\mathbf{1}$ is the vector in which each component is 1 ($\mathcal{H}$-representation)[1]. A polyhedron Q is the Minkowski sum of a polytope P and a cone C: $Q = P + C \stackrel{\text{def}}{=} \{x+y | x \in P, \text{ and } y \in C\}$. Similarly, one can also consider the problem PolytopeCover($\mathcal{P}, D$): Given a collection of polytopes $\mathcal{P} = P_1, \ldots, P_k$, and a convex polytope D, does $\bigcup_{i=1}^k P_i \not\supseteq D$?

Our motivation for studying the above covering problems comes from two other related problems on polytopes. The first is the well-known *vertex-enumeration* problem of finding the vertices of a polytope given its facet defining inequalities, to be described in more details in the next section. The second problem is to check whether the union of a given set of polytopes is convex. Bemporad, Fukuda and Torrisi [3] gave polynomial-time algorithms for checking if the union of $k = 2$ polyhedra is convex, and if so finding this union, no matter whether they are given in $\mathcal{V}$ or $\mathcal{H}$ representations. They also gave necessary and sufficient conditions for the union of a finite number of convex polytopes in $\mathbb{R}^d$ to be convex, and asked whether these conditions can be used to design a polynomial time algorithm for checking if the union is convex. Bárány and Fukuda give slightly stronger conditions in [2]. It will follow from our results that, if both d and k are part of the input, then these conditions can not be checked in polynomial time unless P=NP.

Unless otherwise specified, all the cones considered throughout the paper will be assumed to be pointed, i.e., contain no lines, or equivalently, have a well defined apex, namely the origin. As we shall see, the complexity of the above problem depends on how the cones are represented, and whether they are disjoint or not. We consider 3 different factors, namely:

(f1) whether the cones in $\mathcal{C}$ are given in $\mathcal{V}$- or $\mathcal{H}$-representations, or both representations ($\mathcal{VH}$),

(f2) what the set D is: we consider $D = \mathbb{R}^d$ and $D = \mathbb{R}^t$ for some arbitrary $k \leq d$.

(f3) whether the cones in $\mathcal{C}$ are

- (f3)-(I): pairwise disjoint in the interior and intersect only at faces;
- (f3)-(II): pairwise disjoint in the interior , but can intersect anywhere on the boundaries; and
- (f3)-(III): not necessarily pairwise disjoint.

We denote by ConeCover[$F1, F2, F3$] the different variants of the problem, where $F1 \in \{\mathcal{V}, \mathcal{H}\}$, $F2 \in \{\mathbb{R}^t, \mathbb{R}^d\}$ and $F3 \in \{I, II, III\}$ describes cases (f1)-(I), (f2)-(II), and (f3)-(III). Our results are summarized in Table 1.

*Max Planck Institut für Informatik, Saarbrücken, D-66123 Germany

†Universität des Saarlandes, Saarbrücken, D-66123 Germany

[1] possibly after moving first the polytope so that its relative interior contains the origin

	$\mathbb{R}^d$			$\mathbb{R}^t$		
	I	II	III	I	II	III
$\mathcal{V}$	VE-hard	VE-hard	NPC	NPC	NPC	NPC
$\mathcal{H}$	P	?	NPC	P	?	NPC
$\mathcal{VH}$	P	P	NPC	P	?	NPC

Table 1: Complexity of Cone Covering problem for various input representations.

2 Results

Converting the $\mathcal{H}$-representation of a polytope to its $\mathcal{V}$-representation and vice versa, is a well studied problem. Despite years of research, it is neither known if an output-sensitive algorithm exists for this problem, nor is it known to be NP-hard. The following decision version of this problem is known to be equivalent to the enumeration problem [1].

VertEnum(P, V): Given an $\mathcal{H}$-polytope $P \subseteq \mathbb{R}^d$ and a subset of its vertices $V \subseteq \mathcal{V}(P)$, check whether $P = \text{conv}(V)$.

Let P be the polytope defined as $\{x|Ax \leq \mathbf{1}\}$, where $A \in \mathbb{R}^{m \times d}$. Every rational polytope can be brought into this form by moving the origin in its relative interior and scaling the normals of the facet-defining hyperplanes appropriately. For any vertex v of P, consider the cone of all vectors c such that v is the solution of the following linear program: $\max c^T x$ s.t. $Ax \leq 1$. For every vertex v of P, this cone is uniquely defined. We call this cone the *maximizer cone* of v. Such a maximizer cone can be defined for every proper face of a polytope. The union of all such cones is also known as the *normal fan* of a polytope [8]. It is easy to see that if A' is the maximal subset of rows of A such that $A'v = 1$, then the maximizer cone of v is the conic hull of the rows of A' treated as vectors in $\mathbb{R}^d$.

Theorem 1 *Problem* ConeCover$[\mathcal{V}, \mathbb{R}^d, I]$ *is* VertEnum*-hard.*

Proof. Given an $\mathcal{H}$-polytope P and a subset of its vertices V, the $\mathcal{V}$-representation of the maximizer cone for each vertex in V can be computed easily from the facets of P. Clearly, the union of these cones covers $\mathbb{R}^d$ if and only if $P = conv(V)$. To see this, note that if $P \neq conv(V)$ then P has a vertex v not in V and any vector in the relative interior of the maximizer cone of v does not lie in any of the cones corresponding to the given vertices. □

Theorem 2 *Problem* ConeCover$[\mathcal{V}, \mathbb{R}^t, I]$ *is NP-complete.*

Proof. ConeCover$[\mathcal{V}, \mathbb{R}^t, \text{I}]$ is clearly in NP. Now, given an $\mathcal{H}$-polytope $P \subset \mathbb{R}^d$, an affine subspace $\mathbb{R}^t$ and a $\mathcal{V}$-polytope $Q \subset \mathbb{R}^k$, it is NP-complete to decide whether Q is the projection of P onto the given subspace [7]. We give a polynomial reduction from this problem to ConeCover$[\mathcal{V}, \mathbb{R}^t, \text{I}]$.

Every vertex v of Q is an image of some (possibly more than one) vertices of P. If this is not the case then Q clearly can not be the projection of P. Since the vertices of Q are known this condition can be checked in polynomial time. To see why this is true, consider a vertex v of Q and choose any direction α in the affine space of Q such that $\alpha^T x$ is maximized at v for all points in Q. If we use the same vector α as objective function over the points in P then the maximum is achieved at the face containing all vertices whose image under projection is v.

Now, Pick any such vertex and call it v'. We associate the maximizer cone of v' with v and refer to it as $\mathcal{C}(v)$. The $\mathcal{V}$-representation of $\mathcal{C}(v)$ for every vertex v of Q can be easily computed from the matrix A of the normals of facet defining hyperplanes of P.

It is not difficult to see that if Q is not the projection of P onto the given subspace $\mathbb{R}^t$, then one can find a direction c parallel to the given subspace such that a vertex that maximizes $c^T x$ in P is such that its projection is a vertex of the projection of P but not of Q. Hence, the union of cones $\mathcal{C}(v)$ for each vertex v of Q covers $\mathbb{R}^t$ if and only if Q is the projection of P. Also, all these cones intersect each other only at some proper face. □

For a given set of $\mathcal{H}$-cones, if the union does not cover $\mathbb{R}^d$ then there is a facet with normal $a \in \mathbb{R}^d$, of at least one of these cones such that picking a point p in the interior of this facet, $p + \epsilon a$ lies outside every cone, for some $\epsilon > 0$. Let us call this facet a *witness facet*, and p a *witness point* of the fact that $\mathbb{R}^d$ is not covered.

Theorem 3 *There is a polynomial time algorithm for solving* ConeCover$[\mathcal{H}, \mathbb{R}^d, I]$.

Proof. If the cones are allowed to intersect only at common faces, then every point in the interior of a witness facet is a witness point. Thus, one can determine in polynomial time whether the union of the given cones cover $\mathbb{R}^d$ or not, by picking a point in the interior of every facet, with normal a, of every cone and using linear programming to check if $p + \epsilon a$ lies outside every cone for sufficiently small $\epsilon > 0$. □

Theorem 4 *There is a polynomial time algorithm for* ConeCover$[\mathcal{VH}, \mathbb{R}^d, II]$.

Proof. It is easy to see that if the cones are allowed to intersect only on the boundary, and if the union of the given cones does not cover $\mathbb{R}^d$, then the extreme rays[2] of

[2]formally, an extreme ray of a hole in $\mathbb{R}^d$ is defined by $d-1$ linearly independent facets defining the boundary of the hole, whose intersection lies inside the hole

any (possibly non-convex) "hole" are also the extreme rays of some cone. For any such extreme ray w, if one considers a d-dimensional ball of radius ϵ centered at some point on w, then for small enough ϵ some part of this ball is not covered by any of the given cones.

Consider all the halfspaces $\{x|\ ax \leq 0\}$ corresponding to the facets of the input cones that contain w, *i.e.,* $aw = 0$. Let A be the matrix with each row the normal vector of such a halfspace. The union of the given cones does not cover $\mathbb{R}^d$ if and only if $\{x|\ Ax \geq 0\}$ defines a full-dimensional region. This can be easily checked via linear programming. □

Fact 1 *For any $t \in \mathbb{N}$, we can write $\mathbb{R}^t = \cup_{i=1}^{k+1} R_i$, where $R_1, \ldots, R_{k+1}$ are pointed cones, pairwise-disjoint in the interior, whose $\mathcal{H}$- and $\mathcal{V}$-representations can be found in in polynomial time.*

Let $C_1 = \{x \in \mathbb{R}^m \mid A_1 x \leq \mathbf{0}\} = \text{cone}(S_1)$ and $C_2 = \{x \in \mathbb{R}^n \mid A_2 x \leq \mathbf{0}\} = \text{cone}(S_2)$, where $A_1 \in \mathbb{R}^{l \times m}, A_2 \in \mathbb{R}^{r \times n}$ and $S_1 \subseteq \mathbb{R}^m, S_2 \subseteq \mathbb{R}^n$, be two polyhedral cones. The *direct-sum* of C_1 and C_2, is defined as:

$$\begin{aligned} C_1 \oplus C_2 &= \{(x,y) \in \mathbb{R}^m \times \mathbb{R}^n |\ A_1 x \leq \mathbf{0},\ A_2 y \leq \mathbf{0}\} \\ &= \text{cone}\left(\left\{\begin{pmatrix} v \\ \mathbf{0} \end{pmatrix} : v \in S_1\right\} \bigcup \left\{\begin{pmatrix} \mathbf{0} \\ v \end{pmatrix} : v \in S_2\right\}\right) \end{aligned}$$

Theorem 5 *Problem* ConeCover$[\mathcal{VH}, \mathbb{R}^d, III]$ *is NP-complete.*

Proof. Clearly the problem is in NP since a direction exists outside the union of the given cones if they do not cover $\mathbb{R}^d$. We can easily check if such a given direction indeed lies outside each of the cones since the facets of each cone are known. For proving its NP-hardness, we use a reduction from the following problem:

Sat$(V, \mathcal{F}, \mathcal{G})$: Given a finite set V and two hypergraphs $\mathcal{F}, \mathcal{G} \subseteq 2^V$, is there a set $X \subseteq V$ such that:

$$X \not\supseteq F \text{ for all } F \in \mathcal{F} \text{ and } X \not\subseteq G \text{ for all } G \in \mathcal{G}. \quad (1)$$

When $\mathcal{F} = \mathcal{G}$, this problem is called the *saturation problem* in [4], where it is proved to be NP-complete. Given $\mathcal{F}, \mathcal{G} \subseteq 2^V$, we construct two families of cones $\mathcal{C}_\mathcal{F}$ and $\mathcal{C}_\mathcal{G}$ in $\mathbb{R}^V$, such that there is a point $x \in \mathbb{R}^V \setminus (\mathcal{C}_\mathcal{F} \cup \mathcal{C}_\mathcal{G})$ if and only if $\mathcal{F}$ and $\mathcal{G}$ are not saturated (i.e. there is a set $X \subseteq V$ satisfying (1)).

For $X \subseteq V$, denote respectively by $\mathbb{R}^X_\geq$ and $\mathbb{R}^X_\leq$ the cones $\text{cone}\{\mathbf{e}_i : i \in X\} = \{x \in \mathbb{R}^X :\ x \geq \mathbf{0}\}$ and $\text{cone}\{-\mathbf{e}_i : i \in X\} = \{x \in \mathbb{R}^X :\ x \leq \mathbf{0}\}$, where $\mathbf{e}_i$ denotes the standard ith unit vector. Let $\overline{X} = V \setminus X$, and $\bigcup_{i=1}^{|X|+1} R_i(X) = \mathbb{R}^X$ be the partition of $\mathbb{R}^X$ given by Fact 1. For each $F \in \mathcal{F}$, we define $|V| - |F| + 1$ cones $C^i_F = \mathbb{R}^F_\geq \oplus R_i(\overline{F})$, for $i \in [|\overline{F}| + 1]$, and for each $G \in \mathcal{G}$, we define $|G|+1$ cones $C^i_G = \mathbb{R}^{\overline{G}}_\leq \oplus R_i(G)$, for $i \in [|G|+1]$. Finally, we let $\mathcal{C}_\mathcal{F} = \{C^i_F : F \in \mathcal{F},\ i \in [|\overline{F}| + 1]\}$, $\mathcal{C}_\mathcal{G} = \{C^i_G : G \in \mathcal{G},\ i \in [|G| + 1]\}$, and $\mathcal{C} = \mathcal{C}_\mathcal{F} \cup \mathcal{C}_\mathcal{G}$. Then it is not difficult to see that all the cones in $\mathcal{C}$ are pointed.

Suppose that $X \subseteq V$ satisfies (1). Define $x \in \mathbb{R}^V$ by

$$x_i = \begin{cases} 1, & \text{if } i \in X, \\ -1, & \text{if } i \in V \setminus X. \end{cases}$$

Then $x \notin \cup_{C \in \mathcal{C}} C$. Indeed, if $x \in C^i_F$, for some $F \in \mathcal{F}$ and $i \in [|\overline{F}| + 1]$, then $x_i \geq 0$ and hence $x_i = 1$, for all $i \in F$, implying that $X \supseteq F$. Similarly, if $x \in C^i_G$, for some $G \in \mathcal{G}$ and $i \in [|G| + 1]$, then $x_i \leq 0$ and hence $x_i = -1$, for all $i \in \overline{G}$, implying that $X \subseteq G$.

Conversely, suppose that $x \in \mathbb{R}^V \setminus \mathcal{C}$. Let $X = \{i \in V :\ x_i \geq 0\}$. Then we claim that X satisfies (1). Indeed, if $X \supseteq F$ for some $F \in \mathcal{F}$, then $x_i \geq 0$ for all $i \in F$, and hence there exists an $i \in [|\overline{F}| + 1]$ such that $x \in C^i_F$ (since the cones $R_1(\overline{F}), \ldots, R_{|\overline{F}|+1}(\overline{F})$ cover $\mathbb{R}^{\overline{F}}$). Similarly, if $X \subseteq G$ for some $G \in \mathcal{G}$, then $x_i < 0$ for all $i \in \overline{G}$, and hence there exists an $i \in [|G|+1]$ such that $x \in C^i_G$. In both cases we get a contradiction. □

Corollary 1 ConeCover$[\mathcal{V}, \mathbb{R}^d, III]$, ConeCover$[\mathcal{H}, \mathbb{R}^d, III]$ *and* ConeCover$[\mathcal{H}, \mathbb{R}^t, III]$ *are all NP-complete.*

Proof. NP-completeness of ConeCover$[\mathcal{V}, \mathbb{R}^d, \text{III}]$ and ConeCover$[\mathcal{H}, \mathbb{R}^d, \text{III}]$ follow immediately from Theorem 5. NP-completeness of ConeCover$[\mathcal{H}, \mathbb{R}^t, \text{III}]$ is an immediate consequence of the NP-hardness of ConeCover$[\mathcal{H}, \mathbb{R}^d, \text{III}]$ and the fact that for an $\mathcal{H}$-cone, the intersection of this cone with any affine subspace can be computed easily. □

An interesting special case of problem Sat is when the hypergraphs $\mathcal{F}$ and $\mathcal{G}$ are *transversal* to each other:

$$F \not\subseteq G \text{ for all } F \in \mathcal{F} \text{ and } G \in \mathcal{G}, \quad (2)$$

in which case, the problem is known as the *hypergraph transversal problem*, denoted HyperTrans. Even though the complexity of this problem is still open, it is unlikely to be NP-hard since there exist algorithms [5] that solve the problem in quasi-polynomial time $m^{o(\log m)}$, where $m = |\mathcal{F}| + |\mathcal{G}| + |V|$. Improving this to a polynomial bound is a standing open question. We observe from our reduction in Theorem 5 that ConeCover includes HyperTrans as a special case.

Corollary 2 *Consider a family of cones $\mathcal{C}$ that can be partitioned into two families $\mathcal{C}_1$ and $\mathcal{C}_2$ such that*

$$\text{int}(C_1) \cap \text{int}(C_2) = \emptyset, \quad \textit{for all } C_1 \in \mathcal{C}_1 \textit{ and } C_2 \in \mathcal{C}_2. \quad (3)$$

Then ConeCover$(\mathcal{C}, \mathbb{R}^d)$ *is* HyperTrans*-hard.*

Proof. We note in the construction used on the proof of Theorem 5 that if the hypergraphs $\mathcal{F}$ and $\mathcal{G}$ satisfy (2), then the families of cones $\mathcal{C}_\mathcal{F}$ and $\mathcal{C}_\mathcal{G}$ satisfy (3). Indeed, if $x \in C_F^i \cap C_G^j$, for some $F \in \mathcal{F}$, $i \in [|\overline{F}| + 1]$, $G \in \mathcal{G}$, and $j \in [|G| + 1]$, then $x_k \geq 0$ for all $k \in F$ and $x_k \leq 0$ for all $k \in \overline{G}$. Thus for any $k \in F \setminus G$ (which must exist by (2)), we have $x_k = 0$, implying that x is not an interior point in either C_F^i or C_G^j. □

Freund and Orlin [6] proved that, for an $\mathcal{H}$-polytope P and a $\mathcal{V}$-polytope Q, checking if $Q \supseteq P$ is NP-hard. For all other representations of P and Q, checking $P \subseteq Q$ can be done by solving a linear program. Here we can show that the union version of this problem is hard, no matter how the polytopes are represented.

Corollary 3 *Given a set of $\mathcal{H}$-polytopes $\mathcal{P} = \{P_1, \ldots, P_k\}$ and an $\mathcal{H}$-polytope Q, problem* PolytopeCover$(\mathcal{P}, Q)$ *is NP-hard.*

Proof. We give a reduction from problem ConeCover$[\mathcal{H}, \mathbb{R}^d, \text{III}]$ which is NP-hard by Theorem 5. Let S_d be a "shifted" simplex in $\mathbb{R}^d$ such that $\mathbf{0} \in \text{int}(S_d)$. Given cones $C_1, \ldots, C_k$, we define polytopes $P_1, \ldots P_k$, where $P_i = C_i \cap S_d$. Given the $\mathcal{H}$-representations of C_i, we can compute the $\mathcal{H}$-representations of P_i in polynomial time using linear programming (LP) for removing possible redundancies.

Now one can easily see that $\cup_{i=1}^k C_i = \mathbb{R}^d$ iff $\cup_{i=1}^k P_i = S_d$. □

Corollary 4 *Given a set of $\mathcal{V}$-polytopes $\mathcal{P} = \{P_1, \ldots, P_k\}$ and a $\mathcal{V}$-polytope Q, problem* PolytopeCover$(\mathcal{P}, Q)$ *is NP-hard.*

Proof. We give a reduction from problem ConeCover$[\mathcal{V}, \mathbb{R}^d, \text{III}]$ which is NP-hard by Theorem 5. Recall that in the proof of Theorem 5, for each hyperedge F we construct a set of pointed cones $C_F^i = \mathbb{R}_{\geq}^F \oplus R_i(\overline{F})$, for $i \in [|\overline{F}| + 1]$. Instead of constructing multiple cones for each hyperedge let us just consider one cone $C_F = \mathbb{R}_{\geq}^F \oplus \mathbb{R}^{|\overline{F}|}$ per hyperedge. Similarly for the cones corresponding to the hypergraph $\mathcal{G}$. It is clear that $C_F = \cup_{i=1}^{|\overline{F}|+1} C_F^i$. Note that each such cone is not pointed but instead has a pointed part $\mathbb{R}_{\geq}^F$ corresponding to the vertices in the hyperedge F and the affine space $\mathbb{R}^{|\overline{F}|}$ corresponding to the vertices not in F. Also, $\mathbb{R}_{\geq}^F$ is one orthant in $\mathbb{R}^{|F|}$.

For such cones checking whether the union covers $\mathbb{R}^d$ or not is NP-hard as well (see proof of Theorem 5). Now consider the d-dimensional cross-polytope β_d, and let $C_1, \ldots, C_k$ be the cones constructed above. The cross polytope β_d contains the origin in its interior, and the vertices of $P_i = \beta_d \cap C_i$ for each cone constructed above can be easily computed. It is also easy to see that $\cup_{i=1}^k C_i = \mathbb{R}^d$ iff $\cup_{i=1}^k P_i = \beta_d$. □

Theorem 6 *Given a set of rational convex polytopes $P_1, \ldots, P_k \subseteq \mathbb{Q}^d$, it is coNP-complete to check if their union is convex, for both $\mathcal{H}$ and $\mathcal{V}$-representations of the input polytopes.*

Proof. First we show that the problem is in coNP. Let $Q = \cup_{i=1}^k P_i$. Given two points $x, y \in Q$, we want to verify that the line segment $[x, y] \stackrel{\text{def}}{=} \{\lambda x + (1-\lambda)y |\ \lambda \in [0, 1]\} \not\subseteq Q$. This can be done by iterating the algorithm for two polytopes in [3]: 1. let P be the polytope P_i such that $x \in P_i$; 2. find the (last) point $z \in P$ on the ray $\{x + \lambda(y - x) |\ \lambda \geq 0\}$ such that λ is maximized; 3. if there is another polytope P_j such that $z \in P_j$, then set $P \leftarrow P_j$, $x \leftarrow z$, and go to step 2 else output "No" and halt; 4. if $x = y$ then output "Yes" and halt. The reader can verify that all the above steps can be implemented in polynomial time, given an oracle for LP, and no matter how the polytopes are represented.

Consider the $\mathcal{H}$-representation first. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ and S_d be the polytopes used in the construction in Corollary 3. We now reduce problem PolytopeCover$(\mathcal{P}, S_d)$ to checking if the union of a given set of polytopes is convex. Using an algorithm for the latter problem, we can check if $P = \cup_{i=1}^k P_i$ is convex. If the answer is "No", we conclude that $P \neq S_d$. Otherwise, since $P \subseteq S_d$, either $P = S_d$, or there is hyperplane separating a vertex of S_d from P. The latter condition can be checked in polynomial time by solving k linear programs for each vertex.

For the $\mathcal{V}$-representation the same argument as above works if we use β_d instead of S_d. □

References

[1] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comput. Geom.*, 7:265–301, 1997.

[2] I. Bárány and K. Fukuda. A case when the union of polytopes is convex. *Linear Algebra and its Applications*, 397(6):381–388, 2005.

[3] A. Bemporad, K. Fukuda, and F. D. Torrisi. Convexity recognition of the union of polyhedra. *Comput. Geom.*, 18(3):141–154, 2001.

[4] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.*, 24(6):1278–1304, 1995.

[5] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. 21:618–628, 1996.

[6] R. M. Freund and J. B. Orlin. On the complexity of four polyhedral set containment problems. *Mathematical Programming*, 33(2):139–145, 1985.

[7] H. R. Tiwary. On computing the shadows and slices of polytopes. *CoRR*, abs/0804.4150, 2008.

[8] G. M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics, No. 152. Springer-Verlag, Berlin, 1995.

Linear-Size Meshes*

Gary L. Miller† Todd Phillips‡ Donald Sheehy§

Abstract

Most modern meshing algorithms produce asymptotically optimal size output. However, the size of the optimal mesh may not be bounded by any function of n. In this paper, we introduce *well-paced* point sets and prove that these will produce linear size outputs when meshed with any "size-optimal" meshing algorithm. This work generalizes all previous work on the linear cost of balancing quadtrees. We also present an algorithm that uses well-paced points to produce a linear size Delaunay mesh of a point set in $\mathbb{R}^d$.

1 Introduction

The goal of meshing is to discretize a geometric domain. Such discretizations are necessary for a variety of applications, notably including the finite element method.

We consider the case of meshing a point set $P \subset \mathbb{R}^d$ of size n to produce a "quality" simplicial complex, where quality is a technical condition we describe in Section 3. The vertices of the output include the input set P and some number of Steiner points added to achieve quality.

The most powerful theoretical tool for analyzing meshing algorithms comes from Ruppert[9] in his work on Delaunay refinement meshing in $\mathbb{R}^2$. Define $\text{lfs}_P(x)$, the *local feature size* at a point x in the domain, to be the distance to the second nearest point in P. The following theorem is the standard generalization of Ruppert's results to d-dimensions.

Theorem 1 *The number of number of vertices in any optimal-size quality mesh of a domain* $\Omega \subseteq \mathbb{R}^d$ *is* $\Theta(\int_{x\in\Omega} \frac{1}{\text{lfs}(x)^d} dx)$.

Several known meshing algorithms ([9, 2, 10, 3] to mention a few) witness to the upper bound in Theorem 1, terminating with meshes that are $O(1)$-competitive.

There is a marked absence of n in Theorem 1. In fact, the size of the optimal mesh may not be bounded by any function of n. A guarantee of optimal size output is not a guarantee that output will even be polynomial size. This leads one to believe that perhaps optimality should not be the last word in mesh size analysis.

In this paper, we attack this problem from two directions. First, we show a general condition on point sets for which it is possible to show that the optimal mesh will have linear size. Second, we present an algorithm called LinearMesh, that produces a linear size mesh of any point set by weakening the quality guarantees in regions where the Ruppert lower bound requires a superlinear number of Steiner points.

2 Previous Work

Previous work in simplicial meshing can be roughly divided into two categories, structured and unstructured as typified by quadtree methods and Delaunay refinement respectively. Structured meshing algorithms are characterized by three main properties: a fixed coordinate system, strict control over where Steiner points are added, and predefined mesh templates for filling boxes or other common shapes. These three properties simplify the implementation and analysis of the algorithms, but at a cost. Unstructured meshing algorithms produce meshes that are independent of the coordinate system, allow complete freedom for Steiner point insertion, and have well-defined topology without predefined templates.

Much recent work has sought to bridge the gap between these two paradigms[4, 3, 5].

In this paper, we present two unstructured generalizations of previous results from quadtree meshing. The cost of balancing a quadtree with n boxes is $O(n)$[8]. We present a general class of point sets, of which quadtree vertices are a special case, and for which any quality meshing algorithm will only use a linear number of Steiner points. The algorithm we present in this paper is an unstructured generalization of a quadtree algorithm of Bern et al [2]. The key to both results is a powerful new analytic technique that allows us to analyze optimal mesh size in terms of n without relying on the fixed structure of the quadtree.

3 Well-paced and well-spaced points

We present some standard definitions and introduce two new ones, namely θ-*medial* points and θ-*well-paced ex-*

*This work was partially supported by the National Science Foundation under grant number CCF-0635257.

†Department of Computer Science, Carnegie Mellon University, glmiller@cs.cmu.edu

‡Department of Computer Science, Carnegie Mellon University, tp517@cs.cmu.edu

§Department of Computer Science, Carnegie Mellon University, dsheehy@cs.cmu.edu

tensions of point sets.

Let $\Omega \subseteq \mathbb{R}^d$ be some compact, convex set representing the domain to be meshed. For a point p, denote the distance of p to its nearest neighbor by r_p. The *gap ball* of a point p is the largest empty ball with center in Ω and p on its surface. Let R_p denote the radius of the gap ball of p. A point set P is *ρ-well-spaced* if for every point $p \in P$, $\frac{R_p}{r_p} \leq \rho$.

Say that a point x is *θ-medial* with respect to a point set P if $NN_P(x) \geq \theta \ \mathrm{lfs}_P(x)$ where $NN_P(x)$ denotes the nearest neighbor of x in P. A 1-medial point is equidistant from both nearest neighbors and is thus on the medial axis of P. In general, medial points are near the medial axis where *near* is defined in terms of θ.

An ordered point set $p_1, \ldots, p_n \subset \Omega$ is a *θ-well-paced extension* of a set Q if each p_i is θ-medial with respect to $\{p_1 \ldots, p_{i-1}\} \cup Q$. We call the ordered points p_i *θ-well-paced*. The term "well-paced" is motivated by the way large changes in the local feature are paced out over the sequence of insertions, because any one insertion can only change lfs by a constant factor depending on θ.

In unstructured Delaunay meshing, the topology of the mesh is determined by the location of the points so it is customary to speak of a mesh and a set of points interchangeably. Moreover, properties of point sets have a natural correlation with properties of meshes. For this paper, we will say that a *quality* mesh is simply the triangulation of a well-spaced point set. Usually, mesh quality is defined in terms of some properties of the mesh triangles, but for all of the meshing algorithms we are considering, quality and well-spaced are equivalent notions.

4 Two examples of well-paced point set extensions

The two classic methods for adding points to a mesh are splitting quadtree cells and adding circumcenters of Delaunay triangles. Both methods fit neatly in the theory of well-paced points. In fact, both methods produce 1-well-paced extensions of a constant sized mesh. In Section 5, we show how to bound the cost of meshing such point sets.

Consider the following very simple quadtree construction. Start with a single box. At each step, pick some box and split it in half along each axis. When viewed as a cell-complex, the vertices or 0-faces in this construction form a 1-well-paced extension of the vertices of the initial box. Each time we split a box, we split the corresponding faces in increasing order by dimension. The edge bisectors are 1-medial because the two nearest neighbors must be the endpoints. Likewise, the points splitting higher dimensional faces have nearest neighbors on each lower dimensional face and all are equidistant, so every insertion is 1-medial.

Our second example of a well-paced point set is the case of circumcenter meshes. Start with some Delaunay triangulation of a point set $Q \subset R^d$. At each step, pick a Delaunay triangle and add its circumcenter. The $d+1$ nearest neighbors of a circumcenter at the time of insertion are all the same distance away (they are on the circumsphere), so the circumcenter is 1-medial. Thus, any sequence of circumcenter insertions forms a 1-well-paced extension of Q.

5 The cost of going from well-paced to well-spaced

Running a meshing algorithm on a point set P will add Steiner points until the resulting set P' is well-spaced. The *cost of cleaning* a point set P, denoted by $Cost(P)$ is defined as $|P'|$, the size of the well-spaced output. In this section, we prove that adding an n point well-paced extension of Q will only increase the cost of cleaning by $O(n)$. In Section 4, we showed that inserting points in a quadtree a special case of well-paced points. Balancing a quadtree involves splitting cells to achieve well-spaced vertices. Thus, the result of this section generalizes previous work on the linear cost of balancing quad trees [8].

In particular, if the cost of cleaning Q is $O(1)$ (as is the typical case when Q is a well-spaced bounding box,) then the output mesh will have size $O(n)$.

Theorem 2 *If P is a θ-well-paced extension of Q, then $Cost(Q \cup P) = O(Cost(Q) + |P|)$.*

Proof. The proof will be by induction on $n = |P|$. Let $\mathrm{lfs}^{(i)}$ be the local feature size function induced by $Q \cup \{p_1, \ldots, p_i\}$. Let $\Psi_i = c_1 \int_{x \in \Omega} \frac{1}{\mathrm{lfs}^{(i)}(x)^d} dx$, where c_1 is the constant from the upper bound in Theorem 1. In general, c_1 will depend on the particular meshing algorithm used. Theorem 1 says that $Cost(Q \cup \{p_1, \ldots, p_i\}) \leq \Psi_i$ and $\Psi_0 = O(Cost(Q))$, the base of our induction.

By induction, we assume $\Psi_{n-1} \leq Cost(Q) + c_2(n-1)$ for some constant c_2. It will suffice to show that $\Psi_n - \Psi_{n-1} < c_2$. We can split the Ruppert sizing integral as follows.

$$\Psi_n = c_1 \int_{x \in \Omega} \frac{1}{\mathrm{lfs}^{(n)}(x)^d} dx \tag{1}$$

$$\leq \Psi_{n-1} + c_1 \int_{x \in U} \frac{1}{\mathrm{lfs}^{(n)}(x)^d} - \frac{1}{\mathrm{lfs}^{(n-1)}(x)^d} dx \tag{2}$$

where $U \subseteq \Omega$ is the set of all points for which the local feature size was changed by the insertion of p_n. Let $R = r_{p_n}$. The following two inequalities hold for all $x \in U$, the first is trivial and the second follows from the definition of well-paced points.

$$\mathrm{lfs}^{(n)}(x) \geq |p_n - x|, \text{ and} \tag{3}$$

$$\mathrm{lfs}^{(n-1)}(x) \leq |p_n - x| + \frac{R}{\theta}. \tag{4}$$

We use these inequalities to compute the integral above using spherical coordinates. Since the integrand is positive everywhere, we can upper bound the integral by integrating over all of $\mathbb{R}^d$ instead of just U:

$$\Psi_n - \Psi_{n-1} \leq c_1 \int_{x \in U} \frac{1}{(|x|)^d} - \frac{1}{(|x| + \frac{R}{\theta})^d} dV, \quad (5)$$

$$\leq c_1 \int_0^\infty \int_{S_r} \left(\frac{1}{r^d} - \frac{1}{(r + \frac{R}{\theta})^d} \right) dA dr, \quad (6)$$

$$\leq c_1 s_d \int_0^\infty \left(\frac{1}{r^d} - \frac{1}{(r + \frac{R}{\theta})^d} \right) r^{d-1} dr, \quad (7)$$

where S_r is the sphere of radius r and s_d is the surface area of the unit d-sphere. In the ball of radius $\frac{R}{2}$ around p_n the lfs is at least $\frac{R}{2}$, so the contribution of this region is to Ψ_n at most some constant c_3.

$$\Psi_n - \Psi_{n-1} \leq c_3 + c_1 s_d \int_{\frac{R}{2}}^\infty \left(\frac{1}{r^d} - \frac{1}{(r + \frac{R}{\theta})^d} \right) r^{d-1} dr \quad (8)$$

By the change variable $yR/\theta = r$ and simplifying we get:

$$\Psi_n - \Psi_{n-1} \leq c_3 + c_1 s_d \int_{\frac{\theta}{2}}^\infty \left(\frac{(y+1)^d - y^d}{y(y+1)^d} \right) dy \quad (9)$$

$$\leq c_3 + c_1 s_d \sum_{i=0}^{d-1} \binom{d}{i} \int_{\frac{\theta}{2}}^\infty \frac{y^i}{y^{d+1}} dy \quad (10)$$

$$\leq c_3 + c_1 s_d d^2 \binom{d}{d/2} (2/\theta)^d \quad (11)$$

The last inequality follows from the fact that each integral is bounded by $d(2/\theta)^d$. Choosing c_2 larger than this constant completes the proof. □

One interpretation of this theorem is that the amortized increase in the cost of cleaning a point set is constant if you add a θ-medial point.

Corollary 3 *If Q is a well-spaced point set and P is a well-paced extension then $Cost(Q \cup P) = O(|Q| + |P|)$.*

Proof. Follows from the above theorem and the linear cost of cleaning points that are already well-spaced. □

6 Algorithm LinearMesh

Psuedocode for the algorithm LinearMesh is shown in Figure 1. The algorithm takes a set of points I as input and outputs a superset of points L, such that the Delaunay triangulation of L is linear in the size of I.

The first call is to a simple routine BoundingBox that will calculate the diameter of I and place a bounding box around I that is a constant factor β larger in size. The bounding box is constructed with a constant number of vertices N_β. The bounding box controls the area where new points will be added, and controls interaction with recursive sub-calls.

The `WHILE` loop then selects a subset of I and adds it to P. The selection of θ-medial points makes sure that P is well-paced. The call to DelaunayRefine can invoke any Delaunay refinement algorithm that will accept as input points in a bounding box and produce an optimal quality mesh S. Acceptable algorithms are prevalent in the literature [2, 9, 10, 3].

The `FOREACH` loop now partitions the remaining points from $I - P$ into clusters I_v around the vertices $v \in S$. Each cluster is then meshed recursively, and these points L_v (along with S) are all added together to form L. This is illustrated in Figure 2(d).

6.1 Linearity of LinearMesh

To show that the output of LinearMesh is only linear in the input I, we must first show that it generates only a linear number of vertices.

First, the well-paced pointset P has size $N_\beta + |I \cap P|$. Considering S, by Theorem 2, $|S| \in O(|P|) \in O(|I \cap P|)$. Now consider the recursive partition. Inductively, the cluster submeshes have size $L_v \in O(I_v)$, so their union has total size $O(|I - P|)$. It follows that the final answer has $|L| \in O(|I|)$.

Recall that in three or more dimensions, the number of edges in $Del(L)$ may be $\Omega(|L|^{\lceil d/2 \rceil})$, so we must argue that $|Del(L)|$ is only linear in $|L|$. We claim that the degree of every vertex in $Del(L)$ will be constant. We make use of an established theorem about well-spaced points [7, 3]:

Theorem 4 *If S is a well-spaced point set, then every vertex of $Del(S)$ has constant degree, so that $|Del(S)| \in O(|S|)$ with constants depending on dimension and quality of the well-spacing.*

This theorem guarantees us that every vertex in $Del(S)$ has constant degree (see Figure 2(e)). Next, L was constructed by substituting each L_v for its parent v. Define G as a contraction of $Del(L)$ obtained by collapsing all the vertices of L_v into v (for every choice of v). Intuitively, this contracted graph G should be almost exactly the same as $Del(S)$. (Contracting the graph in Figure 2(f) happens to give exactly Figure 2(e)).

G is not precisely $Del(S)$; its edge structure could be slightly perturbed. The standard "gap-ratio" analysis technique employed in [7, 10, 3] can show that G still has constant degree.

Furthermore, it is straightforward to see the degree of a vertex in $Del(L)$ is at most N_β times as large as its degree in G. Thus $Del(L)$ has constant degree, and so $|Del(L)| \in O(|L|)$.

LinearMesh(I)
  RETURN {} IF $I == \{\}$
  Initialize P =BoundingBox(I)
  WHILE $\exists p \in I - P$ such that p is θ-medial w.r.t. P
    Add p to P
  ENDWHILE
  Initialize $L = S$ =DelaunayRefine(P)
  FOREACH $v \in S$
    $I_v = \{p \in I$ such that $NN_S(p) = v\}$
    L_v = LinearMesh(I_v)
    Add L_v to L
  ENDFOR
  RETURN L

Figure 1: Pseudocode for LinearMesh.

6.2 Output Quality

Besides linearity, we can also guarantee that simplices in the the output triangulation have bounded circumradius to longest edge ratio (R/E). In two dimensions, this is equivalent to bounding the largest mesh angle away from π, which guarantees the quality of the mesh with regards to interpolation [1]. One might hope that in higher dimensions, this condition would guarantee no large dihedral angles, and indeed it does come close, with only the unfortunate exception of allowing sliver tetrahedra. We could imagine a variant involving techniques from [6] that might achieve this guarantee while adding only linearly many new vertices.

7 Conclusions

We have presented a powerful new tool for analyzing quality simplicial meshing algorithms.

In addition our algorithm, LinearMesh, is a fully unstructured method for producing linear size meshes of point sets in $\mathbb{R}^d$. Two potential extensions to this work are to conform to more complex inputs and to use common post processing procedures to improve quality guarantees such as in [6].

It is our hope that this work will fuel new research into optimal (or mostly optimal) quality meshing algorithms with polynomial size output.

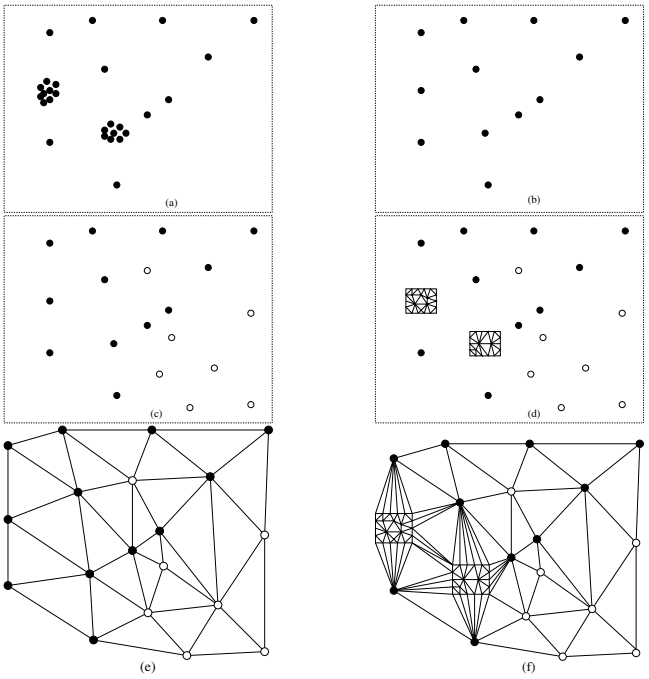


Figure 2: Stages of LinearMesh: (a) Input points I. (b) A well-paced subset P is constructed (bounding box not shown). (c) A well-spaced superset S is constructed. (d) L is S augmented with recursive clusters L_v. (e) The quality mesh of S. (f) Final mesh of L where degree could be larger by a factor of N_β.

References

[1] I. Babuška and A. K. Aziz. On the Angle Condition in the Finite Element Method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, Apr. 1976.

[2] M. Bern, D. Eppstein, and J. R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.

[3] B. Hudson, G. Miller, and T. Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356, Birmingham, Alabama, 2006. Long version available as Carnegie Mellon University Technical Report CMU-CS-06-132.

[4] F. Labelle. Sliver removal by lattice refinement. In N. Amenta and O. Cheong, editors, *Symposium on Computational Geometry*, pages 347–356. ACM, 2006.

[5] F. Labelle and J. R. Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph.*, 26(3):57, 2007.

[6] X.-Y. Li and S.-H. Teng. Generating well-shaped Delaunay meshed in 3D. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 28–37. ACM Press, 2001.

[7] G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. On the radius–edge condition in the control volume method. *SIAM J. Numer. Anal.*, 36(6):1690–1708, 1999.

[8] D. Moore. The cost of balancing generalized quadtrees. In *SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 305–312, 1995.

[9] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Austin, TX, 1993).

[10] A. Üngör. Off-centers: A new type of steiner points for computing size-optimal guaranteed-quality delaunay triangulations. In *Proceedings of LATIN*, 2004.

Exact Pareto-Optimal Coordination of Two Translating Polygonal Robots on a Cyclic Roadmap

Hamidreza Chitsaz* Steven M. LaValle* Jason M. O'Kane†

Abstract

We consider planning optimal collision-free motions of two polygonal robots under translation. Each robot has a reference point that must lie on a given graph, called a roadmap, which is embedded in the plane. The initial and the goal are given for each robot. Rather than impose an a priori cost scalarization for choosing the best combined motion, we consider finding motions whose cost vectors are Pareto-optimal. Pareto-optimal coordination strategies are the ones for which there exists no strategy that would be better for both robots. Our problem translates into shortest path problems in the coordination space which is the Cartesian product of the roadmap, as a cell complex, with itself. Our algorithm computes an upper bound on the cost of each motion in any Pareto-optimal coordination. Therefore, only a finite number of homotopy classes of paths in the coordination space need to be considered. Our algorithm computes all Pareto-optimal coordinations in time $O(2^{5\alpha}m^{1+5\alpha}n^2\log(m^{2\alpha}n))$, in which m is the number of edges in the roadmap, n is the number of coordination space obstacle vertices, and $\alpha = 1 + \lceil(5\ell + r)/b\rceil$ where ℓ is total length of the roadmap and r is total length of coordination space obstacle boundary and b is the length of the shortest edge in the roadmap.

1 Introduction

Previous approaches to multiple-robot motion planning are often categorized as *centralized* or *decoupled*. A centralized approach typically constructs a path in a composite configuration space, which is formed by the Cartesian product of the configuration spaces of the individual robots. A decoupled approach typically generates paths for each robot independently, and then considers the interactions between the robots. In [9, 15], an independent roadmap is computed for each robot, and coordination occurs on the Cartesian product of the roadmap path domains. The suitability of one approach over the other is usually determined by the tradeoff between computational complexity associated with a given problem, and the amount of completeness that is lost. In some applications, such as the coordination of Automated Guided Vehicles (AGV), the roadmap might represent all allowable mobility for each robot.

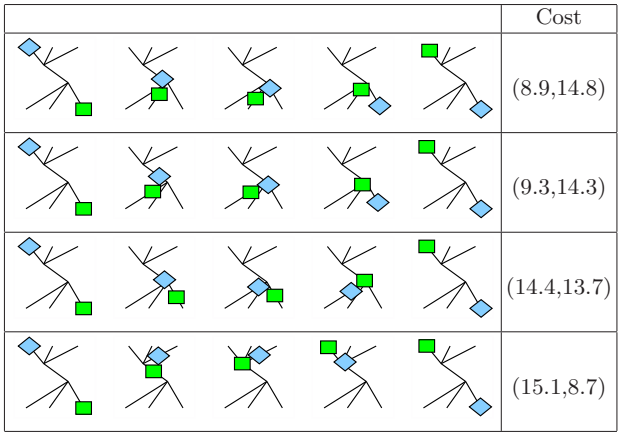


Figure 1: The four Pareto-optimal solutions for a coordination problem in which the robots want to exchange place.

In this paper, we study the problem of planning optimal motions of two polygonal robots traveling on a given roadmap. The robots must be disjoint when they travel, and as a result, there are tradeoffs between the robots' completion times. One approach is to consider a scalar cost that combines the completion times. Minimizing the average time robots take to reach their goals [8, 11], and minimizing the time that the last robot takes have been studied before [14]. The problem with scalarization is that it eliminates many interesting coordination strategies, possibly even neglecting optimality for some robots [9]. Rather than impose an a priori scalarization for choosing the best combined motion, we consider finding motions whose cost vectors (cost of robot 1, cost of robot 2) are Pareto-optimal. Pareto-optimal coordination strategies are the ones for which there exists no strategy that would be better for both robots; see [13]. Optimal coordinations according to a scalar cost impose a predetermined preference between the robots, whereas having all Pareto-optimal coordinations beforehand gives the freedom to determine the preference at run-time. It was shown that the number of Pareto-optimal coordinations for n robots on any

*Department of Computer Science, University of Illinois at Urbana-Champaign, {chitsaz,lavalle}@cs.uiuc.edu

†Department of Computer Science and Engineering, University of South Carolina, jokane@cse.sc.edu

roadmap is finite [5]; therefore, it is plausible to seek all of them. A sample problem and its Pareto-optimal solutions are illustrated in Figure 1.

This work is inspired by previous approaches to multiple robot coordination. O'Donnell and Lozano-Pérez introduced coordination diagrams for planning motions of two robot manipulators [12]. Alt and Godau used similar coordination spaces in a different context to compute the Fréchet distance between two polygonal curves [1]. LaValle and Hutchinson gave the first approach to Pareto-optimal coordination of multiple robots [9]. They presented an approximation algorithm based on dynamic programming in the discretized coordination space. Ghrist et al. gave a characterization of Pareto-optimal coordinations of multiple robots using CAT(0) geometry [6]. They provided an algorithm to shorten a given coordination to a homotopic, possibly Pareto-optimal one. In our previous work, we gave an efficient algorithm for finding Pareto-optimal coordination strategies for two polygonal robots on an acyclic roadmap [4]. In this paper, we present an algorithm for the general case. Due to space limitations, proofs of the propositions, lemmas, and theorems are eliminated.

2 Problem Formulation

We give a brief formulation of the problem. For a more detailed exposition, see [4]. Let the robots, $\mathcal{R}_1$ and $\mathcal{R}_2$, be polygonal open sets embedded in the plane. They translate along a roadmap $\mathcal{G}$, which is an embedded graph in the plane[1]. Edges of $\mathcal{G}$ are piecewise-linear segments. The roadmap need not be connected, so effectively each robot can have its own roadmap. Each edge of $\mathcal{G}$ is weighted by its Euclidean length. In this way, $\mathcal{G}$ turns into a metric graph [3]. The robots have a maximum speed and are capable of instantly switching to any speed between zero and the maximum. By scaling the respective metric graphs, we assume without loss of generality that both robots have unit maximum speed. Under this assumption, the distance function $d(x, y)$ gives the minimum amount of time that it takes $\mathcal{R}_i$ to go from x to y on $\mathcal{G}$.

We are given an initial and a goal configuration $q_i^{init}, q_i^{goal} \in \mathcal{G}$ for each robot $\mathcal{R}_i$. The obstacle region, denoted by $\mathcal{O} \subset \mathcal{G} \times \mathcal{G}$, is the set of configurations at which $\mathcal{R}_1$ and $\mathcal{R}_2$ collide. Since the robots are polygonal and roadmap paths are piecewise-linear, the obstacle region is a collection of polygonal, open connected components. A coordination is a continuous path in the coordination space $\mathcal{G} \times \mathcal{G}$, from $q^{init} = (q_1^{init}, q_2^{init})$ to $q^{goal} = (q_1^{goal}, q_2^{goal})$, that avoids $\mathcal{O}$.

[1] If we assume that $\mathcal{G}$ is locally embedded in the plane, in which case its edges may intersect, then our algorithm correctly works and our results still hold. For the sake of clarity, we preferred to assume $\mathcal{G}$ is embedded.

The vector-valued cost $\mathcal{J} = (J_1, J_2)$ separately measures the time that each robot takes to reach its goal and stop. Define $d_\infty : ((x_1, x_2), (y_1, y_2)) \mapsto \max(d(x_1, y_1), d(x_2, y_2))$, in which d is the metric in $\mathcal{G}$. Let $\mathcal{L}^\infty$ be the functional that gives the length of each continuous path in $\mathcal{G} \times \mathcal{G}$ according to d_∞. For each coordination $\gamma = (\gamma_1, \gamma_2) : [0, 1] \to \mathcal{G} \times \mathcal{G}$, let $t_i = \min\{t \in [0, 1] : \gamma_i([t, 1]) = q_i^{goal}\}$. In that case, $J_i(\gamma) = \mathcal{L}^\infty(\gamma|_{[0,t_i]})$ and $\mathcal{J}(\gamma) = (J_1(\gamma), J_2(\gamma))$. Let C be the set of all coordinations. The cost $\mathcal{J} : \mathrm{C} \to [0, \infty)^2$ induces a partial order on the set of all coordinations C. Each minimal element in this partial order is called a Pareto-optimal coordination. The problem is to find all Pareto-optimal coordinations for the two robots.

3 Canonical Pareto-optimal Coordinations

Different paths that have the same end points can have equal $\mathcal{L}^\infty$ lengths in the coordination space. Consequently, there are different coordinations with equal cost. We fix a canonical form for equivalent Pareto-optimal coordinations based on Euclidean shortest paths.

Proposition 1 *For every Pareto-optimal coordination, there is an equivalent coordination that is composed of a finite sequence of Euclidean shortest segments between the vertices of the obstacle region, q^{init}, q^{goal}, and in some cases (x, q_2^{goal}) or (q_1^{goal}, x).*

The points (q_1^{goal}, x) and (x, q_2^{goal}) that need to be considered are characterized in [4]. A point (q_1^{goal}, x) or (x, q_2^{goal}) needs to be considered if there is a collision-free Euclidean shortest segment, with equal progression for $\mathcal{R}_1$ and $\mathcal{R}_2$, from an obstacle vertex or q^{init} to the point (q_1^{goal}, x) or (x, q_2^{goal}).

4 Algorithm Presentation

To find canonical Pareto-optimal coordinations, our algorithm computes Euclidean shortest segments between obstacle vertices, initial and goal configurations, and some points (q_1^{goal}, x) and (x, q_2^{goal}) in the coordination space. Fixing the end points in the coordination space, there is only one shortest path in every homotopy class, which holds because the space is non-positively curved [5]. The roadmap can be cyclic, and consequently the universal cover of the coordination space can be unbounded. An incremental exploration of the unbounded universal cover may never stop, because there are multiple Pareto-optimal coordinations whose maximum length is unknown beforehand. Our algorithm constructs a bounded portion of the universal cover in which the shortest path algorithm is applied. Using

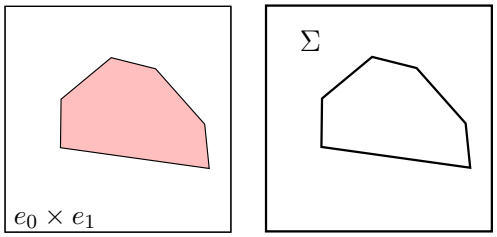


Figure 2: A coordination cell $e_0 \times e_1$, and its skeleton Σ.

shortest path algorithms in the plane such as continuous Dijkstra [7, 10] or visibility graph methods in the universal cover of the coordination space, one can compute the shortest paths. Using a cost upper bound computed in advance, our algorithm constructs the relevant part of the universal cover. The rest of the algorithm is essentially identical to the acyclic case applied to the universal cover [4].

4.1 Coordination Cost Upper Bound

In a scalar minimization problem, the cost of any feasible solution is an upper bound for the cost of an optimal solution. The key idea here is the same. The following lemma derives an upper bound on the cost of every motion in any Pareto-optimal coordination.

Lemma 2 *Let $\Delta_1, \Delta_2 \subseteq \mathcal{G}$ be such that $\{q_1^{goal}\} \times \Delta_2 = \{q_1^{goal}\} \times \mathcal{G} - \mathcal{O}$, and $\Delta_1 \times \{q_2^{goal}\} = \mathcal{G} \times \{q_2^{goal}\} - \mathcal{O}$. Let δ_i be the diameter of Δ_i as a metric graph. Let λ be the Euclidean length of an arbitrary coordination γ. Let τ be a Pareto-optimal coordination. In that case, $J_1(\tau), J_2(\tau) \leq \lambda + \delta$, in which $\delta = \max(\delta_1, \delta_2)$.*

To compute λ, which is the Euclidean length of an arbitrary coordination γ, we use the dimension reduction method of Aronov et al. [2]. Denote the boundary of obstacle region by $\partial\mathcal{O}$. Define $\Upsilon_1 = \{q_1^{init}\} \times \mathcal{G} - \mathcal{O}, \Upsilon_2 = \mathcal{G} \times \{q_2^{init}\} - \mathcal{O}, \Upsilon_3 = \{q_1^{goal}\} \times \mathcal{G} - \mathcal{O}, \Upsilon_4 = \mathcal{G} \times \{q_2^{goal}\} - \mathcal{O}$, and $\Sigma = \partial\mathcal{O} \cup (\bigcup_{j=1}^{4} \Upsilon_j)$. We call Σ the *skeleton* of $\mathcal{G} \times \mathcal{G} - \mathcal{O}$. See Figure 2 for a simple example. Note that the skeleton is a one-dimensional object. It is composed of five pieces: $\mathcal{R}_1$ at its initial, $\mathcal{R}_2$ at its initial, $\mathcal{R}_1$ at its goal, $\mathcal{R}_2$ at its goal, and $\mathcal{R}_1$ touching $\mathcal{R}_2$. The following lemma follows from Lemma 1 in [2].

Lemma 3 (Aronov et al. [2]) *There is a collision-free path from q^{init} to q^{goal} in the coordination space if and only if there is a path from q^{init} to q^{goal} in Σ, the skeleton of $\mathcal{G} \times \mathcal{G} - \mathcal{O}$.*

Our algorithm constructs Σ by gluing $\partial\mathcal{O}$ and Υ_j along their intersection points. We discussed how to compute the obstacle region in [4]. To compute Υ_j, first we compute $\mathcal{M} = \mathcal{R}_1 \ominus \mathcal{R}_2$, the Minkowski difference. By intersecting polygon $\mathcal{M}$ positioned respectively at q_1^{init} and q_1^{goal} with $\mathcal{G}$, we compute $\Gamma_2 = \mathcal{G} - (\{q_1^{init}\} \oplus \mathcal{M})$ and $\Delta_2 = \mathcal{G} - (\{q_1^{goal}\} \oplus \mathcal{M})$. By intersecting $-\mathcal{M}$ positioned respectively at q_2^{init} and q_2^{goal} with $\mathcal{G}$, we compute $\Gamma_1 = \mathcal{G} - (\{q_2^{init}\} \ominus \mathcal{M})$ and $\Delta_1 = \mathcal{G} - (\{q_2^{goal}\} \ominus \mathcal{M})$. It is enough to observe that $\Upsilon_1 = \{q_1^{init}\} \times \Gamma_2$, $\Upsilon_2 = \Gamma_1 \times \{q_2^{init}\}$, $\Upsilon_3 = \{q_1^{goal}\} \times \Delta_2$, and $\Upsilon_4 = \Delta_1 \times \{q_2^{goal}\}$. Dijkstra's algorithm yields γ and the minimum distance of q^{goal} from q^{init} in Σ which is taken as λ. Finally, the diameter, or an overestimate of the diameter, of Δ_i yields δ_i. Recall that the upper bound is $\lambda + \max(\delta_1, \delta_2)$.

4.2 Universal Cover of $\mathcal{G} \times \mathcal{G}$

Given the upper bound computed in Section 4.1, we only need to consider a finite portion of the universal cover. Here we describe an algorithm to construct it. Let $\mathcal{X}$ be the universal cover of $\mathcal{G}$ as a cell complex. In that case, $\mathcal{X} \times \mathcal{X}$ is the universal cover of $\mathcal{G} \times \mathcal{G}$, and it is enough to build the relevant part of $\mathcal{X}$ to construct the relevant part of $\mathcal{X} \times \mathcal{X}$.

Since $\mathcal{X}$ is composed of disjoint copies of a fundamental domain glued along identified vertices, we describe how to build a fundamental domain, denoted by $\mathcal{X}_0$. Let $\mathcal{T}$ be any spanning tree of $\mathcal{G}$ (a collection of trees if $\mathcal{G}$ is not connected). Let $e_i = (u_i, v_i), i = 1, \ldots, k$ be those edges of $\mathcal{G}$ that are not in $\mathcal{T}$. Obtain $\mathcal{X}_0$, the fundamental domain of $\mathcal{X}$, by adding k new vertices u_i^* and k edges (v_i, u_i^*) to $\mathcal{T}$. Note that the length of (v_i, u_i^*) is the same as that of (u_i, v_i). Cycles of $\mathcal{G}$ are opened into paths in $\mathcal{X}_0$. Vertices u_i^* must be identified with u_i in neighboring copies of the fundamental domain. We call u_i and u_i^* *gluing spots* of $\mathcal{X}_0$, because $\mathcal{X}$ is obtained by iteratively gluing disjoint copies of the fundamental domain to $\mathcal{X}_0$ such that $u_i \in \mathcal{X}_0$ is identified with u_i^* in one copy and $u_i^* \in \mathcal{X}_0$ is identified with u_i in another copy. If you want to see an illustration, see the long version of the paper.

Our algorithm builds $\mathcal{X}_0$ first, and initializes $\mathcal{Y} = \mathcal{X}_0$. It inserts u_i and u_i^* onto a list. For every vertex in the list, the algorithm generates a copy of $\mathcal{X}_0$ and glues it to $\mathcal{Y}$ along the relevant vertex. It then inserts the gluing spots of the newly generated copy in the list. It iterates over these steps until $\mathcal{Y}$ covers the relevant part of $\mathcal{X}$. For that purpose, the distance between the vertex and the initial copy of $\mathcal{X}_0$ is computed at each iteration. If that distance is more than the upper bound, then the vertex is neglected and no copies of $\mathcal{X}_0$ is glued. Eventually, the algorithm stops when there are no more vertices in the list.

4.3 Applying the Acyclic Algorithm

We showed how to compute $\mathcal{Y}$, the relevant portion of the universal cover of $\mathcal{G}$, in Section 4.2. Note that $\mathcal{Y} \subset \mathcal{X}$ is contractible. Therefore, it is acyclic and we may now apply our acyclic Pareto-optimal coordination algorithm to it [4]. The acyclic algorithm computes the visibility graph in $\mathcal{Y}\times\mathcal{Y}$ among obstacle vertices and the initial and goal configurations, augments it with some extra edges, and finds the shortest paths. Obstacles are computed once in $\mathcal{G} \times \mathcal{G}$, and they are copied multiple times to obtain obstacles in $\mathcal{Y} \times \mathcal{Y}$. There are several copies of q^{goal} in $\mathcal{Y}\times\mathcal{Y}$ all of which need to be considered in the visibility graph. Any collision-free path from q^{init} to any q^{goal} copy is a coordination. Consequently, there are several copies of visibility graph points (x, q_2^{goal}) and (q_1^{goal}, x) that need to be considered.

4.4 Complexity Analysis

Let m denote the number of edges in $\mathcal{G}$ and let n denote total number of obstacle vertices in $\mathcal{G} \times \mathcal{G}$. Let ℓ be the total length of $\mathcal{G}$ and r the total length of obstacle boundary. Let b denote the length of the shortest edge in $\mathcal{G}$. Define $\alpha = 1 + \lceil (5\ell + r)/b \rceil$.

Theorem 4 *The time complexity of our algorithm is* $O(2^{5\alpha} m^{1+5\alpha} n^2 \log(m^{2\alpha} n))$.

5 Conclusion

We presented an algorithm to compute all Pareto-optimal coordinations of two polygonal robots on a network of piecewise-linear paths in the plane. The key insight was an upper bound on the cost of each motion in a Pareto-optimal coordination. Our algorithm applies the previous acyclic algorithm to a finite portion of the universal cover of the coordination space [4]. This method can be applied to find all Pareto-optimal coordinations, provided the configuration space of each robot is $\mathcal{G}$, all paths in $\mathcal{G} \times \mathcal{G}$ are allowed, and the obstacle regions in $\mathcal{G} \times \mathcal{G}$ are polygonal.

Acknowledgment

We thank Jeff Erickson for his helpful comments. H. Chitsaz and S.M. LaValle are partially supported on NSF Award 0528086 (MSPA-MCS).

References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.

[2] B. Aronov, M. de Berg, A. van der Stappen, P. Švestka, and J. Vleugels. Motion planning for multiple robots. *Discrete and Computational Geometry*, 22(4):505–525, 1999.

[3] M. Bridson and A. Haefliger. *Metric Spaces of Nonpositive Curvature*. Springer-Verlag, Berlin, 1999.

[4] H. Chitsaz, J. M. O'Kane, and S. M. LaValle. Exact pareto-optimal coordination of two translating polygonal robots on an acyclic roadmap. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3981 – 3986, New Orleans, LA, Apr. 2004.

[5] R. Ghrist and S. M. LaValle. Nonpositive curvature and pareto-optimal coordination of robots. *SIAM J. Control & Optimization*, to appear 2006.

[6] R. Ghrist, J. M. O'Kane, and S. M. LaValle. Computing pareto optimal coordinations on roadmaps. *International Journal of Robotics Research*, 24(11):997–1010, Nov. 2005.

[7] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Computing*, 28:2215–2256, 1999.

[8] H. Hu, M. Brady, and P. Probert. Coping with uncertainty in control and planning for a mobile robot. In *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, pages 1025–1030, Osaka, Japan, Nov. 1991.

[9] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Trans. on Robotics and Automation*, 14(6):912–925, Dec. 1998.

[10] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Int. J. Comput. Geom. & Appl.*, 6(3):309–332, 1996.

[11] J. Miura and Y. Shirai. Planning of vision and motion for a mobile robot using a probabilistic model of uncertainty. In *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, pages 403–408, Osaka, Japan, May 1991.

[12] P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *IEEE Int'l. Conf. Robot. & Autom.*, pages 484–489, 1989.

[13] Y. Sawaragi, H. Nakayama, and T. Tanino. *Theory of Multiobjective Optimization*. Academic Press, New York, NY, 1985.

[14] S.-H. Suh and K. G. Shin. A variational dynamic programming approach to robot-path planning with a distance-safety criterion. *IEEE Trans. Robot. & Autom.*, 4(3):334–349, June 1988.

[15] P. Svestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *IEEE Int'l. Conf. Robot. & Autom.*, pages 1631–1636, 1995.

Open Problems from CCCG 2007

Erik D. Demaine* Joseph O'Rourke†

The following is a list of the problems presented on August 20, 2007 at the open-problem session of the 19th Canadian Conference on Computational Geometry held in Ottawa, Ontario, Canada.

Rolling a Sphere Upside-Down

Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Imagine rolling a unit sphere on a plane without slipping or twisting, so that the point of contact follows a closed curve $\mathcal{C}$. The south pole touches the plane at the start. What is the shortest length $L = |\mathcal{C}|$ that results in the north pole touching after one complete circuit of $\mathcal{C}$? Figure 1 shows a curve achieving $L = 3\pi$. Also, $\mathcal{C}$ cannot be shorter than the geodesic distance between the poles: $L \geq \pi$. Another $\mathcal{C}$ that achieves $L = 3\pi$ is an equilateral triangle of side length π. It is established in [Joh07] that the poles cannot be interchanged by any $\mathcal{C}$ that is a circle.

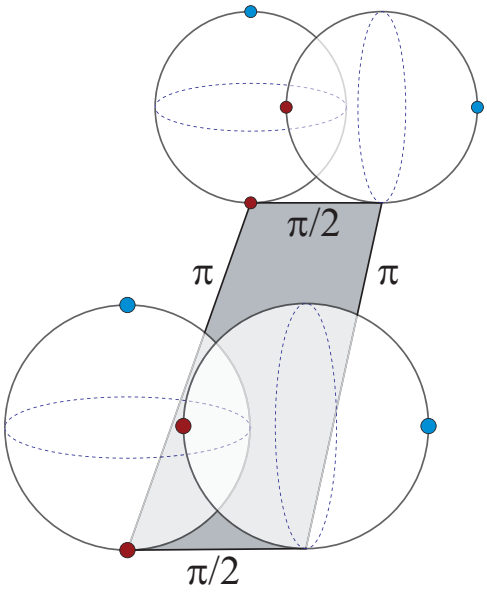


Figure 1: Rolling a sphere to interchange the north and south pole. Here $|\mathcal{C}| = 3\pi$.

Update. At the conference, Jack Snoeyink found a path with length approximately $2.72\,\pi$. Vishal Verma, a graduate student at UNC Chapel Hill, joined him [VS07] to improve this to $< 2.44\,\pi$ for a teardrop-shaped path depicted in Figure 2.

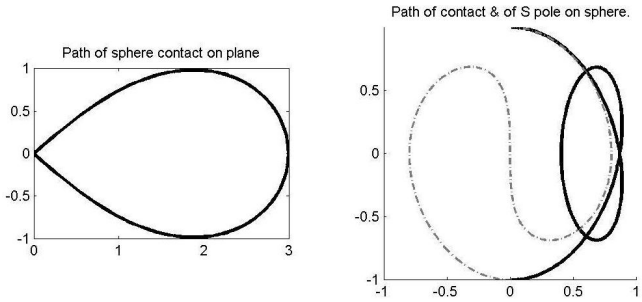


Figure 2: Path of plane/sphere contact in plane (left) and on sphere (right). Dashed curve at right is the path of the south pole as the sphere rolls.

Hammersley [Ham83] had posed a more general form of this problem in the literature on optimal control: for a unit sphere lying on the plane at (x_0, y_0) and having initial orientation C_0, determine the shortest path for it to roll without twisting that brings it to point (x_1, y_1) with orientation C_1. Although the solution to Hammersley's problem does not in general have a closed form, Arthurs and Walsh [AW86] have given an expression as a boundary-value problem with ten coupled partial differential equations that they use to derive information on the curvature of optimal paths. For the special case of a closed path above, their result implies that the curvature is proportional to the distance along the axis of symmetry, which establishes that the curve depicted in Figure 2 is optimal if self-intersections are ruled out.

References

[AW86] A. M. Arthurs and G. R. Walsh. On Hammersley's minimum problem for a rolling sphere. *Mathematical Proceedings of Cambridge Philosophical Society*, 99:529–534, 1986.

[Ham83] J. M. Hammersley. Oxford commemoration ball. In *Probability, Statistics*

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139, USA, edemaine@mit.edu
†Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu

and Analysis, Cambridge University Press, 1983, pp. 112–142.

[Joh07] Brody D. Johnson. The nonholonomy of the rolling sphere. *American Mathematical Monthly*, 116(6):500–508, 2007.

[VS07] Vishal Verma and Jack Snoeyink. Upending a sphere. In *Abstracts from the 17th Annual Fall Workshop on Computational Geometry*, 2007. http://www.research.ibm.com/people/l/lenchner/fwcg2007/Proceedings/submission_35.pdf

Spanning Trees of the Graph of a Polyhedron

Alex Benton
Cambridge University
pb355@damtp.cam.ac.uk

What is the best possible bound on the number of spanning trees of the 1-skeleton of a polyhedron, i.e., a 3-connected planar graph?

Update. An upper bound on the number of spanning trees of a polyhedron graph is derived in [DO07, p. 431], based on a result of McKay [McK83]: the number is $O((16/3)^n/n)$, which is $2^{O(n)}$.

References

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

[McK83] Brendan D. McKay. Spanning trees in regular graphs. *European Journal of Combinatorics*, 4(2):149–160, 1983.

Minimum Length Barrier to X-rays in a Square

Jonathan Lenchner
IBM T.J. Watson Research Center
lenchner@us.ibm.com

Given a unit square, construct a barrier of minimum length that intersects every line passing through a portion of the square. The barrier should consist of one or more piecewise-smooth curves. The barrier need not be connected and portions of the barrier may be located inside, outside, or on the boundary of the square.

This problem appeared in the July 2007 edition of IBM's online puzzle column *Ponder This*[1]. According to the column, the problem has also appeared in [Jon64] and in an internal publication of the Lawrence Livermore National Laboratory.

[1]See http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/challenges/July2007.html.

The obvious barrier is the entire perimeter of the square, with cost 4. However, we can also use just three sides of the square, at cost 3. Even better, we can use the two diagonals, at cost $2\sqrt{2} \simeq 2.828$. Better still is to use two adjacent edges of the square and half of the opposite diagonal, at cost $2 + \sqrt{2}/2 \simeq 2.707$, as shown in Figure 3.

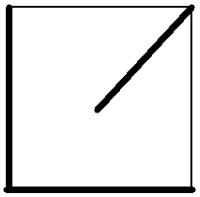


Figure 3: A barrier of length $2 + \sqrt{2}/2 \simeq 2.707$.

As a final observation, we can do better than using the two adjacent edges in the previous barrier, by instead using a three-segment Steiner tree as in Figure 4, at cost $\sqrt{2} + \sqrt{6}/2 \simeq 2.639$. This barrier is conjectured, but not known, to be optimal.

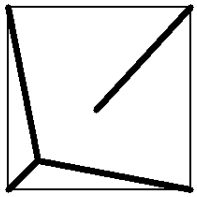


Figure 4: Place the bottom left corner of the square at $(0, 0)$. Then a barrier consisting of the diagonal segment $[(1/2, 1/2), (1, 1)]$ together with three segments formed by joining the corners $(0, 1), (0, 0), (1, 0)$ to a point at $(\frac{1}{2} - \frac{\sqrt{3}}{6}, \frac{1}{2} - \frac{\sqrt{3}}{6})$ yields a barrier of length $\sqrt{2} + \frac{\sqrt{6}}{2} \simeq 2.639$.

At the conference, Otfried Cheong suggested that a lower bound of 2 could be derived from the Cauchy-Crofton formula, and this was later verified to be the case.

A clever and even more elementary proof that 2 is a lower bound was found by Ozgur Ozkan, a student of John Iacono's at Polytechnic University. Ozgur's argument runs as follows:

By a limiting argument, we may assume that the (approximately) optimal solution is piecewise linear. Thus let $S = \{x_1, x_2, \ldots\}$ be the set of line segments making up a barrier. Let the two diagonals of the square be denoted by d_1 and d_2. In order to block just the rays which are perpendicular to each of these two diagonals, the projection of S onto d_1 and d_2 must cover d_1 and d_2. Thus, if θ_i is the angle x_i makes with d_1, then $|x_i| \cos\theta_i$ is the length of x_i's projection onto d_1, and $|x_i| \sin\theta_i$

is the length of x_i's projection onto d_2. Therefore

$$\begin{aligned} 2\sqrt{2} &\leq \sum_{x_i \in S} |x_i|(\cos\theta_i + \sin\theta_i) \\ &\leq \sum_{x_i \in S} |x_i|\sqrt{2}, \quad \text{so} \\ 2 &\leq \sum_{x_i \in S} |x_i|. \end{aligned}$$

References

[Jon64] R. E. D. Jones. Opaque sets of degree α. *American Mathematical Monthly*, 71:535–537, May 1964.

Doubly Orthogonal Point Set?

Therese Biedl
University of Waterloo
biedl@softbase.uwaterloo.ca

Is there a point set in 2D that is the vertex set of an orthogonal polygon such that a rotation of the point set by a nonmultiple of 90° is also the vertex set of an orthogonal polygon?

Precisely, an orthogonal polygon is a (simple) polygon whose edges are all either horizontal or vertical. A vertex is a point on the polygon where two edges of different slopes meet. (Put differently, we do not allow "extra" vertices along the edges.) The open problem is motivated by the question of whether it is possible to reconstruct an orthogonal polygon when given only a set of points that is supposed to be its vertex set. (Think of the popular children's game connect-the-dots, except that the numbers on the dots are illegible.)

This problem has been well-studied. O'Rourke [O'R88] showed that a simple scanning algorithm can recover the orthogonal polygon in $O(n \log n)$ time. Rappaport [Rap86] showed that, if extra vertices are allowed, then the problem becomes NP-hard.

We recently started studying the problem where the given point set is allowed to be rotated before reconstructing the orthogonal point set ([Gen07]; see also [BG07]). Clearly only $O(n)$ rotations could possibly yield an orthogonal polygon, because at least four edges of the polygon have to be on an edge of the convex hull. Therefore, an $O(n^2 \log n)$ algorithm for this problem is trivial. We managed to improve the time complexity to $O(n \log n)$ for orthogonally convex polygons. The crucial insight was that, for orthogonally convex polygons, there can be only one rotation that could possibly work; and the proof of this insight yielded an algorithm to find this rotation efficiently.

This raises the natural question of what happens with orthogonal polygons that are not orthogonally convex. Our proof very clearly fails for such polygons, hence the open question: is there only one rotation that works? Or could there be two different rotations for which the set of points is the vertex set of an orthogonal polygon? The only negative example that we could find consists of the points of a regular octagon, which can be interpreted as the vertex sets of two rectangles in two different rotations. But this is neither a single polygon nor simple.

We would also be interested in whether one could find the rotation efficiently, if there is only one.

Update: Maarten Löffler and Elena Mumford [LM07] resolved this question in the negative. In fact, they consider a more general problem: given a set of points in $\mathbb{R}^d$, is it the vertex set of some connected rectilinear graph (not necessarily a polygon)? They prove that any point set in $\mathbb{R}^d$ has at most one orientation where it is a vertex set of a connected rectilinear graph. For contrast, Figure 5 shows an example of two rectilinear graphs on the same point set, but note that G' is not connected (similar to the regular-octagon example above). For the special case where the points are in the plane and have rational coordinates, Fekete and Woeginger [FW97, Theorem 4.7] already proved that at most one orientation is possible.

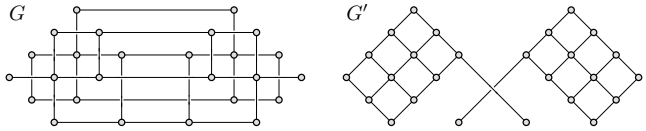


Figure 5: Two rectilinear graphs in the plane with the same vertex set, but different slopes. Note that G' is not connected.

The details of the argument in the planar case are mostly algebraic, but the main idea is to determine the "greatest common divisor" in some sense between the coordinates of the vertices of an axis-aligned connected rectilinear graph, and argue that, if there is a connected rectilinear graph in another orientation, we can use that to travel to a point that is not a multiple of this divisor away from the others, which leads to a contradiction. The result can be extended to arbitrary dimensions by simply projecting it down to a suitable plane.

To find the right orientation of the graph (in the planar case), we can improve the $O(n^2 \log n)$ result to $O(n^2)$ by taking the dual of the problem. A relatively simple algorithm can sweep the arrangement

of lines and identify potential good orientations, of which there are at most $O(n)$. We maintain the sorted order of the other lines, which allows us to test an orientation in linear time.

If the goal is to find a orientation that allows a *planar* connected rectilinear graph (a more general case than simple polygons), then the problem becomes NP-hard. Because we know that there is at most one such orientation, we can use the algorithm described above to find it. Then we can compute the maximal rectilinear graph with this slope. However, now we need to decide whether this graph has a planar connected subgraph. Jansen and Woeginger [JW93] proved this problem NP-complete.

The main remaining open question is whether the (unique) rectilinear orientation can be computed in less than quadratic time. Also, algorithmic extensions to higher dimensions are open.

References

[BG07] Therese Biedl and Burkay Genc. Reconstructing orthogonal polyhedra from putative vertex sets. Technical Report CS-2007-28, School of Computer Science, University of Waterloo, August 2007. http://www.cs.uwaterloo.ca/research/tr/2007/CS-2007-28.pdf

[FW97] Sándor P. Fekete and Gerhard J. Woeginger. Angle-restricted tours in the plane. *Computational Geometry: Theory and Applications*, 8(4):195–218, 1997.

[Gen07] Burkay Genc. Reconstruction of Orthogonal Polyhedra. Ph.D. Thesis, School of Computer Science, University of Waterloo, December 2007.

[JW93] Klaus Jansen and Gerhard J. Woeginger. The complexity of detecting crossingfree configurations in the plane. *BIT*, 33(4):580–595, 1993.

[LM07] Maarten Löffler and Elena Mumford. Rotating rectilinear graphs. In *Abstracts from the 17th Annual Fall Workshop on Computational Geometry*, 2007. http://www.research.ibm.com/people/l/lenchner/fwcg2007/Proceedings/submission_23.pdf

[O'R88] Joseph O'Rourke. Uniqueness of orthogonal connect-the-dots. In Godfried T. Toussaint, editor, *Computational Morphology*, pages 97–104. North-Holland, Amsterdam, Netherlands, 1988.

[Rap86] David Rappaport. On the complexity of computing orthogonal polygons from a set of points. Technical Report TR-SOCS-86.9, McGill University, 1986.

Pushing Cubes Around

Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Dumitrescu and Pach [DP04] have showed that any configuration of unit squares on the integer lattice can be reconfigured to any other such shape via two types of moves, while remaining connected throughout. "Connected" here means 4-connected, i.e., edge-edge connected. I ask the same question for a configuration of unit cubes, where connectivity must be via face-face connections. Such objects, formed by face-to-face gluing of unit cubes, are called *polycubes*. The two moves permitted are exactly the Dumitrescu-Pach moves, shown in Figure 6, except now available parallel to any of the three coordinate planes. Perhaps applying the Dumitrescu-Pach algorithm to each xy-layer of cubes will help, but even if this maintains connectivity, in general cubes must be transfered between layers.

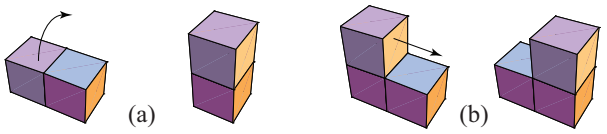


Figure 6: Cube moves based on Dumitrescu-Pach square moves.

Update. Zachary Abel and Scott D. Kominers [AK08] have recently announced a solution to this problem in the affirmative and have given a generalization to configurations of hypercubes of any dimension. Their method uses iterative relocation of modules from a configuration V to form a canonical chain at a distinguished module of V. They efficiently locate a module on the boundary of V which can either be relocated to the canonical chain immediately or can be relocated after recursive modification of the interior V.

Specifically, their methods prove the existence of a module x on V's outer boundary such that if $V \setminus \{x\}$ is not connected then $V \setminus \{x\}$ consists of exactly two components, one of which is disjoint from the outer boundary of V. Furthermore, this module x may be located quickly. Indeed, the solution yields an algorithm which requires at most $O(n^2)$ calculation time and at most $O(n^2)$ moves; this is asymptotically optimal.

References

[AK08] Zachary Abel and Scott D. Kominers. Pushing hypercubes around. Preprint, `arXiv:0802.3414`.

[DP04] Adrian Dumitrescu and Janos Pach. Pushing squares around. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, pages 116–123, 2004.

Surface Flips
Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Although this problem can be stated with more generality, I choose to pose it only for a polycube object P as defined in the previous problem. For any portion S of the surface P that constitutes a topological disk, and whose boundary is a cycle C of edges of P all lying in one plane Π, we define a *surface flip* as reflecting S through Π, as long as this operation maintains weak simplicity of the resulting surface P'. See Figure 7.

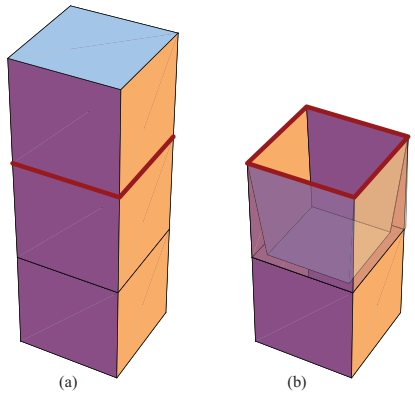


Figure 7: Surface flip about red 4-cycle.

Note that, whereas the previous problem preserved the volume, this move preserves the surface area. Also, the combinatorial structure of the surface does not change under these surface flips.

Characterize the class of polycube shapes that are connected by surface flips. In particular, are all shapes with the same combinatorial type connected?

Polycube Pops
Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Again this problem is restricted to polycubes, and the moves preserve the surface area, but now they will alter the combinatorial structure. My goal is to define a set of moves that will be able to convexify a polycube, or convert to some other canonical form, in a manner that will serve in some sense as a generalization of the vertex pop moves for unit orthogonal polygons (i.e., *polyominoes*) explored in [ABB+07].

There are five moves, two primary moves, and three moves concerned with collocated faces, which we will call "fences" (the analog of "pins" in 2D). They are named as follows:

1. Corner pop.
2. U-pop.
3. Fence walk.
4. Fence pop.
5. Fence corner pop.

The two primary moves are illustrated in Figure 8. The first move, a *corner pop*, is the one most closely inspired by a vertex pop: three faces incident to the corner of a cube are replaced by the three other faces of that cube. The second move, a *U-pop*, also replaces three faces of a cube by three others. Note that both moves preserve surface area.

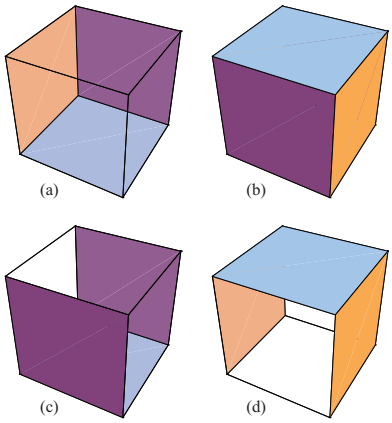


Figure 8: Two "pop" moves: (a,b) Corner pop; (c,d) U-pop.

To complete the definition of these moves, we need to specify how the adjacent faces are connected. This is shown in Figure 9, which makes it clear that in an abstract network of quadrilaterals forming the surface, the two moves replace the "wirings" internal to a 6- or 8-cycle; all exterior connections remain in place.

To have hope of reaching an interesting canonical form, it is important to permit the surface to become *weakly simple*, with perhaps several faces lying back-to-back on top of one another. We call

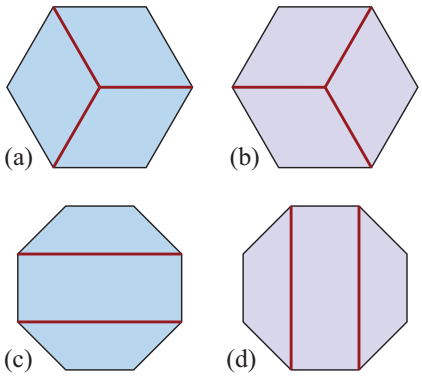


Figure 9: Combinatorial diagrams corresponding to the 3D moves in Fig. 8. (a,b) Corner pop; (c,d) U-pop.

two collocated faces a *fence*. It seems that three fence moves are needed. The *fence walk*, illustrated geometrically in Figure 10 and combinatorially in Figure 11(a,b), can be viewed as replacing three faces of a cube with three others, but this time two of the three faces are collocated.

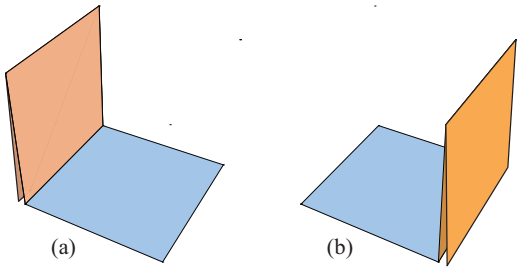


Figure 10: Fence walk.

A *fence pop* takes a connected tree of fences all perpendicular to the same plane Π, and reflects them through Π. This move involves no combinatorial change. Finally, a variation on this is a *fence corner pop*, which does involve a combinatorial change, shown in Figure 11(c,d).

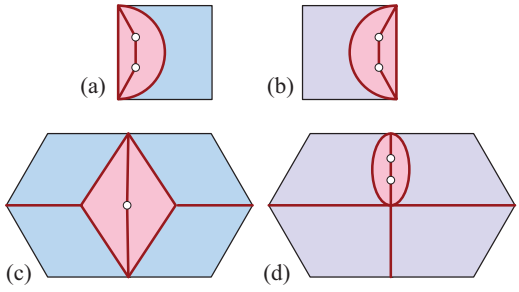


Figure 11: (a,b) Fence walk. (c,d) Fence corner pop.

Figure 12 shows an example of two shapes that can be connected by these polycube pop moves: two corner pops, one per dent, two fence corner pops, and finally a U-pop. Note that the dents in (a) of the figure consist of 4 faces each, and are replaced by 2 faces. The extra 4 faces gained are then enough to build the highest cube in (b).

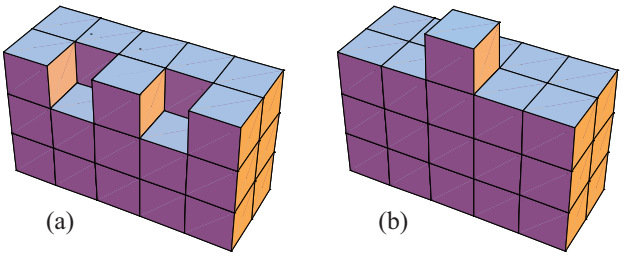


Figure 12: Object (a) can be converted to (b) via polycube pops.

There are (at least) two questions here: (1) Which class of shapes can be convexified (converted to an orthogonally convex polyhedron, e.g., Figure 12(b)) via the above five moves? (2) If not all shapes, then is there a natural set of moves that does suffice to connect all polycubes of the same surface area?

References

[ABB+07] Greg Aloupis, Brad Ballinger, Prosenjit Bose, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Ferran Hurtado, Stefan Langerman, Joseph O'Rourke, Perouz Taslakian, and Godfried Toussaint. Vertex pops and popturns. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 137–140, 2007.

Dice Rolling in a Rectangle

Erik Demaine
MIT
edemaine@mit.edu

When a rectangle R is fully labeled, and so there are no free nor blocked cells inside R, what is the complexity of deciding whether a cube can roll over R compatibly with the labels? For definitions, see [BBD+07]. That paper conjectures this restricted decision problem is solvable in polynomial time.

Two variants were suggested at the presentation: (1) What if R is labeled only with a subset of the six die labels? (2) What if the sides of R are glued to form a torus (left glued to right, top glued to bottom)?

References

[BBD+07] Kevin Buchin, Maike Buchin, Erik D. Demaine, Martin L. Demaine, Dania El-Khechen, Sándor Fekete, Christian Knauer, André Schulz, and Perouz Taslakian. On rolling cube puzzles. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 141–148, 2007.

Room Reconstruction from Point/Normal Data
Jack Snoeyink
Univ. North Carolina Chapel Hill
snoeyink@cs.unc.edu

Many sensors, such as the Delta-sphere constructed at UNC Chapel Hill, can send out beams and get range information for scattered points in a room. Suppose that your sensor gives not only a point, but also the normal to a plane when it hits a wall (perhaps by gathering individual point returns and using consensus to reduce error in wall positioning). You'd like to determine whether you have seen all the walls in your room. That is, from just the walls you have seen, can you construct a room that explains all observations? These questions are interesting even in the orthogonal case, where the walls must lie parallel to the coordinate planes.

This problem is relatively easy with a single sensor of known position in 2D: sort the sensed points radially and use the corresponding lines to construct a polygon, then check whether it is star-shaped with the sensor in the kernel. What about 3D, where the walls can have more complex shapes?

There are many variants on this problem: whether there are one or more sensors, whether you know each sensor(s) position, whether sensors return a few or all points (visibility polygon), whether the room must be orthogonal or simply connected, and whether the room reconstructed from the walls is unique.

Update: Biedl and Snoeyink have since been able to show that it is NP-hard to determine whether there is a unique reconstruction of an orthogonal polygon from a collection of 2D visibility polygons representing the information from several sensors.

Wireless Reflections
Boaz Ben-Moshe
College of Judea and Samaria
benmo@yosh.ac.il

Given a transmitter and receiver in an environment with barries (walls, etc.), and given an integer $c > 1$, find all the paths that go from transmitter to receiver by c billiard reflections (and perhaps also go through walls via transmission). In general, c will be small, say, $c < 100$. The problem arises in MIMO (Multiple-Input, Multiple-Output) communications.

Update: Ben-Moshe et al. have in some sense solved their problem and are implementing an algorithm as part of the Israeli Short Range Communication Consortium, http://www.isrc.org.il/index.asp.

Polygon that Projects as Chain
Prosenjit Bose
Carleton Univ.
jit@scs.carleton.ca

Is there a (closed) simple polygon in 3-space that projects to an open polygonal chain in three orthogonal directions? It is known that there is an open polygonal chain in 3-space that projects to a (closed) simple polygon in three orthogonal directions.

This problem was posed by Jack Snoeyink at an earlier CCCG, reporting a problem originally posed by Claire Kenyon.

Stretch Factor for Points in Convex Position
Prosenjit Bose
Carleton Univ.
jit@scs.carleton.ca

The *stretch factor* for a geometric graph G is the maximum, over all vertices u and v in G, of the ratio of the length of the shortest path from u to v in G to the Euclidean distance between them, $|u - v|$. If G has a stretch factor of t, it is called a *t-spanner*. Chew conjectured that the Delaunay triangulation is a t-spanner [Che89] for some constant t. Dobkin et al. [DFS90] established this for $t = \pi(1+\sqrt{5})/2 \approx 5.08$. The value of t was improved to $2\pi/(3\cos(\pi/6)) \approx 2.42$ by Keil and Gutwin [KG92], and further strengthened in [BM04]. Chew showed that t is $\pi/2 \approx 1.57$ for points on a circle, providing a lower bound. "It is widely believed that, for every set of points in $\mathbb{R}^2$, the Delaunay triangulation is a $(\pi/2)$-spanner" [NS07, p. 470].

This suggests the following special case: for points S in convex position (i.e., every point is on the hull of S), is the Delaunay triangulation of S a $(\pi/2)$-spanner?

References

[BM04] P. Bose and P. Morin. Online routing in triangulations. *SIAM Journal on Computing*, 33(4):937–951, 2004.

[Che89] L. Paul Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

[DFS90] David P. Dobkin, Steven J. Friedman, and Kenneth J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990.

[KG92] J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992.

[NS07] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

Polygonal Chain Simplification with Small Angle Constraints

Ovidiu Daescu* Anastasia Kurdia†

Abstract

We consider the problem of simplifying an n-vertex polygonal chain with *small* angle constraints in $\mathbb{R}^2$ and $\mathbb{R}^3$, thus closing the gap on the range of angles left in previous work on the problem. Specifically, we show that the *min-#* version of the polygonal chain simplification problem with small angle constraints can be solved in $O(n^2)$ time and space in $\mathbb{R}^2$, and in $O(n^2 \log^2 n)$ time, $O(n^2)$ space in $\mathbb{R}^3$.

1 Introduction

The problem of simplifying a polygonal chain with angle constraints was introduced in [6], and it was defined as follows: Given a polygonal chain $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$ or $\mathbb{R}^3$, find another chain $P' = (p_{i_1}, p_{i_2}, \ldots, p_{i_m})$ such that (1) $1 = i_1 < i_2 < \ldots < i_m = n$; (2) the tolerance zone criterion is satisfied for a given tolerance $\varepsilon \geq 0$: for every $j = \overline{1, m-1}$ the vertices of the subpath $(p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P are within distance ε from the line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P'; (3.min) the turn angle between any two consecutive line segments $\overline{p_i p_j}$ and $\overline{p_j p_k}$ of P' is at least a specified value $\delta(\overline{p_i p_j}) \in [0, \pi)$ (*min turn angle problem*); or (3.max) the turn angle between any two consecutive line segments $\overline{p_i p_j}$ and $\overline{p_j p_k}$ of P' is at most a specified value $\delta(\overline{p_i p_j}) \in [0, \pi)$ (*max turn angle problem*).

The problem was solved in [6] only for a subset of ranges of the turn angle δ, specifically for $\delta \in [0, \pi/2)$ for the (3.min) constraint and for $\delta \in [\pi/2, \pi)$ for the (3.max) constraint. Solving the problem when $\delta \in [\pi/2, \pi)$ for (3.min) and $\delta \in [0, \pi/2)$ for (3.max) remained open. In this paper we close the gap in the range of δ.

Let $ray(p_i p_j)$ be the ray originating at p_j, extending the line segment $\overline{p_i p_j}$ to infinity and not containing $\overline{p_i p_j}$. The turn angle between two line segments $\overline{p_i p_j}$ and $\overline{p_j p_k}$ is defined as the smallest angle needed to rotate the ray $ray(p_i p_j)$ at p_j such that it overlaps with the line segment $\overline{p_j p_k}$ [6] (see Fig. 1).

Using the tolerance-zone error criterion, the *min-#* problem (given an error-tolerance ε, minimize m) was solved in [6] in $O(n^2)$ time and space in $\mathbb{R}^2$, and in $O(n^2 \log n)$ time, $O(n^2)$ space, in $\mathbb{R}^3$. The *min-ε* problem (given m, minimize ε) was solved in $O(n^2 \log n)$ time and $O(n^2)$ space in $\mathbb{R}^2$, and in $O(n^2 \log^3 n)$ time and $O(n^2)$ space in $\mathbb{R}^3$. These bounds match the best known bounds for the unconstrained polygonal chain simplification problem with the tolerance-zone criterion.

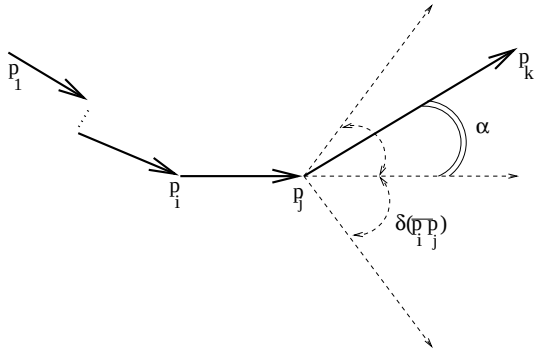


Figure 1: Angle constraint for the max turn angle problem in $\mathbb{R}^2$: α is a turn angle between $\overline{p_i p_j}$ and $\overline{p_j p_k}$; the edges $\overline{p_i p_j}$ and $\overline{p_j p_k}$ satisfy the angle constraint $\delta(\overline{p_i p_j})$.

A line segment $\overline{p_i p_j}$, $1 \leq i < j \leq n$, is a valid approximating segment (also called an ε-approximation segment) if the vertices $p_{i+1}, \ldots, p_{j-1}$ of P are within distance ε of $\overline{p_i p_j}$.

The algorithmic approach in [6] is to construct a directed acyclic graph G_p on P containing all valid approximating segments for the corresponding subpaths of P and to compute the shortest p_1-to-p_n path in G_p satisfying the angle constraint using a dynamic programming algorithm. G_p is constructed in $O(n^2)$ time in $\mathbb{R}^2$ [3, 5] and in $O(n^2 \log n)$ time in $\mathbb{R}^3$ [2] using the algorithms for the unconstrained version. The problem of finding the shortest p_1-to-p_n path in G_p satisfying the angle constraint is solved by reduction to a so called off-line ball exclusion search problem (OLBES) [6].

Results. In this work we close the gap in the range of possible turn angles by giving solutions for $\delta \in [\pi/2, \pi)$ for min turn angle problem and for $\delta \in [0, \pi/2)$ for max turn angle problem. As is the case in [6], the min turn and max turn problems can be solved using the same technique, so we discuss the max turn angle version only. Following the notation of [6], we describe the reduction of the shortest path problem in G_p to a special instance of the *off-line ball inclusion search* problem (OLBIS),

*Department of Computer Science, University of Texas at Dallas, daescu@utdallas.edu. Daescu's research was partially supported by NSF grant CCF-0635013.

†Department of Computer Science, University of Texas at Dallas, akurdia@utdallas.edu

defined in Section 2. We then give solutions to the 1-dimensional (1D) and 2-dimensional (2D) OLBIS problems.

Surprisingly, in $\mathbb{R}^3$, the time complexity of the solution for small turn angle constraints (maximum turn angle $\delta \in [0, \pi/2)$) seems to be inherently higher by an $O(\log n)$ factor than that for large angle constraints (maximum turn angle $\delta \in [\pi/2, \pi)$) due to the difference in time complexity for querying the associated data structures. Specifically, we solve the *min-#* problem in $\mathbb{R}^3$ in $O(n^2 \log^2 n)$ time with $O(n^2)$ space. Our solution for small angle constraints in $\mathbb{R}^2$ matches the $O(n^2)$ time and space complexities of that in [6].

2 Preliminaries

We use the notations in [6]. Let $ACSP_j(k)$ denote the angle-constrained shortest path from p_1 to p_k in G_p such that the last segment of $ACSP_j(k)$ is $\overline{p_j p_k}$. At the end of iteration i, $ACSP_j(k)$ is available for $j = \overline{1, i}$ and $k = \overline{i+1, n}$. At iteration $i+1$, from the available $ACSP_j(i+1)$, $j \in 1, 2, \ldots, i$, $ACSP_{i+1}(k)$ is computed for every $k = \overline{i+2, n}$.

At iteration $i+1$ consider $ACSP_j(i+1)$, for some $j \in 1, 2, \ldots, i$. The line segment $\overline{p_j p_{i+1}}$ is an incoming edge at p_{i+1} in G_p. Let p_{i+1} be the center of a unit sphere S_{i+1}. Let $Cone(j, i+1)$ be the cone of directions at p_{i+1} satisfying the angle constraint for $\overline{p_j p_{i+1}}$. An outgoing edge $\overline{p_{i+1} p_k}$, with $i+1 < k$ and $k \in i+2, i+3, \ldots, n$, can succeed $\overline{p_j p_{i+1}}$ to extend an angle constrained (shortest) path in G_p only if $\overline{p_{i+1} p_k}$ is contained within $Cone(j, i+1)$.

Each nonempty $Cone(j, i+1)$ at p_{i+1}, $j < i+1$, is intersected with S_{i+1}; the resulting ball (spherical cap) is assigned a weight equal to the length of $ACSP_j(i+1)$ (the number of edges of $ACSP_j(i+1)$). For each outgoing edge $\overline{p_{i+1} p_k}$ at p_{i+1}, the ray supporting the line segment $\overline{p_{i+1} p_k}$ is also intersected with S_{i+1}. We have at most i weighted balls and at most $n-i-1$ points. For each point we need to find a minimum-weight ball on S_{i+1} such that the point is contained by the ball. Sorting the balls by their weight and adding the points at the end of the resulting sequence we obtain an instance of the OLBIS problem:

OLBIS (Off-Line Ball Inclusion Search): Given a sequence $\mathcal{E} = (e_1, e_2, \ldots, e_n)$ such that each e_i is either a ball B_i or a point p_i, for every point p_k find the smallest-index ball $B_j \in \{e_1, e_2, \ldots e_{k-1}\}$ such that $p_k \in B_j$, or report no such ball exists.

Note that the OLBIS problem is a more general problem, since in the chain simplification problem the points appear after all the balls in $\mathcal{E}$.

The main differences between our solutions and those in [6] are in the construction and querying of the data structures associated with the OLBIS problem, versus those for the OLBES problem.

3 The $\mathbb{R}^2$ Problem

In $\mathbb{R}^2$, the cone projections correspond to intervals (1-dimensional balls). The 1-dimensional (1D) OLBIS problem can be solved by a simple greedy algorithm in $O(n \log n)$ time (see also [1] for the solution to a more general problem). Specifically, we first sort all points in the sequence. Then, starting with the first interval in $\mathcal{E}$, we locate the left endpoint of that interval in the sorted list of points and advance along the list until a point with value greater than the right endpoint of the interval is found. For each encountered point p, if the index of p in $\mathcal{E}$ is greater than the index of the interval we report p as included in that interval. Otherwise there is no interval that contains p and has index smaller than that of p. The points visited are removed from the sequence and the procedure is repeated with the next disk in $\mathcal{E}$.

Lemma 1 *Given a sequence $\mathcal{E}$ of n intervals (1-dimensional balls) and n points on the real line, the OLBIS answers for all points in $\mathcal{E}$ can be determined in $O(n \log n)$ time and $O(n)$ space.*

For the special case that arises in the min-# problem we can actually do better.

Lemma 2 *For a sequence $\mathcal{E}$ of n intervals (1-dimensional balls) and n points on the real line such that (i) the left and right endpoints of the intervals form sorted sequences and (ii) the points appear after all the intervals in $\mathcal{E}$ and are sorted by their values, the OLBIS answers for all points in $\mathcal{E}$ can be found in $O(n)$ time and space.*

Proof. First discard all the points that appear before the leftmost endpoint of the first interval, as they do not belong to any interval in $\mathcal{E}$. Then fix the left endpoint of the first interval I_1 and move along the real line until the right endpoint of I_1 is found; during this scan, enqueue the left endpoints of other intervals as they are encountered. I_1 is a minimum-index interval including all the points in $\mathcal{E}$ encountered so far. Next, dequeue the right endpoint r_2 of the next interval and report all query points encountered before reaching r_2 as belonging to the second interval. If the queue is empty and there are query points that have not yet been reported, advance to the first left endpoint available while discarding all encountered query points. This procedure is repeated until all query points are processed. It requires one pass along the input sequence and takes $O(n)$ time and space. □

We then have the following result for the polygonal chain simplification problem.

Theorem 3 *The polygonal chain simplification problem with small angle constraints in $\mathbb{R}^2$ can be solved in $O(n^2)$ time and space.*

4 The $\mathbb{R}^3$ Problem

We start by addressing the OLBIS problem. The key idea is to decide whether a query point is inside the union of a set of disks. The combinatorial complexity of the boundary of the union of n disks in $\mathbb{R}^2$ is known to be $O(n)$ [9], and it can be computed in $O(n \log n)$ time. However, as we will see below, there is no need to explicitly compute the union of disks to answer the queries. To decide whether a point is inside the union of disks, we use a standard transform that lifts a disk onto the unit paraboloid [7]. The image on the paraboloid of the circle bounding the disk defines a plane. A point is inside the disk if and only if its image on the paraboloid (the point lifted to the paraboloid) is below the plane. We call the halfspace defined by this plane and containing the image of the points inside the disk a proper halfspace. Consider the union of the corresponding proper halfspaces. Its intersection with the paraboloid and projected back to the plane gives the union of disks. The complement of this union is the intersection of a set of halfspaces, that is, a convex polytope. A point is inside the union of disks if it is outside this convex polytope.

Lemma 4 *Given a sequence $\mathcal{E}$ of n disks and n points, the OLBIS answers for all points in $\mathcal{E}$ can be determined in $O(n \log^2 n)$ time and $O(n \log n)$ space.*

Proof. We construct a complete binary tree T. Each leaf of T is associated with the complement of the proper halfspace for a disk D_j from $\mathcal{E}$ in order (i.e., the halfspace for D_1 is associated with the leftmost leaf, the second leaf from left is for the halfspace for D_2, and so on). At each internal node v we compute and store $I(v)$, the intersection of the halfspaces stored at the leaves of the subtree of T rooted at v. We also compute and store the index $i(v)$ of the rightmost leaf of the subtree rooted at the left child of v.

The intersection of half-spaces associated with the root of T is computed in divide-and-conquer fashion using the linear-time algorithm for intersecting three-dimensional convex polyhedra [4] and corresponds to a bottom-up computation in T. The intermediate results of the divide-and-conquer algorithm are intersections of halfspaces associated with internal nodes of T and are stored at those nodes. Thus, over all internal nodes $v \in T$, we can compute and store the intersections of halfspaces in $O(n \log n)$ time using $O(n \log n)$ space. Assigning $i(v)$ to each node $v \in T$ can be done in $O(n)$ time by a simple traversal of T. An n vertex polyhedron can be preprocessed for $O(\log n)$ time point inclusion queries in $O(n)$ time and space [8]. We preprocess the polyhedra associated with the nodes of T for point location queries.

The OLBIS queries for the points in $\mathcal{E}$ are answered as follows (all points are queried simultaneously). Let *root* be the root of T, let $left(root)$ be the left child of the root, and let $right(root)$ be the right child of the root. We select the points that fall outside $I(root)$ (i.e. the points that are inside of at least one disk) and partition the selected points based on inclusion in $I(left(root))$. The points that are outside $I(left(root))$ are given to the search on the subtree rooted at $left(root)$; the remaining points become input for the search on the subtree rooted at $right(root)$ with the only reservation that if a query point in the input set for $right(root)$ has a smaller index than the index of the rightmost leaf in the subtree rooted at $left(root)$ then this point does not belong to any disk preceding it in $\mathcal{E}$ and can be dropped.

Using point location queries it takes $O(\log n)$ time to answer an inclusion query in $I(v)$ at each level of T. There are $O(\log n)$ levels in T, giving $O(\log^2 n)$ query time for one point. Then, the general 2D OLBIS problem can be solved in $O(n \log^2 n)$ time with $O(n \log n)$ space. □

Notice that the extra $O(\log n)$ factor in time complexity, when compared to [6], is due to the query processing. We then obtain the following result for the polygonal chain simplification problem.

Theorem 5 *The* min-# *problem with angle constraints in $\mathbb{R}^3$ can be solved in $O(n^2 \log^2 n)$ time and $O(n^2)$ space.*

For the related *min-ε* problem we obtain:

Theorem 6 *The* min-ε *problem with angle constraints can be solved in $O(n^2 \log n)$ time and $O(n^2)$ space in $\mathbb{R}^2$ and in $O(n^2 \log^4 n)$ time, $O(n^2)$ space in $\mathbb{R}^3$.*

5 Conclusions

In this paper we presented solutions for the polygonal chain approximation problem with *small* angle constraints, thus closing the gap on the range of angles left in previous work [6]. Our solution for small angle constraints in $\mathbb{R}^2$ matches the $O(n^2)$ time and space complexities of that in [6].

Surprisingly, in $\mathbb{R}^3$, the time complexity of the solution for small turn angle constraints is higher by an $O(\log n)$ factor than that for large angle constraints [6] due to an extra $O(\log n)$ factor in processing queries. We leave the elimination of the extra $O(\log n)$ factor in this case as an open problem.

References

[1] Pankaj K. Agarwal, Lars Arge, and Ke Yi. An optimal dynamic interval stabbing-max data structure? In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 803–812, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

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

[3] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *ISAAC '92: Proceedings of the Third International Symposium on Algorithms and Computation*, pages 378–387, London, UK, 1992. Springer-Verlag.

[4] Bernard Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[5] Danny Z. Chen and Ovidiu Daescu. Space-efficient algorithms for approximating polygonal curves in two dimensional space. *Int. J. Comput Geometry Appl.*, 13(2):95–111, 2003.

[6] Danny Z. Chen, Ovidiu Daescu, John Hershberger, Peter M. Kogge, Ningfang Mi, and Jack Snoeyink. Polygonal path simplification with angle constraints. *Computational Geometry*, 32(3):173–187, 2005.

[7] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*.

[8] David P. Dobkin and David G. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 400–413, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[9] K. Kedem, R. Livine, J. Pach, and M. Sharir. On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

Memory Requirements for Local Geometric Routing and Traversal in Digraphs

M. Fraser * E. Kranakis † J. Urrutia ‡

Abstract

Local route discovery in geometric, connected, *undirected* plane graphs is guaranteed by the Face Routing algorithm. The algorithm is local and geometric in the sense that it is executed by an agent moving along a network and using at each node only information about the current node (incl. its position) and a finite number of others (independent of graph size). Local geometric traversal algorithms also exist for undirected plane graphs. In this paper we show that no comparable routing or traversal algorithms exist for the class of strongly connected plane *directed* graphs (digraphs). We construct a class of digraphs embedded in the plane for which either local routing or local traversal requires $\Omega(n)$ memory bits, where n is the order of the graph. We discuss these results in light of finding a suitable model for mobile ad hoc networks with uni-directional edges, showing in the extended version of this paper that digraphs for which the $\Omega(n)$ lower bound holds occur even in the class of embedded digraphs arising out of a very conservative model.

1 Introduction

Face Routing was proposed in [9] as an algorithm that accomplishes discovery of routes locally in geometrically embedded, connected, undirected planar graphs. It guarantees delivery in time $O(n)$ where n is the order of the graph. Since its introduction in 1999, there have been various studies extending the class of graphs over which variations of this algorithm can be applied. In the 2D *undirected* case, these are enough to handle most graphs occuring as models of 2D mobile adhoc networks: Quasi-Planar graphs, Unit Disk Graphs (see survey in [11]) and so called Quasi-Unit Disk Graphs (with radial coefficient $d > \frac{1}{\sqrt{2}}$).

Traversal algorithms have also been well studied for undirected graphs in the plane. For example, for planar embedded graphs [4] and quasi-planar subdvisions [3] local geometric algorithms are known.

In the 3D case relatively little has been done until now. Recently M. Fraser extended Face Routing to undirected graphs embedded on known surfaces of higher genus [8] and Durocher et al. [5] extended it to Unit Ball Graphs (UBG's) with nodes bounded within a relatively thin slab of space (of thickness at most $1/\sqrt{2}$). Also a recent paper by R. Wattenhofer [6] proposes a partially randomized routing algorithm for 3D ad hoc networks.

There has also been little progress on the 2D *directed* case. Some special classes of 2D directed graphs for which local geometric routing algorithms are known include Eulerian and Outerplanar *digraphs* [2]. The question of existence of local geometric algorithms for general geometric digraphs has remained open.

It is important to note that there are several variations in the use of the term **"local"** in the ad-hoc community. In this paper, by a *local algorithm* we mean a deterministic algorithm in which an agent moving along the network uses at each node only information stored at that node concerning the node itself and its neighbours (including current position) together with a certain amount M of memory which is carried as overhead in the message. No further information regarding the rest of the network is available to the agent, e.g. neither the number of nodes, where they are located, the topology of the network, nor any other global information. The agent is not allowed to alter the state of a node.

By *local routing algorithm*, we mean a local algorithm as defined above where the agent should reach a node t started from a node s, given the position or ID of t.

In Face Routing, messages are allowed to carry with them the ID's (or typically the positions) of sender and destination as well as two other nodes and a number representing a distance. This means M has typically been restricted to $O(\log n)$ as part of the definition of "local". Upon arrival to a node, a message can use the local information stored at a node plus the information it carries along.

We show that planar embedded directed graphs (digraphs), unlike their undirected counterparts, do not admit local geometric routing *or* traversal algorithms

*Mathematics Dept., University of Colima, Colima, MEXICO. fraser.maia@gmail.com

†Schoool of Computer Science, Carleton University, Ottawa, CANADA. Research supported in part by NSERC and MITACS grants. kranakis@scs.carleton.ca

‡Inst. of Mathematics, National Autonomous University of Mexico (UNAM), Mexico City, MEXICO, Research supported in part by MTM2006-03909 (Spain) and CONACYT of Mexico, Proyecto SEP-2004-Co1-45876. urrutia@matem.unam.mx

with carried memory $M \in O(\log n)$ (for non-geometric digraphs the corresponding result for traversal was shown by Fraigniaud and Ilcinkis [7]). We show this by constructing a class $\mathcal{C}$ of embedded digraphs for which either geometric traversal or route discovery requires $\Omega(n)$ memory bits.

It is important to notice that in practice, many wireless and ad-hoc networks *do have* oriented links. If a network has transmitters with different broadcast powers, it may happen that a node can receive messages from a node to which it cannot send.

In the extended version of this paper we describe a Face-Routing-like routing algorithm which is correct for plane digraphs and is executed by an agent carrying $O(n)$ memory.

Our main objective in this paper is to prove the following result:

Theorem 1 *There is a class of bounded degree (i.e., both in-degree and out-degree) strongly connected embedded digraphs in the plane for which no deterministic local traversal or routing algorithm (even a geometric one) exists. In particular, if n is the number of vertices of a graph in this class then every traversal or routing algorithm on the graph requires $\Omega(n)$ bits of memory.*

Here we use the term *traversal* in its weakest sense: namely requiring only that all nodes must be visited and making no assumption about return to start.

The result of Fraigniaud and Ilcinkas [7] addressed only *non-geometric* graphs and so hope remained that the use of geometry might make possible an $O(\log n)$ algorithm for at least planar embedded digraphs. We settle this issue in the negative.

We remark that in the absence of planar geometry the routing problem is, loosely speaking, equivalent to that of traversal since an adversary may connect a destination node to any other graph node; however, this is no longer the case for planar embedded graphs, since not all nodes are reachable from a given destination position without introducing edge-crossings.

We also note that while Face Routing provides a linear time, $O(\log n)$ memory local routing algorithm for planar embedded, undirected graphs, when it was introduced in 1999 there was by contrast no known $O(\log n)$ local routing algorithm for abstract undirected graphs. In 2005, one was established by Reingold (using universal exploration sequences, [10]). But, not only is this method much more complicated than Face Routing, its time still remains polynomial with high exponent. Planar geometry thus has a significant impact in reducing the complexity of the routing problem in the undirected case. The point of the present paper is to show that the benefit of geometry is *insuficient* to reduce memory requirements of local routing to logarithmic in the directed case.

2 Main Result

To construct a class with the properties stated in Theorem 1, we define for each $n \in \mathbb{N}$ an embedded digraph, called a *random lock*, which is a "randomized" embedding of a variation on the combination lock (combination locks have unbounded in-degree, i.e., $\Theta(n)$, see [7], and this variation just bounds the degree by merging wigs, albeit in a random way). To this end we first define a kinked embedded lock.

Definition 2.1 [Kinked Embedded Lock] *Let $x = x_1x_2\cdots x_{n-1}$ be any bit string of length $n-1$ (notational warning: we start with index 1). Suppose the 1 bits of x occur for indices $i_0, \ldots, i_k$ and the 0 bits for indices $j_0, \ldots, j_K$. A kinked embedded lock determined by x is an embedded digraph $\varphi(H_x)$ where $H_x = (\bar{V}_x, \bar{E}_x \cup \{e\})$ is an abstract digraph with*

- *set of vertices $\bar{V}_x = \{u_0, u_1, u_2, \ldots, u_{n-1}, u_n, w, v_1, v_2, \ldots, v_{n-1}\}$,*
- *set of directed edges $\bar{E}_x = E'_n \cup E''_n$ where, $E'_n = \{(u_i, u_{i+1}) : 0 \le i \le n-1\} \cup \{(u_i, v_i) : 1 \le i \le n-1\}$ and $E''_n = \{(u_n, v_{i_k}), (u_n, v_{j_K})\} \cup \{(v_{i_q}, v_{i_{q-1}}) : 1 \le q \le k\} \cup \{(v_{j_q}, v_{j_{q-1}}) : 1 \le q \le K\} \cup \{(v_{i_0}, u_0), (v_{j_0}, u_0)\}$[1],*
- *and one bi-directional edge $e = (u_n, w)$,*

and φ is any embedding $\varphi : H_x \to \mathbb{R}^2$ satisfying the following recursive relation:

- *$\varphi(u_0) = (0,0), \varphi(u_1) = (1,0)$ and*
- *for $1 \le i \le n-1$ if u_i has coordinates (a,b) then:*

  $\varphi(u_{i+1}) = (a+1, b+1), \varphi(v_i) = (a+1, b-1)$ *for $x_i = 0$, and*

  $\varphi(u_{i+1}) = (a+1, b-1), \varphi(v_i) = (a+1, b+1)$ *for $x_i = 1$,*

and $\varphi(w) = (n+1, 0)$, with all edges being straight line segments. We denote by K_x the kinked embedded lock determined by x.

Note that the order of the kinked embedded lock K_x determined by an $(n-1)$-bit string x is $|K_x| = 2n+1$. It is easily checked that φ as described is an embedding (i.e. that there are no edge crossings).

An example of the requirements on φ for the string $x = 0111$ is depicted in Figure 1. This figure includes only the edges $\varphi(E'_n)$.

The full embedded digraph $\varphi(H_x)$ is obtained from the one shown in Figure 1 by adding embedded directed edges from $\varphi(E''_n)$, namely adding edges "all along the top" and "all along the bottom" (to connect subsequent

[1] where undefined elements are assumed omitted

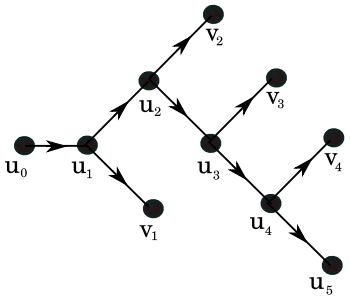


Figure 1: An example of the requirements on $\varphi(E'_n)$ for the string $x = 0111$.

upper vertices v_{i_q} and subsequent lower vertices v_{j_q} as given by E''_n) and then adding the bi-directional edge e connecting w to u_n. This is depicted in Figure 2. Note that the position of $\varphi(w)$ does not depend on the actual bit string x, just on n.

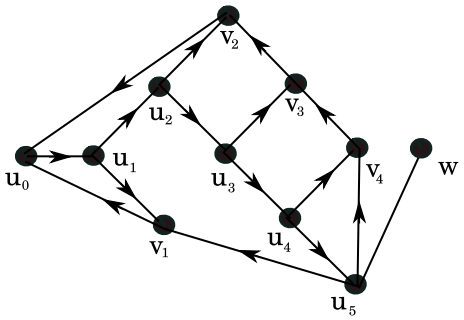


Figure 2: The kinked embedded lock K_x determined by the bit string $x = 0111$.

We will now show that if we use random bit strings to construct kinked embedded locks we obtain a class of geometrically embedded digraphs – random locks – which satisfy the requirements of the theorem. The idea is that any traversal or routing algorithm, $\mathcal{A}$, on such graphs must be able to reach w from u_0 knowing at most the positions of these two points and local information. To do so, it must be able to reach u_n using only this data. This allows one to generate the underlying random bit string and the desired lower bound follows.

Proof. *(of Theorem 1)* Let x be an infinite Kolmogorov random (i.e. algorithmically random) bit string $x = x_1x_2\ldots$ (notational warning: we start with index 1). Then for any $n \in \mathbb{N}$, $x^n := x_1x_2\cdots x_n$ has Kolmogorov complexity at least n. In other words, any algorithm which can generate this string has space complexity at least n.

Let $\mathcal{C} = \{K_{x^n} : n \in \mathbb{N}\}$ be the set of kinked embedded graphs determined by all finite initial substrings of x. $\mathcal{C}$ consists of bounded-degree embedded digraphs. We call these *random locks*. Recall that $|K_{x^{n-1}}| = 2n + 1$, so the order of these graphs is linear in n.

Assume that $\mathcal{A}$ is a correct geometric traversal or routing algorithm for $\mathcal{C}$ using memory $M(n)$ bits of (transported and working) memory for the random lock $K_{x^{n-1}} \in \mathcal{C}$ determined by the $(n-1)$-bit string x^{n-1}. Since the size of these graphs is linear in n we need only show $M(n) \in \Omega(n)$.

Starting at u_0, $\mathcal{A}$ must in particular be able to reach w using only local geometric data (coordinates of current node and its neighbours) *plus* $M(n)$ bits of memory, which may include local geometric data gathered along its route (coordinates of vertices visited and their neighbours) or even coordinates of w (in the case of a routing algorithm) . Note that knowing the coordinates of w just corresponds to knowing $n + 1$.

To reach w from u_0, $\mathcal{A}$ must eventually – i.e. at some time T – be at u_1 and then follow exactly the path $u_1, u_2, \ldots u_n, w$. To be precise, let T be the time just before computing an exit vertex at u_1 and let S_T be the state of $\mathcal{A}$ at this time, i.e. the stored memory at this moment including the current line number L_T that has just been executed at this time in the program P encoding $\mathcal{A}$. Storing S_T requires memory at most $M(n)$.

Let $\mathcal{G}$ be an efficient algorithm which given a pair of positive integers (x, y) and a bit b calculates $(x, y), (x+1, y+1-2b), (x+2, y+2-2b), (x+2, y-2b)$. We may assume $\mathcal{G}$ has space complexity linear in the size of (x, y). If we assume the numbers $x, y \leq n$, then the space complexity of $\mathcal{G}$ is $O(\log n)$. Note that $\mathcal{G}$ is essentially a *geometry simulator* for $\mathcal{A}$ as long as $\mathcal{A}$ travels along a path consisting of only u_i's. More precisely, whenever $\mathcal{A}$ is at some u_i for which it will next advance to u_{i+1}, if it passes to $\mathcal{G}$ the coordinates of u_i and the value 0 or 1 indicating its decision at u_i (to turn respectively up or down) then $\mathcal{G}$ will output the coordinates of u_{i+1} and of all its neighbours. This is in fact all the geometric data that the algorithm $\mathcal{A}$ can use to make its decision at u_{i+1}.

Using $\mathcal{A}$ and $\mathcal{G}$, we now define an algorithm $\mathcal{B}$ of space complexity $M(n) + O(\log n)$ which generates the bit string x^n. This includes the constant overhead to store instructions for all three algorithms (incl. passing of data between them, conversion of left/right instructions to bits etc...).

The algorithm $\mathcal{B}$ takes input $\mathtt{S_T}$, $\mathtt{F} = n - 1$. Its working data will include the coordinates of four points in the plane: $\mathtt{u_{curr}}$ and $\mathtt{u_{left}}$, $\mathtt{u_{rightUp}}$, $\mathtt{u_{rightDn}}$. It initializes these to $(1, 0), (0, 0), (2, 1), (2, -1)$ respectively, these being the coordinates of u_1, u_0 and the upper then lower of u_2, v_1. The algorithm then proceeds as follows:

1. Set $i = 1$. Place $\mathcal{A}$ in state $\mathtt{S_T}$.

2. Run $\mathcal{A}$ for a single iteration giving it local pseudo-geometric data $u_{\texttt{curr}}$, $u_{\texttt{left}}$, $u_{\texttt{rightUp}}$, $u_{\texttt{rightDn}}$ in order to generate a bit b encoding whether it will turn upwards or downwards ($b = 0$ or 1 respectively). This corresponds to deciding which of the two neighbours to the right is labelled u_2.

3. Output b. If $i \geq F$, exit; else increment i by one.

4. Call $\mathcal{G}$ with input $u_{\texttt{curr}}$ and b. Let its output be used to set $u_{\texttt{left}}$, $u_{\texttt{curr}}$, $u_{\texttt{rightUp}}$, $u_{\texttt{rightDn}}$ respectively. Go to 2.

Recall that when the algorithm $\mathcal{A}$ is started in state S_T using local geometric data from u_1, it performs exactly the run $u_1, u_2, \ldots, u_{n-1}$. Thus the output of $\mathcal{B}$ will be exactly the bit string x^n. Since the complexity of $\mathcal{B}$ is $M(n) + O(\log n)$, and x^n is Kolmogorov random we must have $M(n) \in \Omega(n)$. □

k-Locality We remark that a k-local version of Theorem 1 also holds: namely, even a geometric routing or traversal algorithm with access to geometric data k hops away from its current location (k, any fixed positive integer) requires $\Omega(n)$ memory bits. This is proved by inserting k extra vertices on each of the edges $\{(u_0, u_1), (u_i, v_i), (u_i, u_{i+1}) : i \in \mathbb{N}, 1 \leq i \leq n-1\}$ (i.e. k-subdividing these edges) in the random locks above and adjusting the geometry simulator accordingly so as to generate these extra points as well.

Remark In fact, it is easy to construct another class $\mathcal{C}'$ of digraphs which can be used to prove Theorem 1 and which consists of plane digraphs arising from wireless networks with just two different broadcast radii. We discuss this further in the extended version of this paper.

3 Conclusions

Since the introduction of Face Routing, considerable efforts have been made trying to extend it to wider families of networks. The robustness and simplicity of Face Routing make it a tantalizing model to strive for, as the computational requirements in network communications are thus greatly reduced, e.g. drastic reductions on the amount of traffic, and the elimination of costly mechanisms such as routing tables that have to be updated periodically [11]. It is worth mentioning here that the locality condition, as stated here, implies a robustness beyond existing modes; since a message is not aware of the existence of most nodes of a network, it is unaffected by their potential failures (as long as the network remains connected).

Especially, it would be desirable to achieve these properties of Face Routing for cellular networks in $\mathbb{R}^3$ or directed networks as these settings are a reality in telephony and sensor networks nowadays. Unfortunately, extending Face Routing beyond undirected networks in the plane has proved to be a formidable task, and all the evidence suggests it may not be possible. As far as we know, this is the first result that proves some *intrinsic* limitations of the model. The results presented here might give a hint as to why previous attempts have been unsuccessful, and might in fact dictate restrictions that should be imposed on wireless and, in general, communication networks to take full advantage of the benefits of Face Routing.

References

[1] P. Bose, P. Morin, I. Stojmenic, and J. Urrutia, *Routing with guaranteed delivery in ad hoc wireless networks*. In 3rd Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIALM '99), pp. 48-55 (1999).

[2] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, J. Urrutia, *Route Discovery with Constant Memory in Oriented Planar Geometric Networks*. Networks, Volume 48, Issue 1, pages 7-15.

[3] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, J. Urrutia *Traversal of a quasi-planar subdivision without using mark bits*. Journal of Interconnection Networks Vol. 5, Issue 4, pp. 395 - 407, 2004.

[4] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. *Simple traversal of a subdivision without extra storage*. International Journal of Geographic Information Systems, 11:359373, 1997.

[5] S. Durocher, D. Kirkpatrick, L. Narayanan, *On Routing with Guaranteed Delivery in Three-Dimensional Ad Hoc Wireless Networks*. In proceedings of ICDCN 2008.

[6] R. Flury and R. Wattenhofer, *Randomized 3D Geographic Routing*. In 27th Annual IEEE Conference on Computer Communications (INFOCOM), Phoenix, USA, April 2008.

[7] P. Fraigniaud and D. Ilcinkas, *Digraphs Exploration with Little Memory*. In 21st Symposium on Theoretical Aspects of Computer Science (STACS'04), LNCS 2296, pages 246-257, 2004.

[8] M. Fraser, *Local Routing on Tori*. In Proceedings of AD-HOCNOW, held in Morelia Mexico, Sep 24-26, 2007, pp. 1- 14, Springer LNCS 4686, 2007.

[9] E. Kranakis, H. Singh, J. Urrutia, *Compass Routing in Geometric Graphs*, in proceedings of 11th Canadian Conference on Computational Geometry, CCCG-99, pages 51-54, Vancouver Aug. 15-18, 1999.

[10] O. Reingold, *Undirected ST-Connectivity in Log-Space*, Proceedings of the 37th annual ACM symposium on Theory of Computing, pp. 376-385, ACM, 2005.

[11] J. Urrutia, *Local solutions for global problems in wireless networks*. Journal of Discrete Algorithms, 5, pp. 395-407, 2007.

A Distributed Algorithm for Computing Voronoi Diagram in the Unit Disk Graph Model

Yurai Núñez-Rodríguez[(1)] Henry Xiao[(1)] Kamrul Islam[(1)] Waleed Alsalih[(1)]
[(1)]Queen's University(yurai/xiao/islam/waleed@cs.queensu.ca)
School of Computing, Queen's University
Kingston, Ontario, Canada K7L 3N6

Abstract

We study the problem of computing Voronoi diagrams distributedly for a set of nodes of a network modeled as a Unit Disk Graph (UDG). We present an algorithm to solve this problem efficiently, which has direct applications in wireless networks. Comparing with some existing algorithms, our algorithm correctly computes the complete Voronoi diagram and uses a significantly smaller number of transmissions. Furthermore, useful geometric structures such as the Delaunay triangulation and the convex hull can be obtained through our algorithm.

1 Introduction

With the recent wave of research in wireless networks, geometric structures like the Voronoi Diagram (VD) have been studied in different computational platforms such as mobile devices and sensors [11, 10, 2]. These devices have limited battery power and usually cooperate with each other without a centralized control. Given a set of n points in the plane, it is known that the VD can be computed in $O(n \log n)$ time in a centralized fashion (see Fortune's algorithm for an example [6]). In a distributed setting, computing the VD introduces new challenges [4, 5, 10, 11, 2].

We investigate the problem of distributedly computing the VD of a network where the nodes are modeled as points in the plane. Our main goal is to compute the accurate VD while minimizing the communication cost. The communication cost is proportional to the number of transmissions between two adjacent nodes. We propose a distributed algorithm to compute the VD of a connected network. Our approach is purely based on cooperation among nodes. A preliminary version of our work has been announced in [1].

Throughout this paper, a network is modeled as a Unit Disk Graph (UDG). According to this model, an edge between two nodes v and w exists if and only if the Euclidean distance between v and w is not greater than one unit (normalized). We assume that the induced UDG is connected. The nodes are also assumed to know their geographic locations. This is usually achieved through GPS or other techniques [9, 7]. The communication of the network is not required to be synchronous.

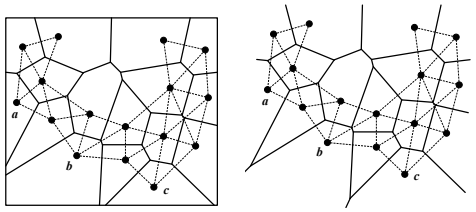


Figure 1: Voronoi diagrams of a set of nodes. Edges between nodes are represented by dashed lines. The bounded VD (left) and the complete VD (right).

In recent attempts to design distributed algorithms for computing the VD, Bash and Desnoyers [2] proposed an algorithm to compute the *bounded* VD (see Figure 1) utilizing the GPSR routing protocol ([3, 8]) for wireless sensor networks. The basic idea is to successively refine the approximations of the Voronoi cells upon discovery of other sensors in the network. Given a sensor s, their algorithm starts with the entire bounded region as the approximation of the Voronoi cell of s. A probing message is then sent to each of the vertices of the current Voronoi cell using GPSR, which will be delivered to a sensor t that is the nearest to the probed vertex. Sensor t replies to s and the current Voronoi cell of s is refined with respect to t. No more probes are sent by s once it becomes the nearest sensor to all its Voronoi vertices. Bash and Desnoyers' algorithm [2] is referred to as BD07[1] in this paper.

In the following, we describe our distributed algorithm for computing the VD and prove its correctness. We further discuss more practical environments where the correctness of our algorithm holds. Some of our simulation results in comparison with BD07 are presented in Section 4.

[1]Limited by the space, optimizations of BD07 are not reviewed here.

2 Distributed Computation of Voronoi Diagram

We propose a distributed algorithm, namely the *completely cooperative* (CC) algorithm, for computing the VD of a set of nodes in the plane. Recall that the network is modeled as a UDG. The nodes are assumed to be in general position[2].

2.1 The CC algorithm

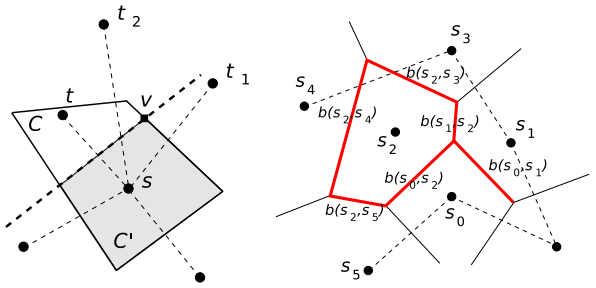
The basic idea behind the CC algorithm is that nodes do not need to send out queries to discover their Voronoi neighbours; instead, nodes are informed about possible Voronoi neighbours by other neighbours. We adopt the following terminology. Let S be a set of nodes embedded in the plane and let $G = (S, L)$ be the connected UDG induced by S, where $L \subseteq S \times S$ contains pairs of nodes that are within unit distance. Let $VD(G)$ be the VD of G. We refer to an element of S as a *node* and an element of L as a *link*, saving the terms *vertex* and *edge* for the corresponding elements in the VD. Similarly, we refer to nodes that share a link as *adjacent* and to nodes that share a Voronoi edge as *neighbours*.

Let s be a node that receives a message about a potential (Voronoi) neighbour, t, at some point during the computation. Then s computes the intersection of its current cell, C, with the half plane defined by the bisector between itself and t. We call this step the *refinement* of a cell. If the new cell C' resulting from the intersection is equal to C, t is ignored; otherwise ($C' \subset C$), t becomes a neighbour of s. In the latter case, new vertices appear on C' and some vertices of C fall outside of C'. Figure 2a illustrates the refinement process. Two adjacent vertices that fall outside of C', define a piece of bisector for a node t_2 that is then discarded. A new vertex v on C' is created by the intersection of the bisector between s and t, and the bisector between a certain node t_1 and s. Therefore, t and t_1 may be neighbours of each other since they have a common Voronoi vertex according to the cell of s. Consequently, s informs both t and t_1 about each other. This way, the information about possible neighbours flows towards the corresponding nodes until each node finds all its neighbours.

Initially, the cell of any node s is equal to the entire plane. Then all nodes broadcast their locations triggering the entire computation as explained above.

Algorithm 1 is a pseudocode description of the CC algorithm. In the description of the algorithm, node s has location $s.loc$, a field $s.cell$ that stores the description of its Voronoi cell, and a message queue $s.q$. The refinement of $s.cell$ with respect to t is done through $s.refine(t.loc)$. The method $send\_message(t_1.loc, t_2.loc)$ sends a message to node t_1 containing the location of t_2 and results in $t_2.loc$ being added to the message queue of t_1 (i.e., $t_1.q$). Vertices in $s.cell$ can be accessed through $s.cell.verts$. A vertex v of a Voronoi cell is equipped with a method $v.third(s.loc, t.loc)$ that returns the location of the third node associated to v that is neither s nor t.

[2]No three nodes are collinear and no four nodes fall on the same circle



(a) Cell Refinement (node s discovers node t)

(b) Disconnected Node

Figure 2: Illustration of the CC algorithm.

Algorithm 1: Completely Cooperative (CC)

```
// Initialize the cell
s.cell = ENTIRE_SPACE
// Broadcast the node location to all adjacent nodes
s.send_message( BROADCAST, s.loc )
// Process each (node) message in the queue
while( t.loc = s.q.get_message )
    old_Cell = s.cell
    s.cell = s.refine( t.loc )
    for each( v in s.cell.verts and not in old_Cell.verts )
      // Notify each pair of possible neighbours about each other
        s.send_message( t.loc, v.third( s.loc, t.loc ) )
        s.send_message( v.third( s.loc, t.loc ), t.loc )
    end
end
```

2.2 Proof of correctness

Theorem 1 *Let G be the induced UDG of a set of nodes in the plane. The CC algorithm computes the correct Voronoi cell of every node in G.*

Proof. The reader is referred to Figure 2b for a graphical description of this proof. It is not hard to see that the algorithm terminates after a finite number of steps given that every message sent is the result of the refinement of a cell whose area has decreased. Therefore, the computation ends as the approximations of the Voronoi cells converge to the correct cells.

In order to prove that the algorithm determines the correct cells, suppose, for the sake of contradiction, that the Voronoi cell corresponding to a node s_0, was not properly determined. This means that s_0 did not find at least one of its (Voronoi) neighbours. Let s_1 be a neighbour of s_0 that was not discovered by the application of Algorithm 1 to s_0. It would be a contradiction that $(s_0, s_1) \in L$ since adjacent nodes know about each other and must have been neighbours from the initial

refinements. Therefore, we assume that there is no link between s_0 and s_1.

Let $b(s_0, s_1)$ be the edge of $VD(G)$ corresponding to the bisector between s_0 and s_1. Note that VD edges are segments, lines, or semilines. $b(s_0, s_1)$ can be a line only if $|S| = 2$, but since there is no link between s_0 and s_1, G would not be connected, which is a contradiction. Therefore, $b(s_0, s_1)$ must be a segment or a semiline. In both cases $b(s_0, s_1)$ has at least one end point $(v(s_0, s_1, s_2))$ that is a Voronoi vertex of the cells associated to s_0, s_1, and a third node s_2. The CC algorithm guarantees that s_2 informs s_0 and s_1 about each other once the corresponding bisectors have been considered and the intersection point $(v(s_0, s_1, s_2))$ has been found. Since s_0 and s_1 were not informed about each other, one of three possible cases must have occurred: s_2 did not find s_0, or s_1, or both. Without loss of generality, we assume that s_2 did not find s_1. The same reasoning applied to s_0 and s_1 can be applied to s_1, s_2 and a third node $s_3 \neq s_0$. This process can be repeated until one of two stop conditions is satisfied: (1) a cycle is created when a vertex is involved twice, (2) a semiline bisector in $VD(G)$ is reached. If this process ends with a semiline between two unbounded cells, the same procedure is applied starting at the other endpoint of $b(s_0, s_1)$, if any. This process again ends because of condition (1) or (2).

In the end, this process leads to a cycle or path P consisting of missing Voronoi edges that partitions the plane into two disjoint regions and is not crossed by any link between neighbours. It is not hard to see that if P is not crossed by any link between neighbours, it can not be crossed by any other link, given that G is the induced UDG. Therefore, G would have two disconnected subgraphs (one in each region) which contradicts the initial assumption. □

2.3 Optimizations

The CC algorithm is described in its simplest form. Some optimizations can be introduced to make it more efficient. First of all, before the initial refinements, every node s broadcasts its adjacency list. Every message that involves s includes its adjacency list. If a node r, that is about to inform two nodes s and t about each other, finds s (resp. t) in the adjacency list of t (resp. s), no notification is sent. This remarkably reduces the number of transmissions. However, some lists of adjacency can be significantly large. So only the information of a bounded number of adjacent nodes is sent along. We have set this bound to 6 for our experiments. The second key optimization consists in not sending two messages simultaneously to possible neighbours s and t while trying to inform them about each other. Instead, a message is first sent to s and then it is s that informs t, if required. This also reduces the number of messages since s and t may already be neighbours by the time s receives the notification and, consequently, there is no need to inform t.

3 Discussion and Extensions

Compared to BD07, the CC algorithm computes the complete VD. The CC algorithm also provides the complete Delaunay triangulation and the convex hull without additional communication. After the computation of the VD, a node that has two consecutive neighbours separated by an angle larger than π is on the outer face of the Delaunay triangulation and, therefore, on the convex hull.

The CC algorithm does not rely on any specific routing algorithm. Also, according to experimental results (see Section 4), the CC algorithm uses a smaller number of transmissions. Recall BD07 relies on the GPSR routing protocol. For the sake of fairness, we use GPSR as the underlying routing protocol for the CC algorithm as well. We believe that with a better routing algorithm, the CC algorithm may further reduce the number of transmissions.

From a practical point of view, it is desirable to extend the underlying model beyond UDG. We can extend the scope of Theorem 1, with slight modifications, to more general graphs. Let G' be an arbitrary network obtained from G by removing links, and let $DT(G')$ be the subgraph of G' that contains only the links of G' that are edges of the Delaunay triangulation of S. Theorem 2 shows that as long as $DT(G')$ is connected, the CC algorithm computes the VD correctly.

Theorem 2 *Let G' be a subgraph of G, such that $V(G') = S$ and $DT(G')$ is connected. The CC algorithm computes the correct Voronoi cell of every node in G'.*

Proof. The proof is similar to the proof of Theorem 1. Once the cycle or path P is found, by assuming that the VD was not properly constructed, a contradiction arises with respect to the connectivity of $DT(G')$. In this case no link of $DT(G')$ crosses P while $DT(G')$ should be connected. □

4 Simulation Results

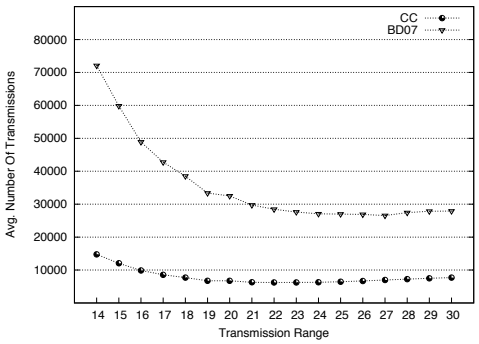
We have conducted intensive simulations on randomly generated networks to study the performance of the CC algorithm and compare the results with the BD07 algorithm [2].

Experiments were done with test sets consisting of 100 nodes randomly placed in a 100×100 unit grid. The density of the graph is controled by different transmission ranges (14 to 30 units). Also two different error rates, 0% and 20% are considered. The error rate is

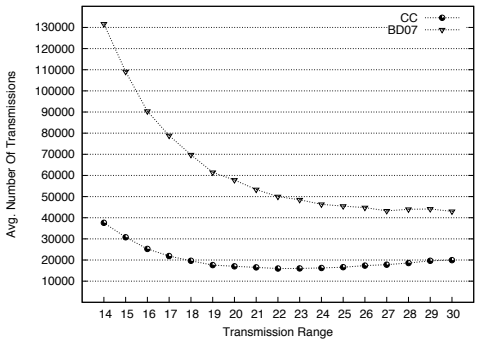
the probability of a transmission to fail. Hence, when a link temporarily fails, the transmission is repeated. For each transmission range and each error rate two algorithms (CC and BD07) are run with 1000 randomly generated networks as defined above. We also incorporate 50 randomly placed opaque obstacles in the form of bars of length 5. Special care is taken such that each graph generated contains a connected subgraph of the Delaunay triangulation as required by Theorem 2.

The entire number of simulations per algorithm is equal to [number of networks] $\times$ [number of transmission ranges] $\times$ [number of error rates] $= 1000 \times 17 \times 2 = 34,000$.

The graphs shown in Figures 3a and 3b provide the total number of transmissions in average for each transmission range with 0% and 20% error rates. For small values of the transmission range, BD07 requires a much larger number of transmissions than the CC algorithm. This is because the network is sparser and the GPSR protocol performs poorly. Because the CC algorithm does not require probing, it is not as affected as BD07 by small transmission ranges.



(a) 0% error rate



(b) 20% error rate

Figure 3: Simulation results.

5 Conclusion

We propose a novel distributed algorithm to compute the VD of a given network. The CC algorithm offers two significant advantages over previous works: (1) it computes the complete VD, and thus, can provide other useful structures such as the Delaunay triangulation and the convex hull; (2) it is more efficient in terms of the number of transmissions as verified through a large number of simulations.

Interesting problems remain open regarding the distributed construction of the VD. A natural question is whether it is possible to find a non-trivial efficient distributed algorithm for constructing the VD of arbitrarily connected networks.

Acknowledgments

We thank Selim Akl and David Rappaport for discussions and comments.

References

[1] W. Alsalih, K. Islam, Y. Núñez-Rodríguez, and H. Xiao. Distributed voronoi diagram computation in wireless sensor networks. In *SPAA '08: 20th ACM Symposium on Parallelism in Algorithms and Architectures*, Munich, Germany, June 2008.

[2] B. A. Bash and P. J. Desnoyers. Exact distributed voronoi cell computation in sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 236–243, New York, NY, USA, 2007. ACM Press.

[3] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[4] J. Byers, J. Considine, and M. Mitzenmacher. Geometric generalizations of the power of two choices. In *16th ACM Symp. on Parallel Algorithms and Architectures*, 2004.

[5] W. Chen, J. Hou, and L. Sha. Dynamic clustering for acoustic target tracking in wireless sensor networks. *IEEE Transaction on Mobile Computing*, 3(3):258–271, 2004.

[6] S. Fortune. A sweepline algorithm for voronoi diagrams. In *SCG '86: 2nd Annual Symposium on Computational Geometry*, pages 313–322. ACM Press, 1986.

[7] L. Girod and D. Estrin. Robust range estimation using acoustic and multimodal sensing. In *IEEE/RSJ Conference on Intelligent Robots and Systems*, 2001.

[8] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *ACM MobiCom*, pages 243–254, 2000.

[9] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *6th Annual International Conference on Mobile Computing and networking*, pages 32–43, New York, NY, USA, 2000.

[10] M. Sharifzadeh and C. Shahabi. Supporting spatial aggregation in sensor network databases. In *12th International Symposium of ACM GIS*, 2004.

[11] Z. Zhou, S. Das, and H. Gupta. Variable radii connected sensor cover in sensor networks. In *IEEE SECON*, 2004.

A Framework for Multi-Core Implementations of Divide and Conquer Algorithms and its Application to the Convex Hull Problem *

Stefan Näher Daniel Schmitt †

Abstract

We present a framework for multi-core implementations of divide and conquer algorithms and show its efficiency and ease of use by applying it to the fundamental geometric problem of computing the convex hull of a point set. We concentrate on the Quickhull algorithm introduced in [2]. In general the framework can easily be used for any D&C-algorithm. It is only required that the algorithm is implemented by a C++ class implementing the job-interface introduced in section 3 of this paper.

1 Introduction

Performance gain in computing is no longer achieved by increasing cpu clock rates but by multiple cpu cores working on shared memory and a common cache. In order to benefit from this development software has to exploit parallelism by multi-threaded programming. In this paper we present a framework for the parallelization of divide and conquer algorithms and show its efficiency and ease of use by applying it to a fundamental geometric problem: computing the convex hull of a point set in two dimensions.

In general our framework supports parallelization of divide and conquer algorithms working on on linear containers of objects (e.g. an array of points). We use the STL iterator interface ([1]), i.e., the input is defined by two iterators $left$ and $right$ pointing to the leftmost and rightmost element of the container. The framework is generic. It can be applied to any D&C-algorithm that is implemented by a C++ class template that implements a certain job interface defined in section 3.

The paper is structured as follows. In Section 2 we discuss some aspects of the parallelization of D&C-algorithms, Section 3 defines the job-interface which has to be used for the algorithms, such that the solvers presented in Section 5 can be applied. Section 6 presents some experimental results, in particular the speedup achieved for different numbers of cpu cores and different problem instances. Finally, Section 7 gives some conclusions and reports on current and ongoing work.

*This work was supported by DFG-Grant Na 303/2-1

†Department of Computer Science, University of Trier, Germany. {naeher, schmittd}@uni-trier.de

2 Divide and Conquer Algorithms

Divide and conquer algorithms solve problems by dividing them into subproblems, solving each subproblem recursively and merging the corresponding results to a complete solution. All subproblems have exactly the same structure as the original problem and can be solved independently from each other, and so can easily be distributed over a number of parallel processes or threads. This is probably the most straightforward parallelization strategy. However, in general it can not be guaranteed that always enough subproblems exist, which leads to non-optimal speedups. This is in particular true for the first divide step and the final merging step but is also a problem in cases where the recursion tree is unbalanced such that the number of open sub-problems is smaller than the number of available threads.

Therefore, it is important that the divide and merge steps are solved in parallel when free threads are available, i.e. whenever the current number of sub-problems is smaller than number of available threads. Our framework basically implements a management system that assigns jobs to threads in such a way that all cpu cores are busy.

3 Jobs

In the proposed framework a *job* represents a (sub-)problem to be solved by a D&C-algorithm. The first (or root) job represents the entire problem instance. Jobs for smaller sub-problems are created in the divide steps. As soon as the size of a job is smaller than a given constant it is called a *leaf* job which is solved directly without further recursion. As soon as all children of a job have been solved the merge step of the D&C-algorithm is applied and computes the result of the entire problem by combining the results of its children.

In this way jobs represent sub-problems as well as the corresponding solutions. Note that the result of a job is either contained in the corresponding interval of the input container or has to be represented in a separate data structure, e.g. a separate list of objects. Quicksort is an example for the first case and Quickhull (as presented in Section 4) for the second case.

The algorithm is implemented by member functions

of the job class which must have the following interface.

```
class job
{ job(iterator left, iterator right);
  bool  is_leaf();
  void  handle_leaf();
  list<job> divide();
  void  merge(list<job>& L);
};
```

In the constructor a job is created by storing two iterators (e.g. pointers into an array) that define the first and last element of the problem. If the `is_leaf` predicate returns true recursion stops and the problem is solved directly by calling the handle_leaf operation. The `divide` operation breaks a job into smaller jobs and returns them in a list, and the `merge` operation combines the solutions of sub-jobs (given as a list of jobs) to a complete solution. There are no further requirements to a job class.

4 Quickhull

We show how to define a job class `qh_job` implementing the well-known Quickhull algorithm ([2]) for computing the convex hull of a point set. For simplicity we consider a version of the algorithm that only computes the upper hull of the given point set and we assume that the input is give by a pair of iterators `left` and `right` into an array of points such that `left` contains the minimal and `right` the maximal point in the lexicographical xy-ordering. The result of a `qh_job` instance is the sequence of points of the upper hull lying between left and right. In this scenario any job of size two (only the leftmost and rightmost point) represents a leaf problem and has the empty list as result. Consequently, the handle_leaf operation is trivial (keeping an empty result list).

The `divide` operation is using two auxiliary functions: `farthest_point(l,r)` computes a point between l and r with maximal distance to the line segment (l, r) and `partition_triangle` implements the partition step of quickhull as shown in Figure 1 and returns the generated sub-problems as a list of jobs. We tried different variants of this partition function. In particular, one using only one thread and one using all available threads. The latter version is similar to the parallel partition strategy proposed in [4] for a multi-core implementation of Quicksort. In the experiments in Section 6) we will see that this can have a dramatic effect on the speedup achieved.

Finally, the `merge` operation takes a list of (two) jobs as input, concatenates their result lists, and inserts the right-most point of the first problem in between. The complete implementation is given by the following piece of C++ code.

```
template<class iterator> class qh_job {
 iterator left;
 iterator right;
 list<point> result;

public:

qh_job(iterator l, iterator r): left(l),right(r) {}
int  size()    { return right - left + 1; }
bool is_leaf() { return size() == 2; }
void  handle_leaf() {}

list<qh_job> divide()
{ iterator pivot = farthest_point(left,right);
  iterator lh,rh;
  partition_triangle(pivot,left,right,lh,rh);
  list<qh_job> L;
  L.push_back(qh_job(left,lh));
  L.push_back(qh_job(rh,right));
  return L;
}

void  merge(list<qh_job>& children)
{ qh_job j1 = children.front();
  qh_job j2 = children.back();
  result.conc(j1.result);
  result.push_back(j1.right);
  result.conc(j2.result;
}
};
```

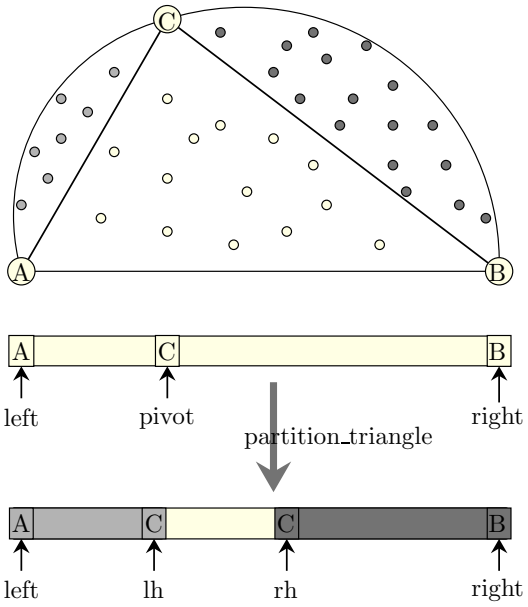


Figure 1: The partition step of Quickhull.

5 Solvers

Our framework provides different *solvers* which can be used to compute the result of a job. As a very basic and simple example we give the code for a generic serial recursive solver. It can simply be implemented by a C++ function template.

```
template <class job>
void solve_recursive(job& j)
{ if (j.is_leaf()) j.handle_leaf();
  else { list<job> Jobs = j.divide();
         job x;
         forall(x,Jobs) solve_recursive(x);
         j.merge(Jobs);
       }
};
```

Note that `solve_recursive` is a generic dc-solver. It accepts any job type *job* that implements the `dc_job` interface. We can now use it easily to implement a serial quickhull function taking an array of points as input.

```
list<point> QH_SERIAL(array<point>& A)
{ int n = A.size();
  qh_job<point*> j(A[0],A[n-1]);
  solve_recursive(j);
  list<point> hull = j.result;
  hull.push_front(A[0]);
  hull.push_back(A[n-1]);
  return hull;
};
```

It is an easy exercise to write a non-recursive version of this serial solver: simply push all jobs created by divide operations on a stack and use an inner loop processing all jobs on the stack.

5.1 Parallel Solvers

A parallel solver is much more complex. It maintains unsolved jobs, builds the recursion tree of jobs while the algorithm proceeds and checks for the mergeability of sub-jobs. It also has to administrate all parallel working threads.

We implement parallel solvers by C++ class templates. The constructor takes as argument the number of threads to be used for solving the problem. There are more parameters that can be changed by corresponding methods of the class. For instance, a limit d for the minimal problem size for any thread. If a the size of job gets smaller than d it will not be divided into new jobs but solved by the same thread using a serial algorithm. Using this limit the overhead of starting a huge number of threads on very small problem instances can be avoided.

```
template <class Job>
class dc_parallel_solver {
public:
  dc_parallel_solver(int thread_num);
  void set_limit(int d);
  void run(Job& j)
};
```

We now can use the parallel solver template to implement a parallel version of the quickhull function.

```
list<point> QH_PARALLEL(array<point>& A, int thr_n)
{ int n = A.size();
  dc_parallel_solver<job<point*> > solver(thr_n);
  job<point*> j(A[0],A[n-1]);
  solver.run(j);
  list<point> hull = j.result;
  hull.push_front(A[0]);
  hull.push_back(A[n-1]);
  return hull;
};
```

6 Experiments

All experiments were executed on a Linux PC with an Intel quad-core processor running at a speed of 2.6 GHz. As implementation platform we used a thread-safe version of LEDA ([3]). In particular, we used the exact geometric primitives of the rational geometry kernel and some of the basic container types such as arrays and lists. All programs were compiled with gcc 4.1.

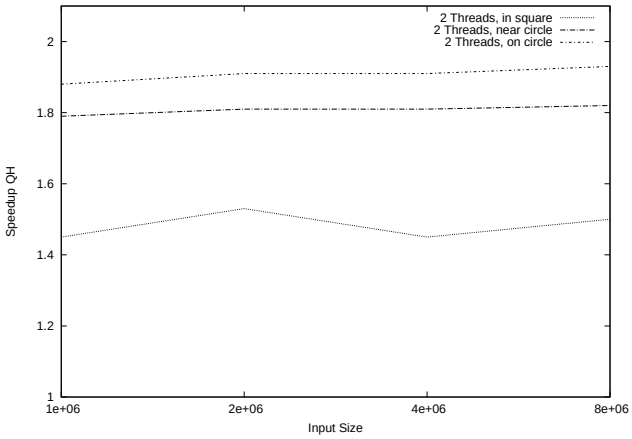


Figure 2: Quickhull: Speedup with 2 cores.

For the experiments we used three different problem generators: random points lying in a square, random points near a circle, and points lying exactly on a circle. Figures 2 and 3 show that we our framework achieved a good speedup behavior for points on or near a circle, which is the difficult case for Quickhull because only a few or none of the points can be eliminated in the partitioning step. Note that the 1.0 baseline indicates the

performance of a serial version of the algorithm (using only one thread). It turned out that $n/100$ was good choice for the limit mentioned in section 5.1.

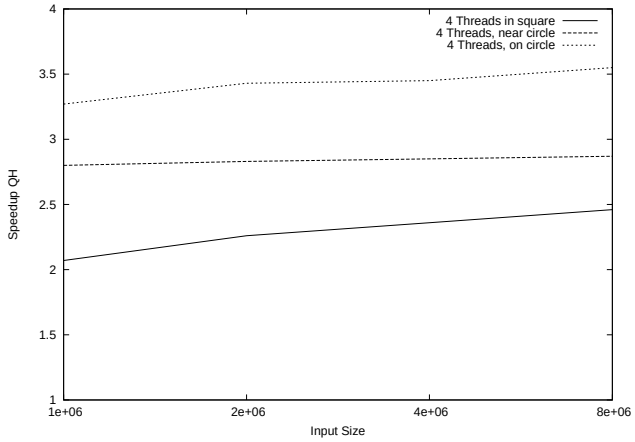


Figure 3: Quickhull: Speedup with 4 cores.

For random points in a square Quickhull eliminates almost all of the input points in the root job of the algorithms (with high probability), i.e. almost the entire work is done here. In this case the achieved speedup is not optimal. However, Figure 4 shows that without parallelization of the partitioning step we have no speedup at all. We have some ideas to improve the parallel partitioning and hope to improve the results for this kind of problem instances.

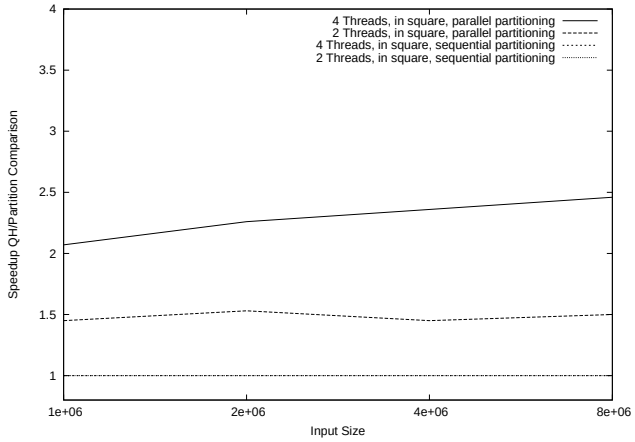


Figure 4: Quickhull: The effect of parallel partitioning.

We also want to mention here that we ran experiments with different D&C algorithms for convex hulls. In particular, a recursive version of the gift wrapping method where the merge step does most of the work by constructing two tangents. Figure 5 shows the speedup behavior of this algorithm for the same set of input instances.

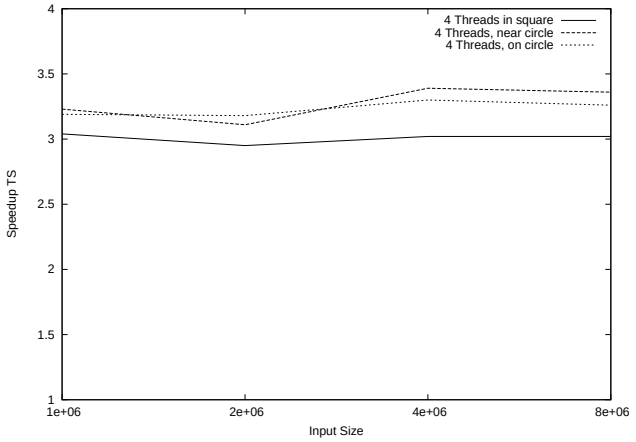


Figure 5: Tangent Search: Speedup with 4 cores.

7 Conclusions

We have presented a framework for the implementation and parallelization of divide and conquer algorithms. The framework is generic (by using C++ templates) and can be used very easily. The experiments show that a considerable speedup can be achieved by using two or four threads on a quad core machine. We have some ideas to improve the parallel partitioning of the quickhull algorithm and hope to be able to improve the efficiency in cases where most of the work is done in the root job. In this short version of the paper we could not present all experiemental results. In particular, our framework shows a very good performance also on basic D&C algorithms such as Quicksort (see the online version of the paper for more details). We also work on the parallelization of incremental algorithms for geometric problems and higher dimensional problems. One of the major problems is the need of more complicated thread-safe dynamic data structures such as graphs or polyhedra.

References

[1] M. H. Austern *Generic programming and the STL*, Addison-Wesley, 2001.

[2] A. Bykat *Convex hull of a finite set of points in two dimensions.* IPL, 7:296-298, 1978.

[3] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[4] Philippas Tsigas and Yi Zhang *A Simple, Fast Parallel Implementation of Quicksort and its Performance Evaluation on SUN Enterprise 10000.*

Guaranteed Voronoi Diagrams of Uncertain Sites

William Evans* and Jeff Sember†

Abstract

In this paper we investigate the Voronoi diagram that is induced by a set of sites in the plane, where each site's precise location is uncertain but is known to be within a particular region, and the cells of this diagram contain those points guaranteed to be closest to a particular site. We then examine the diagram for sites with disc-shaped regions of uncertainty, prove that it has linear complexity, and provide an optimal $O(n \log n)$ algorithm for its construction. We also examine the diagram for polygonal regions of uncertainty, and prove that it has linear complexity as well. We then describe a generalization of these diagrams, in which each Voronoi cell is associated with a subset of the sites, and each point in a cell is guaranteed to be closest to some site in the subset associated with the cell.

1 Introduction

Suppose we do not know the precise locations of n sites (n points in the plane) and yet we would like to determine, for every point in the plane, the closest site to that point. If we know the approximate location of each site, say, that the ith site lies in a subset D_i of the plane, then we might be able to answer this question perhaps not for every point but for many points in the plane. Our goal is to find, for each site i, the set of points that are guaranteed to be closer to that site than to any other. In other words, no matter where each site lies (as long as the jth site is in D_j for every j) the closest site to the point is always site i. For some points, we cannot guarantee a closest site. These points form a subset of the plane that we call the 'neutral zone'.

In this paper, we first formally define the partition of the plane into cells of guaranteed closest points and the neutral zone and state some properties of this partition. We then consider the special case when the uncertain regions (i.e. the subsets D_i) are discs and show that the complexity of the partition in this case is linear in the number, n, of sites, and that it can be calculated in $O(n \log n)$ time.

We also consider the case where each D_i is a polygon, and show that the complexity of the resulting partition is linear in the total number of polygon edges.

We then consider a finer partition of the neutral zone into regions of points that we can guarantee are closest to some site in a set of sites. For example, points that may be closest to sites 1 or 2 form the region for the set $\{1, 2\}$. We show that the complexity of this finer partition is at most $O(n^4)$ for uncertain discs.

An applet demonstrating these diagrams is available at `http://www.cs.ubc.ca/~jpsember/gv.html`.

2 Related work

Voronoi diagrams are a fundamental data structure in computational geometry; see [2] for a survey. Voronoi diagrams involving uncertain sites were investigated with respect to the probabilistic concepts of *expected closest site* and *probably closest site* in [3].

The guaranteed Voronoi diagram of a set of uncertain regions is closely related to the standard Voronoi diagram of those regions. Thus our results rely heavily on properties of standard Voronoi diagrams such as diagrams for circles [8] and diagrams for segments [6].

One of the biggest differences between the guaranteed Voronoi diagram and traditional variants of Voronoi diagrams is that the union of the regions associated with uncertain sites does not cover the plane. The guaranteed Voronoi diagram contains a neutral region that contains those points that are not guaranteed to be closest to any particular site. Zone diagrams also have this property. In zone diagrams, for a point to be in a site's region, it must be closer to the site than to any point in any other site's region. The recursive nature of this definition raises the question of the uniqueness and existence of zone diagrams; a question that Asano et al. [1] answered (positively).

Some properties of guaranteed Voronoi diagrams of uncertain polygons are given in [5], including a proof of the diagrams' computability, though no complexity claims are made.

3 Properties

We are given a set of regions in the plane $\mathcal{D} = \{D_1, \ldots, D_n\}$, called *uncertain regions*, each containing a site. Let $H(i, j)$ be the set of points in the plane that are guaranteed to be at least as close to site i as site j.

*UBC Computer Science,Vancouver, B.C., Canada, V6T 1Z4; will@cs.ubc.ca

†UBC Computer Science,Vancouver, B.C., Canada, V6T 1Z4; jpsember@cs.ubc.ca; research is supported by NSERC

That is,

$$H(i,j) = \{p \mid \forall x \in D_i \; \forall y \in D_j \; d(p,x) \leq d(p,y)\}$$

where $d(\cdot,\cdot)$ is Euclidean distance. We denote the boundary of $H(i,j)$ by $\langle i,j\rangle$; formally,

$$\langle i,j\rangle = \{p \mid \max_{x\in D_i} d(p,x) = \min_{y\in D_j} d(p,y)\} \; .$$

The *cell* for site i, denoted $R_{\langle i\rangle}$, is

$$R_{\langle i\rangle} = \bigcap_{j\neq i} H(i,j) \; . \tag{1}$$

The boundaries of all such cells $R_{\langle i\rangle}$ form the *guaranteed Voronoi diagram* for the set $\mathcal{D}$, and we denote it by $V(\mathcal{D})$.

An *edge of* $\langle i,j\rangle$ in $V(\mathcal{D})$ is a maximal connected set of points $p \in \langle i,j\rangle$ that lie on the boundary of cell $R_{\langle i\rangle}$.

Some properties of $V(\mathcal{D})$ are easy to show.

If every uncertain region is a single point, $V(\mathcal{D})$ is the standard nearest-point Voronoi diagram for the regions, which we denote by $V^\circ(\mathcal{D})$. In general, for arbitrary uncertain regions, every cell $R_{\langle i\rangle}$ of $V(\mathcal{D})$ is a subset of the corresponding cell $R_{\langle\langle i\rangle\rangle}$ of $V^\circ(\mathcal{D})$.

It is possible for a cell boundary to not be a one-dimensional curve. Consider $D_i = \{(x,0) \mid x \in [0,2]\}$ and $D_j = \{(x,0) \mid x \in [2,4]\}$. In this case, $\langle i,j\rangle$ is the halfplane $\{(x,y) \mid x \leq 1\}$, and $R_{\langle i\rangle} = \langle i,j\rangle$. To generalize, if D_j intersects CH_i, the convex hull of D_i, then $H(i,j) = \langle i,j\rangle$; and if this intersection is not confined to vertices of CH_i, then $H(i,j) = \langle i,j\rangle = \emptyset$. From this point on, we assume that any nonempty intersection of two regions D_i and D_j is not confined to vertices of CH_i.

A site whose cell is empty can still influence the cell of another site. For example, if the interiors of D_i and D_j intersect, then $R_{\langle i\rangle} = \emptyset$, yet an edge of $\langle k,i\rangle$ for some other site k can still appear in $V(\mathcal{D})$.

A connected subset of the plane S is *inside-tangent* to another such subset C (or C has inside-tangent S) if $S \subseteq C$ and the boundary of C intersects S; and S is *outside-tangent* to C (or C has outside-tangent S) if $S \cap C$ is a non-empty subset of the boundary of C.

Lemma 1 *Every point p on an edge of $\langle i,j\rangle$ in $V(\mathcal{D})$ is the center of a unique disc C_p that has inside-tangent D_i, outside-tangent D_j, and intersects the interior of no $D_k \in \mathcal{D}$ for $k \notin \{i,j\}$.*

Proof. This follows immediately from the definition of an edge of $\langle i,j\rangle$. □

Consider a point p on an edge of $\langle i,j\rangle$ in $V(\mathcal{D})$, and its disc C_p from Lemma 1. Let b be a point of tangency of C_p with D_j. Define $\delta(p)$ to be the (unique) point on segment $\overline{pb}$ that is the center of a disc C° that has outside tangent D_j (at the point b) and outside tangent D_i. Since $C^\circ \subseteq C_p$, C° also intersects the interior of no $D_k \in \mathcal{D}$ for $k \notin \{i,j\}$. Thus $\delta(p)$ lies on an edge of the bisector $\langle\langle i,j\rangle\rangle$ between D_i and D_j that is part of the standard Voronoi diagram $V^\circ(\mathcal{D})$ for the regions $\mathcal{D}$. In fact, $\delta(p)$ is on the boundary of region $R_{\langle\langle i\rangle\rangle}$.

Note that if more than one region D_j is outside-tangent to C_p (or if D_j is tangent to C_p at more than one point), then there is more than one candidate point of tangency b. To make $\delta(p)$ well-defined, we select a b according to some total order on possible b's.

We now show that the ordering of points p on the boundary of a cell $R_{\langle i\rangle}$ in $V(\mathcal{D})$ agrees with that of points $\delta(p)$ on the boundary of $R_{\langle\langle i\rangle\rangle}$ in $V^\circ(\mathcal{D})$. To do this, we will need the following lemma.

Lemma 2 *If b is a point on the boundary of disc P centered at p, and d a point on the boundary of disc Q centered at q, and line segments $\overline{pb}$ and $\overline{qd}$ intersect at a single point, interior to both, then either b is in the interior of Q or d is in the interior of P.*

Proof. Assume such an intersection point w exists. Without loss of generality, assume $d(w,b) \leq d(w,d)$. By the triangle inequality,

$$\begin{aligned} d(q,b) &< d(q,w) + d(w,b) \\ &\leq d(q,w) + d(w,d) \end{aligned}$$

which implies b is in the interior of Q. □

We denote a point p being encountered before point q as we traverse the boundary of a convex region counterclockwise (ccw) from starting point s by $p \prec_s q$.

Lemma 3 *If p, q, and s are points on the boundary of cell $R_{\langle i\rangle}$ (with nonempty interior), and $p \prec_s q$, then $\delta(p) \prec_{\delta(s)} \delta(q)$.*

Proof. Each point p on the boundary of cell $R_{\langle i\rangle}$ is mapped to a point $\delta(p)$ on the boundary of cell $R_{\langle\langle i\rangle\rangle}$. Note that segment $\overline{p\delta(p)}$ does not intersect the interior of $R_{\langle i\rangle}$, since for every point p' on this segment, the disc with center p' that has outside-tangent D_j does not contain all of D_i except when $p' = p$. Note also that this disc does intersect D_i (and no other D_k), thus $\overline{p\delta(p)}$ is within $R_{\langle\langle i\rangle\rangle}$. Therefore if $p \prec_s q$ and $\delta(p) \succeq_{\delta(s)} \delta(q)$ then some two of the segments $\{\overline{s\delta(s)}, \overline{p\delta(p)}, \overline{q\delta(q)}\}$ intersect. Without loss of generality, assume $\overline{p\delta(p)}$ intersects $\overline{q\delta(q)}$.

By Lemma 1, disc C_p exists which has outside-tangent some $D_{j\neq i}$ at b, such that $\delta(p) \in \overline{pb}$. Similarly, disc C_q exists which has outside-tangent some $D_{k\neq i}$ at d where $\delta(q) \in \overline{qd}$. This implies $\overline{pb}$ intersects $\overline{qd}$ (since $\overline{p\delta(p)} \subset \overline{pb}$ and $\overline{q\delta(q)} \subset \overline{qd}$). The intersection is a single interior point since $p \notin \overline{qd}$ and $q \notin \overline{pb}$ (otherwise C_p or C_q would not contain D_i), and $\delta(p) \neq b$ and $\delta(q) \neq d$ (otherwise D_i intersects D_j or D_k, and $R_{\langle i\rangle}$ has an empty interior). By Lemma 2, either b is in the interior of C_q or d is in the interior of C_p, which is a contradiction. □

4 Uncertain discs

We now consider the case where the uncertain regions are discs; see Figure 1.

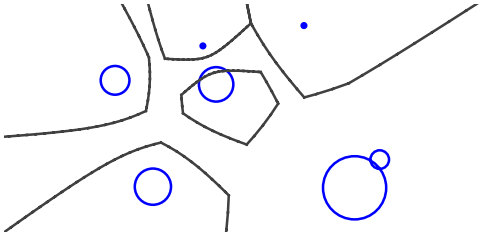


Figure 1: Guaranteed Voronoi diagram

Each disc has a nonnegative radius r_i, and a center S_i. Each $p \in \langle i,j \rangle$ satisfies

$$d(p,S_i) + r_i = d(p,S_j) - r_j \ . \tag{2}$$

Since r_i and r_j are constants, the points p which satisfy (2) lie on an arm of a hyperbola with foci at S_i and S_j. If the discs' radii are both zero, this is the perpendicular bisector of $\overline{S_iS_j}$; otherwise, it is the hyperbolic arm closest to S_i.

Some properties of $V(\mathcal{D})$ include the following.

Each cell of $V(\mathcal{D})$ is convex, since the cells are intersections of convex halfplanes bounded by hyperbolic arms.

It is possible that more than one edge of $\langle i,j \rangle$ appears in $V(\mathcal{D})$.

We will now show that the number of edges in a guaranteed Voronoi diagram of n discs is $O(n)$. We will do this by showing that for each cell $R_{\langle i \rangle} \in V(\mathcal{D})$, there is a mapping from each edge in $R_{\langle i \rangle}$ to a distinct edge in the corresponding cell of $V^\circ(\mathcal{D})$, which is known to have $O(n)$ edges.

Theorem 4 *The number of edges in a guaranteed Voronoi diagram of n uncertain discs is $O(n)$.*

Proof. We will show that the number of edges in $V(\mathcal{D})$ is at most twice the number of edges in $V^\circ(\mathcal{D})$. The theorem then follows from the fact that $V^\circ(\mathcal{D})$ has $O(n)$ edges (property (7) of [8]).

Consider the edges around $R_{\langle i \rangle}$ in ccw order. We charge each edge E of $\langle i,j \rangle \in V(\mathcal{D})$ to the edge F of $\langle\langle i,j \rangle\rangle$ on which $\delta(p)$ lies, for p the ccw-first point of E (or any interior point p if E is ccw-infinite). Suppose two distinct edges E_1 and E_2 of $\langle i,j \rangle$ map to the same edge F of $\langle\langle i,j \rangle\rangle$ in $R_{\langle\langle i \rangle\rangle}$. Since E_1 and E_2 are distinct but both of $\langle i,j \rangle$, there must exist an edge E' of $\langle i,k \rangle$ $(k \neq j)$ between them in the ccw traversal of $R_{\langle i \rangle}$ that maps to some other edge F' of $\langle\langle i,k \rangle\rangle$ in $R_{\langle\langle i \rangle\rangle}$. This contradicts Lemma 3 since all points of F either precede or follow the points of F' in ccw-order.

Thus each edge in $V^\circ(\mathcal{D})$ of $\langle\langle i,j \rangle\rangle$ is charged at most twice: once by an edge of $\langle i,j \rangle$, and once by an edge of $\langle j,i \rangle$. Hence $V(\mathcal{D})$, like $V^\circ(\mathcal{D})$, has $O(n)$ edges. □

We now show how $V(\mathcal{D})$ for a set of discs can be constructed by first constructing $V^\circ(\mathcal{D})$ for the discs, then performing a linear-time transformation from $V^\circ(\mathcal{D})$ to $V(\mathcal{D})$.

We can construct $I_{\langle i \rangle}$, a sequence of neighboring sites to cell $R_{\langle i \rangle}$, by starting from an edge containing some point p on the boundary of $R_{\langle i \rangle}$ and traversing the boundary edges in ccw order. We construct $I_{\langle\langle i \rangle\rangle}$, the sequence of neighboring sites to cell $R_{\langle\langle i \rangle\rangle}$, by a similar ccw traversal, starting from the edge containing $\delta(p)$.

Lemma 5 *For every cell $R_{\langle i \rangle} \in V(\mathcal{D})$, $I_{\langle i \rangle}$ is a subsequence of $I_{\langle\langle i \rangle\rangle}$.*

Proof. If site j is in $I_{\langle i \rangle}$ then, since $\delta(\cdot)$ maps points on edges of $\langle i,j \rangle$ to points on edges of $\langle\langle i,j \rangle\rangle$, j is in $I_{\langle\langle i \rangle\rangle}$. Furthermore, the order of sites in $I_{\langle i \rangle}$ is preserved in $I_{\langle\langle i \rangle\rangle}$ since $\delta(\cdot)$ preserves this order by Lemma 3. □

Theorem 6 *$V(\mathcal{D})$ for n sites can be constructed in $O(n \log n)$ time, and this running time is optimal.*

Proof. The running time of any algorithm to construct $V(\mathcal{D})$ is $\Omega(n \log n)$, since if the site radii are all zero, $V(\mathcal{D})$ is the standard Voronoi diagram of n points.

Constructing $V^\circ(\mathcal{D})$ for the disc sites $\mathcal{D}$ takes $O(n \log n)$ time [4]. We generate the sequence $I_{\langle\langle i \rangle\rangle}$ of sites comprising the boundary of cell $R_{\langle\langle i \rangle\rangle}$ in $V^\circ(\mathcal{D})$ for $i = 1, 2, \ldots, n$ from this diagram in linear time by a simple traversal. From $I_{\langle\langle i \rangle\rangle}$ we construct the boundary of $R_{\langle i \rangle}$ by generating and intersecting the sequence of hyperbolic arcs it specifies. Lemma 5 ensures that we consider a correctly ordered super-sequence of the arcs bounding $R_{\langle i \rangle}$. This suffices to construct the boundary of $R_{\langle i \rangle}$ in time proportional to the length of $I_{\langle\langle i \rangle\rangle}$.

Since each of the $O(n)$ edges of $V^\circ(\mathcal{D})$ appears in two cell boundaries, the running time for the construction of the edges of all cells of $V(\mathcal{D})$ is $O(n)$. The time to construct $V(\mathcal{D})$ is thus dominated by the time to construct $V^\circ(\mathcal{D})$. □

5 Uncertain polygons

We now turn our attention to the case where the region of uncertainty for each site is a polygon. In this case, each $\langle i,j \rangle$ consists of some number of (possibly unbounded) parabolic arcs, each induced by a vertex u of D_i and a vertex[1] or open edge v of D_j. We denote such a parabolic arc by $\langle i^u, j^v \rangle$, and define an edge of $\langle i^u, j^v \rangle$ to be a maximal connected set of points $p \in \langle i^u, j^v \rangle$ that lie on the boundary of cell $R_{\langle i \rangle}$. We define $\langle\langle i^u, j^v \rangle\rangle$ for

[1] In this case, the induced parabola degenerates to a line.

$V^{\triangle}(\mathcal{D})$, the standard Voronoi diagram of the polygons $\mathcal{D}$, analogously.

Theorem 7 *The number of edges in the guaranteed Voronoi diagram of $\mathcal{D}$, a set of n polygons with m total edges, is $O(m)$.*

Proof. We show that the number of edges in $V(\mathcal{D})$ is at most twice the number of edges in $V^{\triangle}(\mathcal{D})$ plus twice the complexity of the furthest point Voronoi diagram of the vertices in D_i summed over all i. The theorem then follows from the fact that $V^{\triangle}(\mathcal{D})$ has $O(m)$ complexity [6] and that the total complexity of the furthest point Voronoi diagrams is $O(m)$ [7].

Let E be an edge of $\langle i^u, j^v\rangle$ on the boundary of $R_{\langle i\rangle}$ and let p be an interior point of E. Section 3 showed there must exist a point $\delta(p)$ on an edge of $\langle\langle i^w, j^v\rangle\rangle$ where w is a vertex or edge of D_i.

Consider the edges around $R_{\langle i\rangle}$ in ccw order. We charge each edge E of $\langle i^u, j^v\rangle$ to the edge F of $\langle\langle i^w, j^v\rangle\rangle$ on which $\delta(p)$ lies, for p the ccw-first point of E (or any interior point p if E is ccw-infinite). Now it may happen that a consecutive sequence of edges around $R_{\langle i\rangle}$ all map to F. (By Lemma 3, the edges must be consecutive if they map to the same F.) Let E_1 of $\langle i^{u_1}, j^v\rangle$ and E_2 of $\langle i^{u_2}, j^v\rangle$ be two successive (adjacent) edges in this ccw sequence. The point p shared by E_1 and E_2 lies on an edge of the furthest-point Voronoi diagram of the vertices of D_i that separates the furthest-point regions for u_1 and u_2. We charge the edge E_2 to this edge T of the furthest-point Voronoi diagram. We now show that at most two edges are charged to each T. Every such p intersecting T is the center of a disc C_p that has inside-tangent D_i (at the two farthest vertices u_1, u_2 associated with T) and outside-tangent D_j. Assume by way of contradiction that there are three such points, p_1, p_2, p_3 in order along T. Observe that C_{p_2} is contained within $C_{p_1} \cup C_{p_3}$; thus D_j must be outside-tangent to C_{p_2} at either u_1 or u_2 to avoid intersecting the interior of the other two discs. But then $D_i \cap D_j$ is a nonempty subset of $\{u_1, u_2\}$, both vertices of CH_i, a contradiction.

Thus the number of edges on the boundary of region $R_{\langle i\rangle}$ is at most the number of edges on the boundary of region $R_{\langle\langle i\rangle\rangle}$ plus twice the number of edges in the furthest-point Voronoi diagram for the vertices of D_i. The theorem then follows since each edge of $V^{\triangle}(\mathcal{D})$ bounds two regions $R_{\langle\langle i\rangle\rangle}$ and $R_{\langle\langle j\rangle\rangle}$. □

6 Extension to subsets of closest points

In this section, we look at an extension of the Voronoi diagram which assigns every point in the plane to a cell, including points in the neutral zone.

Equation (1) can be generalized so that each point in a cell is guaranteed to be at least as close to a site in a particular subset of sites as to any other site. For a set $S \subseteq \{1 \ldots n\}$, we define the cell for S (denoted $R_{\langle S\rangle}$) as

$$R_{\langle S\rangle} = \bigcup_{i\in S}\Big[\bigcap_{j\notin S} H(i,j)\Big] - \bigcup_{S'\subset S} R_{\langle S'\rangle}$$

where $R_{\langle\emptyset\rangle} = \emptyset$. See Figure 2 for an example of such a *guaranteed subset Voronoi diagram*, which we denote by $V^{\{\}}(\mathcal{D})$.

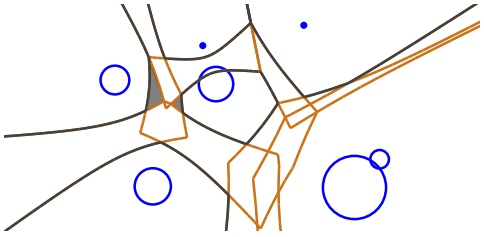


Figure 2: Guaranteed subset Voronoi diagram

The cells of $V^{\{\}}(\mathcal{D})$ are not necessarily connected. In Figure 2, for instance, the two shaded regions belong to the same cell.

Lemma 8 *The number of edges in a guaranteed subset Voronoi diagram of n uncertain discs is $O(n^4)$.*

Proof. The proof follows immediately from the fact that each edge in $V^{\{\}}(\mathcal{D})$ is an edge in the arrangement of the $2 \cdot \binom{n}{2}$ possible hyperbolic arcs $\langle i, j\rangle$. □

References

[1] T. Asano, J. Matoušek, and T. Tokuyama. Zone diagrams: Existence, uniqueness, and algorithmic challenge. *SIAM J. Comput.*, 37(4):1182–1198, 2007.

[2] F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[3] F. Aurenhammer, G. Stöckl, and E. Welzl. The post office problem for fuzzy point sets. In *CG '91: Proc. International Workshop on Computational Geometry - Methods, Algorithms and Applications*, pages 1–11, London, UK, 1991. Springer-Verlag.

[4] S. Fortune. A sweepline algorithm for Voronoi diagrams. In *SCG '86: Proc. 2nd annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM.

[5] A. A. Khanban. *Basic algorithms of computational geometry with imprecise input.* PhD thesis, Imperial College of London, 2005.

[6] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 20th annual symposium on Foundations of Computer Science*, pages 18–27, 1979.

[7] D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, 31(6):478–487, 1982.

[8] M. Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comput.*, 14(2):448–468, 1985.

The Solution Path of the Slab Support Vector Machine

Michael Eigensatz* Joachim Giesen† Madhusudan Manjunath‡

Abstract

Given a set of points in a Hilbert space that can be separated from the origin. The slab support vector machine (slab SVM) is an optimization problem that aims at finding a slab (two parallel hyperplanes whose distance—the slab width—is essentially fixed) that encloses the points and is maximally separated from the origin. Extreme cases of the slab SVM include the smallest enclosing ball problem and an interpolation problem that was used (as the slab SVM itself) in surface reconstruction with radial basis functions. Here we show that the path of solutions of the slab SVM, i.e., the solution parametrized by the slab width is piecewise linear.

1 Introduction

Data structures used in fields like graphics, visualization and learning often have many free parameters. In most cases a good choice of these parameters is not obvious. Computational geometry was facing similar problems: for example when using alpha shapes [Ede95] for surface reconstruction or in bio-geometric modeling the question arises as to what value to choose for alpha. Computational geometry [Ede95, ELZ02, GCPZ06] gave an answer to this question that can be seminal also for the aforementioned areas of computer science, namely, do not compute the solution for a fixed more or less well chosen value of the parameter, but compute the whole spectrum of structures and then look for good solutions in this spectrum. One method to determine a good structure is topological persistence pioneered by Edelsbrunner, Harer and Zomorodian [ELZ02].

Here we investigate an optimization problem that has its roots in machine learning and was also applied in various forms to the surface reconstruction problem. The problem is called slab support vector machine (slab SVM) [SGS04] and takes as input a set of data points in a Hilbert space that can be separated from the origin and aims at finding a slab (two parallel hyperplanes whose width is essentially fixed as $\delta > 0$) that encloses the points and is maximally separated from the origin. The slab SVM has found applications in surface reconstruction [SGS04], and quantile estimation and novelty detection [SS02]. In these applications the data points reside in d-dimensional Euclidean space but are mapped by a feature map into another (often infinite dimensional) Hilbert space. The structure of the slab SVM is such that the feature map does not have to be given explicitly, but only implicitly through a positive kernel: the dual optimization problem of the slab SVM depends only on the pairwise inner products of the data points. A positive kernel can be used to replace these inner products without changing the nature (convex quadratic program) of the optimization problem.

The parameter we are interested in is δ, which essentially fixes the width of the slab. In the applications, it is difficult to tell beforehand what a good choice of δ is. Hence in the spirit of the computational geometry approach we want to compute the solution to the slab SVM for all values of δ. Once we have this spectrum of solutions other methods can be employed to find good choices for δ. Here we do not want to discuss how such methods could look like, but focus on the structure of the solution spectrum. We show that the solution path of the slab SVM, i.e., the solution parametrized by δ is piecewise linear. Our arguments provide a complete geometric characterization of the turning points (nodes) of the solution path.

Our results are in spirit similar to results of Hastie et al. [HRTZ04] who obtained the piecewise linearity of the solution of the classification support vector machine [SS02]. Though both results give piecewise linear solution paths, the parameters are different in nature and so are the means to establish the results. Our proof is of geometric nature, whereas Hastie et al. use algebraic arguments.

2 The slab SVM

Given *data points* $X = \{x_1, \ldots, x_n\} \subset \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space with inner product $\langle \cdot, \cdot \rangle$, such that the data points can be separated from the origin by a hyperplane, i.e., there exists $w \in \mathcal{H} \setminus \{0\}$ and $\rho \neq 0$ such that

$$\langle w, x_i \rangle \geq \rho \text{ for all } i = 1, \ldots, n.$$

The distance of the hyperplane $\{x \in \mathcal{H} : \langle w, x \rangle = \rho\}$ to the origin of $\mathcal{H}$ is given as $\rho / \|w\|$, where the norm of

*Applied Geometry Group, ETH Zürich, eigensatz@inf.ethz.ch

†Institut für Informatik, Friedrich-Schiller-Universität Jena, giesen@minet.uni-jena.de

‡Max-Planck Institut für Informatik, manjun@mpi-inf.mpg.de

w in $\mathcal{H}$ is defined as usual by $\|w\| = \sqrt{\langle w, w\rangle}$.

The *slab SVM* is the following convex quadratic optimization problem that aims at finding the slab (the space between two parallel hyperplanes) with width $\delta/\|w\|$ that contains all the data points and minimizes $\frac{1}{2}\|w\|^2 - \rho$, i.e., essentially maximizes the distance of the slab to the origin (see also Figure 1):

$$\begin{array}{ll} \min_{w,\rho} & \frac{1}{2}\|w\|^2 - \rho \\ \text{s.t.} & \rho \le \langle w, x_i\rangle \le \rho + \delta \text{ for all } i = 1, \ldots, n \end{array}$$

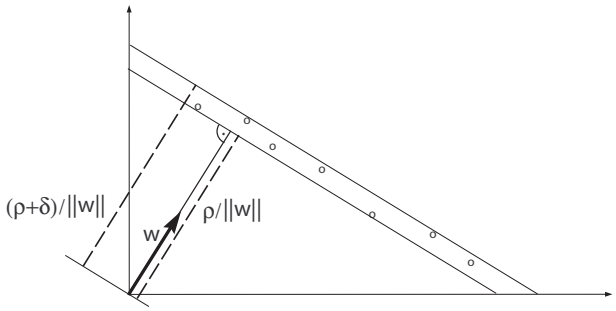


Figure 1: The geometric set-up for the slab SVM.

Note that the slab SVM problem is always feasible since $(w, \rho) = (0, 0)$ is always contained in the constraint polytope.

The Lagrangian dual to this problem can be derived from the saddle point condition for the Lagrangian

$$\begin{aligned} L(w, \rho, \alpha, \beta) &= \frac{1}{2}\|w\|^2 - \rho - \sum_{i=1}^{n} \alpha_i(\langle w, x_i\rangle - \rho) \\ &\quad + \sum_{i=1}^{n} \beta_i(\langle w, x_i\rangle - \rho - \delta), \end{aligned}$$

where $\alpha_i, \beta_i \ge 0$. The saddle point condition gives $\partial L/\partial w = 0$ which implies $w = \sum_{i=1}^{n}(\alpha_i - \beta_i)x_i$ and $\partial L/\partial \rho = 0$ which implies $\sum_{i=1}^{n}(\alpha_i - \beta_i) = 1$ from which the dual follows

$$\begin{array}{ll} \min_{\alpha,\beta} & \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \beta_i)(\alpha_j - \beta_j)\langle x_i, x_j\rangle + \delta\sum_{i=1}^{n}\beta_i \\ \text{s.t.} & \alpha_i, \beta_i \ge 0 \text{ for all } i = 1, \ldots, n. \\ & \sum_{i=1}^{n}(\alpha_i - \beta_i) = 1 \end{array}$$

In most applications [SS02] the data points are obtained from applying a feature map ϕ to input data points $y_1, \ldots, y_n \in \mathbb{R}^d$, i.e., $x_i = \phi(y_i) \in \mathcal{H}$, where the feature map is not given explicitly, but implicitly in form of a positive kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, i.e.,

$$\langle x_i, x_j\rangle = \langle \phi(y_i), \phi(y_j)\rangle = k(x_i, x_j).$$

and $\mathcal{H}$ is the kernel reproducing Hilbert space. Since the dual of the slab SVM only depends on the inner products of the data points, we can replace $\langle x_i, x_j\rangle$ by $k(x_i, x_j)$. A popular positive kernel is the Gaussian

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right),$$

which is an example of a so called radial basis function kernel, i.e., a kernel that only depends on the distance $\|x_i - x_j\|$. The data points $x_i = \phi(y_i)$ are linearly independent and the Gram matrix $\big(k(x_i, x_j)\big)$ associated with the Gaussian kernel is positive, i.e., it has full rank and thus is invertible. In the following we always assume that the data points x_i are linear independent.

3 Surface reconstruction

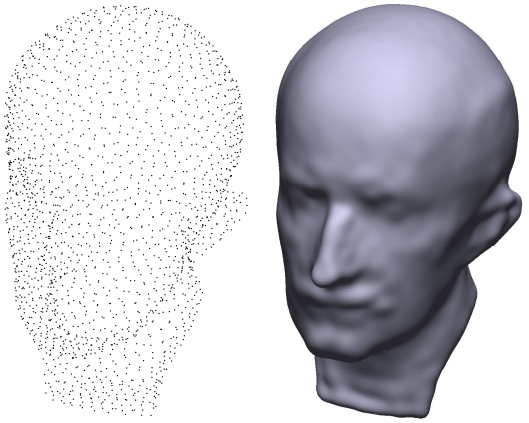


Figure 2: An example surface reconstruction (Max-Planck Head: 2022 points) using the slab SVM for a fixed (small) value of δ.

Let us briefly recapitulate how the slab SVM can be used directly for surface reconstruction [SGS04]. Given are sample points $y_1, \ldots, y_n \in \mathbb{R}^3$ from a smooth surface embedded into $\mathbb{R}^3$. These sample points are mapped into the feature space associated with the Gaussian kernel. The reconstruction is given implicitly as $f^{-1}(0)$, where $f : \mathbb{R}^3 \to \mathbb{R}$ is the kernel expansion

$$f(x) = \langle w, \phi(x)\rangle - \rho = \sum_{i=1}^{n}(\alpha_i - \beta_i)\exp\left(-\frac{\|x_i - x\|^2}{2\sigma^2}\right) - \rho,$$

where $x \in \mathbb{R}^3$, $\phi(\cdot)$ is the feature map associated with the Gaussian kernel, and α and β are the solutions to the dual SVM. Note that ρ can also be computed from the solution to the slab SVM (or its dual). See Figure 2 for an example and also note that especially in the

presence of noise one probably does not want to have an interpolating solution (as one gets it from the slab SVM for $\delta = 0$ and a related method proposed in [CBC+01]), but would like to allow small slack in terms of a small value of $\delta > 0$. Note that the slab SVM works the same for surface reconstruction in dimensions beyond three.

4 States and events

For a given value of $\delta \in (0, \infty)$ let (w, ρ) be the optimal solution of the slab SVM. We associate *states* with the data points $x_i, i = 1, \ldots, n$:

(1) lower supporting, if $\langle w, x_i \rangle = \rho$

(2) upper supporting, if $\langle w, x_i \rangle = \rho + \delta$

(3) non-supporting, if neither lower- nor upper supporting

An *event* occurs when while decreasing δ the state of any data point changes. We distinguish two types of events: a supporting data point becomes non-supporting, or a non-supporting data point becomes supporting. We call the first type of event a *lose event* and the second type of event a *gain event*.

5 The Solution Path

From the constraints $\sum_{i=1}^{n}(\alpha_i - \beta_i) = 1$ and $\alpha_i, \beta_i \geq 0$ of the dual of the slab SVM we can conclude that there exists $\alpha_i > 0$. This in turn allows us to conclude using the Karuhn-Kuhn-Tucker condition $\alpha_i\big(\langle w, x_i \rangle - \rho\big) = 0$ that for any δ there always exists a lower supporting data point. For a given δ', let x_i be a lower supporting data point. The continuous dependence of the coefficient α_i on the parameter δ implies that $\alpha_i > 0$ for some neighborhood of $U(\delta') \subset (0, \infty)$. Hence x_i is a lower supporting data point for all $\delta \in U(\delta')$. We use this insight to locally, i.e., for $\delta \in U(\delta')$, transform the slab SVM into an equivalent distance problem. Note that we have $\rho = \langle w, x_i \rangle$. Thus we can write the objective function of the slab SVM as

$$\frac{1}{2}\|w\|^2 - \rho = \frac{1}{2}\|w\|^2 - \langle w, x_i \rangle = \frac{1}{2}\|w - x_i\|^2 - \frac{1}{2}\|x_i\|^2.$$

Since $\frac{1}{2}\|x_i\|^2$ is constant, i.e., does not depend on w or ρ, we can drop it from the objective function. This gives if we set $w' = w - x_i$ and reformulate the constraints in the new variable w' accordingly the following version of the slab SVM:

$$\begin{aligned} \min_{w,\rho} \quad & \tfrac{1}{2}\|w'\|^2 \\ \text{s.t.} \quad & 0 \leq \langle w', x_j - x_i \rangle + \langle x_i, x_j \rangle - \|x_i\|^2 \leq \delta \\ & \text{for } j \neq i \end{aligned}$$

This problem asks for the shortest vector w' in the constraint polytope or equivalently the distance of the constraint polytope to the origin. Note that this distance problem is also always feasible, i.e., the constraint polytope does not become empty. To see this observe that $w' = -x_i$ is always in the polytope. The gain and lose events can be nicely illustrated for the distance problem, see Figure 3.

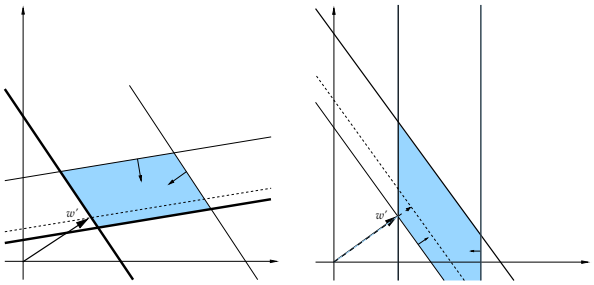


Figure 3: The lower (non-moving) constraints are shown by thick solid lines and the upper (moving) constraints are shown by thin solid lines. On the left: when the moving constraint hits w' this constraint becomes binding (gain event) and the solution is no longer stationary. On the right: once the moving constraint becomes orthogonal to w' we lose the non-moving constraint (lose event).

The formulation of the slab SVM as a distance problem allows to make some observations.

Lemma 1 *The solution to the slab SVM is unique.*

Proof. There is always a unique point in the convex constraint polytope of an equivalent distance problem that realizes the distance of the polytope to the origin. □

Lemma 2 *There exists a δ_0 such that for all $\delta > \delta_0$ the solution to the slab SVM is stationary, i.e. does not vary with δ.*

Proof. The proof is via the distance problem. Let x_i be one of the (lower) supporting data points of the open slab SVM. We use this x_i to formulate the distance problem. The solution of the distance problem at $\delta = \infty$ is finite (we can conclude this from the properties of the open slab SVM). Coming from small values of δ the constraint polytopes of the distance problem for these values of δ sweep the constraint polytope of the distance problem at $\delta = \infty$. Since the solution to the latter is finite the sweep needs to hit the point that realizes this finite distance at some finite value δ_0 of δ. That is, for all $\delta > \delta_0$ the point x_i is lower supporting for the slab SVM and we can conclude that the solution of the slab

SVM can be derived from this stationary solution of the distance problem as $w = w' + x_i$ and $\rho = \langle w, x_i \rangle$. □

Lemma 3 *For all $0 < \delta < \delta_0$ the slab SVM has an upper supporting data point.*

Proof. By the proof of Lemma 2 we have that at δ_0, the slab SVM needs to have an upper supporting data point, because only the upper constraints sweep the constraint polytope of the distance problem at $\delta = \infty$. Assume there exists $0 < \delta < \delta_0$ such that at δ the slab SVM has no upper supporting data point. Let Δ be the set of all δ with this property and let $\delta' = \sup \Delta$. At δ' the slab SVM needs to have an upper supporting data point. To see this note that there exists a data point x_j that is upper supporting at $\delta + \varepsilon$ for all sufficiently small $\varepsilon > 0$. At δ' we can derive a distance problem that is equivalent to the slab SVM for some neighborhood of δ'. The data point x_j needs to be upper supporting also for this distance problems at $\delta' + \varepsilon$ for all sufficiently small $\varepsilon > 0$. The constraint hyperplane given by

$$\langle w', x_j - x_i \rangle + \langle x_i, x_j \rangle - \|x_i\|^2 = \delta' \qquad (1)$$

for the data point x_j has all the constraint hyperplanes given by

$$\langle w', x_j - x_i \rangle + \langle x_i, x_j \rangle - \|x_i\|^2 = \delta' + \varepsilon \qquad (2)$$

on one side. The latter hyperplanes all contain a point that realizes the solution of the corresponding distance problem. By the continuity of the distance problem in δ any sequence in the latter point set converges to the solution of the distance problem at δ'. Hence this solution needs to be contained in the constraint hyperplane given by Equation (1) and x_j is an upper supporting data point for both the distance- and the slab SVM problem at δ'. By our assumption there needs to exist some neighborhood U of δ' such that the distance problem does not have an upper supporting data point for all $\delta \in U \cap (0, \delta')$. This means that the family of hyperplanes given by Equation (2) sweeps with $\varepsilon \to 0$, i.e., at δ', out of the constraint polytope given by the constraints

$$\langle w', x_j - x_i \rangle + \langle x_i, x_j \rangle - \|x_i\|^2 = 0.$$

But this can only happen if the constraint polytope of the distance problem grows while sweeping the hyperplane given by Equation (2) from $\delta' + \varepsilon$ to $\delta' - \varepsilon$, which is a contradiction. □

Corollary 1 *For all $0 < \delta < \delta_0$ the solution to the slab SVM is non-stationary.*

We can conclude that the solution path of the slab SVM is piecewise linear (since w' the point that realizes the distance of the constraint polytope to the origin is a piecewise linear curve parametrized by δ).

Theorem 4 *The solution path of the slab SVM, i.e., the optimal coefficients α_i and β_i (in the dual) and w and ρ (in the primal) are piecewise linear functions of δ.*

Corollary 2 *The optimal solution w to the slab SVM is a piecewise linear path that connects the point closest to the origin on the convex hull (solution at $\delta = \infty$) of the data points with the point closest to the origin on the affine hull (solution at $\delta = 0$) of the data points.*

6 Conclusions

Theorem 4 characterizes the solution path, but does not immediately suggest an algorithm to compute it. But algorithms for parametrized convex quadratic programs (such as the slab SVM) are known, see for example [Rit81].

Acknowledgments. Joachim Giesen wants to thank Edgar Ramos and Bardia Sadri for valuable discussions on the slab SVM.

References

[CBC+01] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruce C. McCallum, and Tim R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH*, pages 67–76, 2001.

[Ede95] Herbert Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13:415–440, 1995.

[ELZ02] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[GCPZ06] Joachim Giesen, Frédéric Cazals, Mark Pauly, and Afra Zomorodian. The conformal alpha shape filtration. *The Visual Computer*, 22(8):531–540, 2006.

[HRTZ04] Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 2004.

[Rit81] Klaus Ritter. On parametric linear and quadratic programming problems. In *Mathematical Programming: Proceedings of the International Congress on Mathematical Programming*, pages 307–335, 1981.

[SGS04] Bernhard Schölkopf, Joachim Giesen, and Simon Spalinger. Kernel methods for implicit surface modeling. In *NIPS*, 2004.

[SS02] Bernhard Schölkopf and Alex Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, 2002.

Adaptive Searching in One and Two Dimensions

Reza Dorrigiv* Alejandro López-Ortiz*

1 Introduction

Searching in a geometric space is an active area of research, predating computer technology. The applications are varied ranging from robotics, to search-and-rescue operations in the high seas [24, 23] as well as in land, such as in an avalanche [5] or an office space [12, 7, 13], to scheduling of heuristic algorithms for solvers searching an abstract solution space for a specific solution [16, 17, 22, 2, 19]. Within academia, the field has seen two marked boosts in activity. The first was motivated by the loss of weaponry off the coast of Spain in 1966 in what is known as the Palomares incident and of the USS Thresher and Scorpion submarines in 1963 and 1966 respectively [24, 26]. A second renewed thrust took place in the late 1980s when the applications for autonomous robots became apparent.

Geometric searching has proved a fertile ground within computational geometry for the design and analysis of search and recognition strategies under various initial conditions [14, 12, 6, 7, 8, 18, 20].

The basic search scenarios consist of exploring a one dimensional object, such as a path or office corridor, usually modeled as the real line, and of exploring a two dimensional scene, such as a room or a factory floor, usually modelled as a polygonal scene. However, in spite of numerous advances in the theoretical understanding of both of these scenarios, so far such solutions have generally had a limited impact in practice.

Over the years various efforts have been made to address this situation, both in terms of isolated research papers attempting to narrow the gap, as well as in organized efforts such as the Algorithmic Foundations of Robotics conference and the Dagstuhl seminars on on-line robotics which bring together theoreticians and practitioners. From these it is apparent that the cost model and hence the solutions obtained from theoretical analysis do not fully reflect real life constraints. Several efforts have been made to resolve this, such as including the turn cost, the scanning cost, and error in navigation and reckoning [9, 10, 15, 20, 18].

In this paper we address one more shortcoming of the standard model. Consider for example a vacuuming robot—such as Roomba(TM). Such a robot explores the environment using sophisticated motion planning algorithms with the goal of attaining complete coverage of the floor surface within a reasonable amount of time. It is not hard to devise worst case floor plans (such as complex mazes) which would not be covered very efficiently. In practice this is not a concern since (i) most rooms are relatively simple and (ii) if the robot ever encounters such a complex scene a drop in performance is only to be expected and users would not mind a severe degradation in performance. This naturally leads to the concept of adaptive algorithms, in which on simpler inputs the robot must perform more efficiently than on more complex ones.

In this paper we consider adaptive analysis of two basic geometric primitives: searching on the real line and looking around the corner.

Searching on the real line consists of finding a target t on the real line located at an unknown distance d (in either direction) from a search robot. The robot detects t upon contact. The optimal strategy visits the rays under a doubling strategy with competitive ratio of 9 [4, 11, 3, 21]. We refer the reader to the survey of Alpern and Gal [1] for a thorough discussion. However upon being presented by the optimal doubling strategy practitioners routinely report that they find the answer non-intuitive and generally "not optimal". This holds for the optimal strategy for either the average or the worst case. There are several non-mutually exclusive explanations for this disparity. In particular we incorporate the observation that in some settings, exploration is a valuable task in which case the goal is to simultaneously minimize the time to the target, and maximize the amount of information gained during the search. For this case we obtain an optimal strategy that is, subjectively, more pleasing to practitioners.

For the second case study we consider searching around a corner. Icking et al. [14] provided an algorithm with competitive ratio $c \approx 1.21218$ and proved that this is the best competitive ratio possible. We extend this result by applying adaptive analysis to this problem.

2 Searching on the Real Line

Without loss of generality, we assume the robot searches starting from the origin x_1 units to left, then it returns to the origin and moves past it x_2 units to the right. In general in the ith phase, it goes x_i units from origin to left or right (depending on the parity of i) and returns to the origin. The search ends when the robot finds the

*Cheriton School of Computer Science, University of Waterloo, {rdorrigiv,alopez-o}@cs.uwaterloo.ca

target. In the *doubling strategy* we have $x_i = 2^{i-1}$. In the standard cost model, we minimize the ratio of the distance travelled by the robot to the straight distance from the target to the origin, which is termed the competitive ratio. As stated before, the doubling method has competitive ratio 9, which is optimal.

In order to reflect the "waste" of robot when traversing a region that has already been explored, we propose a dual cost model. It costs one unit whenever the robot traverses one unit of distance of unknown territory, while it costs c units ($c \geq 1$) when the robot traverses a region that has already been explored.

In order to find the worst case for doubling method under the new cost model, assume that the target is located at distance $2^k + \varepsilon$ from the origin, for some integer k. Therefore robot will find the target at phase $k+3$. For $3 \leq i \leq k+2$, let $C(i)$ be the cost robot incurs at phase i. At phase i, the robot goes 2^{i-1} units away from the origin and then returns to the origin. Of the first 2^{i-1} units, 2^{i-3} units are already explored and $2^{i-1} - 2^{i-3} = 3 \times 2^{i-3}$ units are newly explored. All 2^{i-1} units on the robot's return to the origin are already explored. Therefore we have $C(i) = 2^{i-3}(5c+3)$. Thus the total cost of the first $k+2$ phases is $(1+c) + (2+2c) + \sum_{i=3}^{k+2} 2^{i-3}(5c+3) = (5 \times 2^k - 2)c + 3 \times 2^k$. In the last phase, the robot finds the target at distance $2^k + \varepsilon$, incurring cost $2^k c + \varepsilon$. Thus the competitive ratio of doubling is $\frac{(6\times 2^k - 2)c + 3\times 2^k + \varepsilon}{2^k + \varepsilon}$ which becomes arbitrarily close to $6c+3$ as k grows. Note that for $c = 1$ we get the standard competitive ratio of 9.

Observe that the doubling might no longer be the optimal strategy under the new model. As usual we consider the family of geometric search strategies A_r: we have $x_i = r^{i-1}$ for an arbitrary real number $r > 1$ (the doubling strategy corresponds to $\mathcal{A}_2$). Using arguments similar to the analysis of the doubling method, the cost of robot at phase $3 \leq i \leq k+2$ is $C(i) = r^{i-3}((r^2+1)c + (r^2-1))$ and the total cost of $\mathcal{A}_r$ is $(r + 1 + (r^2+1)(\frac{r^k-1}{r-1}) + r^k)c + (r^2-1)(\frac{r^k-1}{r-1}) + \varepsilon$. Thus the competitive ratio of A_r for this case is $CR(\mathcal{A}_r) = \frac{(r+1+(r^2+1)(\frac{r^k-1}{r-1})+r^k)c + (r+1)(r^k-1) + \varepsilon}{r^k + \varepsilon}$, which becomes arbitrarily close to $(\frac{r^2+r}{r-1})c + r + 1$ as k grows. Through symbolic manipulation, we find out that the competitive ratio is minimized for $r = 1 + \frac{\sqrt{2c+2c^2}}{c+1}$. As c goes to ∞, this optimal value of r goes to $1 + \sqrt{2} = 2.414213\ldots$ with a search cost of $(3+2/\sqrt{2})c + 2 + \sqrt{2} \approx 5.83c + 3.41$. This improves over the $6c+3$ cost of doubling for large c.

Furthermore, this is optimal, as it can be shown using the Gal-Schuierer functional theorem [11, 25] as follows. For any given strategy, let $X = x_0, x_1, x_2, \ldots$ denote the (infinite) sorted sequence of turn points incurred by the strategy. Then using ideas similar to [22] we can lower bound the competitive ratio by $CR \geq cost(ALG)/cost(OPT)$, where $cost(ALG) = (x_0 + cx_0) + (x_1 + cx_1) + (x_2 - x_0 + cx_0 + cx_2) + \ldots + (x_{k+1} - x_{k-1} + cx_{k-1} + cx_{k+1}) + cx_k$, and $cost(OPT) = x_k$. Therefore, we have that

$$CR(X,k) \geq \frac{(c+1)\sum_{i=0}^{k+1} x_i + (c-1)\sum_{i=0}^{k-1} x_i + cx_k}{x_k} \quad (1)$$

Let $X^{+i} = (x_i, x_{i+1}, \ldots)$ denote the suffix of a sequence $X = (x_0, x_1, \ldots)$ starting at x_i.

Theorem 1 ([25]) *Let $X = (x_0, x_1, \ldots)$ be a sequence of positive numbers, r an integer, and $a = \limsup_{n\to\infty}(x_n)^{1/n}$, for $a \in \mathbb{R} \cup \{+\infty\}$. If F_k, $k \geq 0$, is a sequence of functionals which satisfy*
(1) $F_k(X)$ only depends on $x_0, x_1, \ldots, x_{k+r}$,
(2) $F_k(X)$ is continuous, $\forall x_i > 0$, with $0 \leq i \leq k+r$,
(3) $F_k(\alpha X) = F_k(X)$, $\forall \alpha > 0$,
(4) $F_k(X+Y) \leq \max(F_k(X), F_k(Y))$, and
(5) $F_{k+i}(X) \geq F_k(X^{+i})$, $\forall i \geq 1$,
then $\sup_{0\leq k<\infty} F_k(X) \geq \sup_{0\leq k<\infty} F_k(A_r)$.

It is not hard to verify that the hypothesis of the theorem holds for the modified cost model, and hence it suffices to consider x_i of the form r^{i-1} in the expression for $CR(X,k)$ above. Note that the left-hand side of inequality 1 above is precisely the expression we derived when upper-bounding the competitive ratio. Therefore, substituting r with $1+\sqrt{2}$ yields a lower bound on $CR(X,k)$ which is identical to the upper bound, which in turn implies that the geometric strategy with $r = 1 + \sqrt{2}$ is in fact optimal.

We can extend our dual cost model to cases in which $c < 1$, i.e., revisiting is less expensive than discovering. As suggested by an anonymous reviewer, the case $c = 0$ can also be used to model two sequential communicating searchers. If $c < 0$, the robot can reduce its cost by revisiting the discovered territories forever and no optimal strategy exists. For $0 < c < 1$, we can use an analysis analogous to the case $c \geq 1$ to show that A_r with $r = 1 + \frac{\sqrt{2c+2c^2}}{c+1}$ is optimal. For $c = 0$, the optimal strategy is A_r with $r = 1 + \varepsilon$ for a very small constant ε and this leads to the competitive ratio $2 + \varepsilon$.

3 Looking Around a Corner

In this particular case we consider the setting in which the robot is exploring a man made setting in which there is a preferential occurrence for orthogonal and near orthogonal angles. We wish to explore the change in the nature of the solution when this assumption is made.

We follow the same approach as [14] and formulate the problem using a differential equation. Therefore we use similar terminology and notation and just highlight the differences between the methods; refer to [14] for omitted details. First we formally define the problem. Figure 1 shows a typical instance of the problem. The

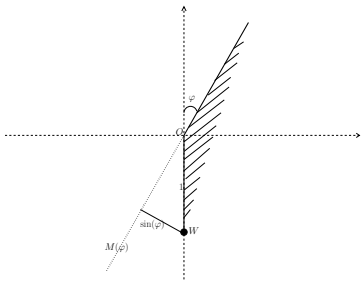


Figure 1: A typical instance of the corner problem.

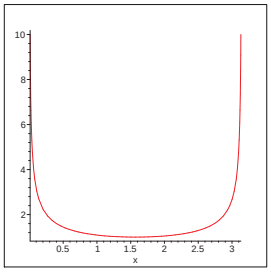


Figure 2: Plot of $d(\varphi) = \frac{1}{\sqrt{\sin\varphi}}$ for $0 < \varphi < \pi$.

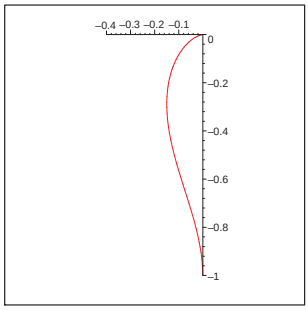


Figure 3: Robot's Optimal Path in the New Model.

corner is placed at the origin O and one of its halflines coincides with the negative y axis. The other halfline of the corner makes an angle $0 \le \varphi \le \pi$ with the positive y axis. A mobile robot is located at point $W = (0, -1)$ and is equipped with an on-board vision system facing O. When $\varphi > 0$, the robot cannot see the other halfline (wall) of the corner and his goal is to discover that (invisible) halfline by minimum movement. The robot sees the invisible line the first time it visits any point on the prolongation $M(\varphi)$ of the invisible line. Let $a(\varphi)$ be the distance between W and $M(\varphi)$. We have

$$a(\varphi) = \begin{cases} \sin\varphi & \text{if } 0 \le \varphi \le \pi/2 \\ 1 & \text{if } \pi/2 < \varphi \le \pi \end{cases} \qquad (2)$$

If the robot knows φ then it can discover the invisible wall by the optimal movement $a(\varphi)$. However this is not the case and the robot should come up with a strategy S that works for all $0 \le \varphi \le \pi$. Let $A_S(\varphi)$ be the length of the path generated by S from W to the first point on $M(\varphi)$. Then the *competitive function* of S is defined as $f_S(\varphi) = \frac{A_S(\varphi)}{a(\varphi)}$ and the *competitive factor* of S is defined as $c_S = \sup_{\varphi\in(0,\pi]} f_S(\varphi)$.

In practical robot navigation most corners have angles close to $\pi/2$ and usually we do not have angles close to 0 or π. As a first attempt for applying adaptive analysis ideas we consider $d(\varphi) = 1/\sqrt{\sin\varphi}$ as difficulty measure. Figure 2 shows the behaviour of $d(\varphi)$ for $0 < \varphi < \pi$. We normalize the competitive function further by $d(\varphi)$ and the new competitive function is defined as $g_S(\varphi) = \frac{f_S(\varphi)}{d(\varphi)} = \begin{cases} \frac{A_S(\varphi)}{\sqrt{\sin\varphi}} & \text{if } 0 \le \varphi \le \pi/2 \\ A_S(\varphi)\sqrt{\sin\varphi} & \text{if } \pi/2 < \varphi \le \pi \end{cases}$

Icking et al. [14] describe the strategies by curves of form $S = (\varphi, s(\varphi))$ in polar coordinates about O that satisfy certain properties, e.g., $s(0) = 1$. They show that the optimal competitive strategy is given by the solution to

$$f_R(\varphi) = \frac{A_R(\varphi)}{\sin\varphi} = c,$$

for all $\varphi \in [0, \pi/2]$ and for some constant c (the smallest c if there are several solutions). For our cost model, the corresponding equation becomes $g_R(\varphi) = \frac{A_R(\varphi)}{\sqrt{\sin\varphi}} = c$. We have $A_R(\varphi) = c\sqrt{\sin\varphi} \Rightarrow \frac{c\cos\varphi}{2\sqrt{\sin\varphi}} = A_R'(\varphi) = \sqrt{r'^2(\varphi) + r^2(\varphi)} \Rightarrow r'(\varphi) = -\sqrt{\frac{c^2\cos^2\varphi}{4\sin\varphi} - r^2(\varphi)}$. We take the negative square root because in an optimal strategy the robot should always come closer to the corner. By replacing $r(\varphi)$ by $cu(\varphi)$ we get the differential equation

$$u'(\varphi) + \sqrt{\frac{\cos^2\varphi}{4\sin\varphi} - u^2(\varphi)} = 0, \qquad (3)$$

with initial condition $u(0) = 1/c$. Therefore, our problem reduces to:

Problem Find the minimum $c > 1$, such that the ordinary differential equation (3) has a solution on some interval $[0, \sigma] \subseteq [0, \pi/2]$, subject to the following constraints:

$$u(0) = 1/c \qquad u(\varphi) > 0 \quad \text{for } \varphi \in [0, \sigma] \qquad u(\sigma) = 0$$

Since this type of differential equations generally do not have a closed form we use numerical methods to compute the solution $c \approx 1.08$. The strategy with this competitive factor is shown in Figure 3. We can prove the optimality of this strategy using arguments analogous to [14].

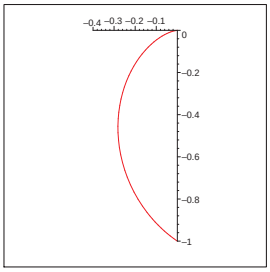


Figure 4: Robot's optimal path in the previous model.

The optimal strategy in the standard model is shown in Figure 4. It has competitive factor ≈ 1.21 [14]. Observe that since less weight is given to small angles the solution takes a shorter path to reach sightlines for angles around $\pi/4$.

Acknowledgements We thank Spyros Angelopoulos and anonymous referees for their comments.

References

[1] S. Alpern, S. Gal, The Theory of Search Games and Rendezvous, Kluwer Academic Publishers, 2003.

[2] S. Angelopoulos, A. López-Ortiz, A. M. Hamel, Optimal scheduling of contract algorithms with soft deadlines, in: AAAI, AAAI Press, 2008.

[3] R. Baeza-Yates, J. Culberson, G. Rawlins, Searching in the plane, Inf.&Comp. 106 (1993) 234–252.

[4] A. Beck, On the linear search problem, Israel Journal of Math. 2 (1964) 221–228.

[5] T. C. Biedl, M. Hasan, J. D. Horton, A. López-Ortiz, T. Vinar, Searching for the center of a circle, in: CCCG, 2002.

[6] K. Chan, T. W. Lam, An on-line algorithm for navigating in an unknown environment, Int. J. Comput. Geom. Appl. 3 (1993) 227–244.

[7] A. Datta, C. Hipke, S. Schuierer, Competitive searching in polygons—beyond generalized streets, in: Proc. 6th ISAAC, LNCS 1004, 1995.

[8] A. Datta, C. Icking, Competitive searching in a generalized street, in: Proc. 10th SoCG, 1994.

[9] E. D. Demaine, S. P. Fekete, S. Gal, Online searching with turn cost, Theor. Comput. Sci. 361 (2-3) (2006) 342–355.

[10] S. P. Fekete, R. Klein, A. Nüchter, Online searching with an autonomous robot, Comput. Geom. Theory Appl. 34 (2) (2006) 102–115.

[11] S. Gal, Search Games, Academic Press, 1980.

[12] F. Hoffmann, C. Icking, R. Klein, K. Kriegel, The polygon exploration problem, SIAM J. Comput 31 (2) (2001) 577–600.

[13] C. Icking, Motion and visibility in simple polygons, Dissertation, Fernuniversität Hagen (1994).

[14] C. Icking, R. Klein, L. Ma, How to look around a corner, in: CCCG, 1993.

[15] T. Kamphans, Models and algorithms for online exploration and search, Dissertation, University of Bonn (2005).

[16] M.-Y. Kao, Y. Ma, M. Sipser, Y. Yin, Optimal constructions of hybrid algorithms, in: Proc. 5th SODA, 1994.

[17] M.-Y. Kao, J. H. Reif, S. R. Tate, Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem, in: Proc. 4th SODA, 1993.

[18] A. López-Ortiz, On-line target searching in bounded and unbounded domains, Ph.D. thesis, Dept. of Comp. Sci., University of Waterloo (1996).

[19] A. López-Ortiz, S. Angelopoulos, A. M. Hamel, Optimal scheduling of contract algorithms for anytime problems, in: AAAI, AAAI Press, 2006.

[20] A. López-Ortiz, S. Schuierer, Simple, efficient and robust strategies to traverse streets, in: Proc. 7th CCCG, 1995.

[21] A. López-Ortiz, S. Schuierer, The ultimate strategy to search on m rays?, Theor. Comput. Sci. 261 (2) (2001) 267–295.

[22] A. López-Ortiz, S. Schuierer, Online parallel heuristics and robot searching under the competitive framework, in: SWAT, 2002.

[23] A. López-Ortiz, G. Sweet, Parallel searching on a lattice, in: CCCG, 2001.

[24] H. R. Richardson, L. D. Stone, Operations analysis during the underwater search for Scorpion, Naval Research Logistics Quarterly 18 (2) (1971) 141–157.

[25] Schuierer, Lower bounds in on-line geometric searching, Computational Geometry: Theory and Applications 18 (1) (2001) 37–53.

[26] L. D. Stone, What's happened in search theory since the 1975 Lanchester prize?, Operations Research 37 (3) (1989) 501–506.

Competitive Search for Longest Empty Intervals

Peter Damaschke*

Abstract

A problem arising in statistical data analysis and pattern recognition is to find a longest interval free of data points, given a set of data points in the unit interval. We use the inverse length of the empty interval as a parameter in the complexity bounds, since it is small in statistically relevant cases. For sorted point sets we get nearly optimal strategies. While the asymptotic complexities are trivial, achieving an optimal number of operations appears to be difficult. Constant factors can be of practical interest for huge data sets. We derive deterministic and randomized upper and lower bounds. Matching bounds and smooth trade-offs between the different operations (reads, comparisons, subtractions) are open questions. For unsorted point sets, the complexity is at least linear. Therefore we also use statistical inference to get approximate solutions in sublinear time.

1 Introduction

Given a set of n data points in a finite-size part of a geometric space, we call a subset of this space (with prescribed shape) free of data points an empty region. Searching for largest empty regions is a natural problem in, e.g., data mining [10]. It has been considered for rectangles in the plane [1, 2, 5, 8, 11] and boxes in d dimensions. Usually, the complexity of algorithms is expressed as a function of input size n. However, empty regions are statistically relevant only if they are large compared to the expected size if the data point set were drawn from a uniform distribution. Then, large empty regions may be found faster than in the worst case. Thus it is sensible to measure complexity as a function of both n and a parameter inverse to the size of the empty region. Here we study, as a first step, the 1-dimensional case: empty intervals between n data points in the unit interval. While the worst-case complexity is trivially $\Theta(n)$, the parameterized problem has a different nature. Still, its optimal asymptotic complexity is easy to determine by standard arguments, but the exact number of operations appears to be a surprisingly difficult question. Constant factors make a difference in practice, when huge data sets are processed. Analyzing the number of operations (e.g., comparisons) without ignoring constant factors is quite common for sorting, searching, and order statistics.

We state our problem Longest Empty Interval more formally. A sorted set of real numbers $0 = x_0 < x_1 < \ldots < x_n = 1$ is given. An *empty interval* is an interval delimited by two consecutive x_i, x_{i+1}. We can access x_i through index i in constant time. (The x_i are either stored in an array or delivered by an oracle.) Our goal is to find a longest empty interval, that is, one with largest difference $\max_i(x_{i+1} - x_i)$. This can be trivially done by n read operations (*reads* for short), subtractions, and comparisons, respectively, and linear time is optimal due to an obvious adversary argument. Define $r := 1/\max_i(x_{i+1} - x_i)$. Supposing that a "very long" empty interval is expected, with $r \ll n$, we want an algorithm that takes advantage of the small r.

Throughout the paper, logarithms are base 2. We call the x_k values *data points*. In our complexity bounds we neglect minor-order terms. To avoid clumsy notation we also silently suppress factors $1 + o(1)$ where $o(1)$ tends to 0 as n grows.

We show that Longest Empty Interval can be solved optimally with $r \log(n/r)$ reads. However, in order to keep the number of other operations within $O(r \log(n/r))$ we need some more reads. We have to add factor 2 (deterministic) or 1.4427 (randomized). We also study the case of unsorted data point sets, called Longest Empty Interval (unsorted). Amazingly, n and r almost switch their roles: We give an algorithm with roughly $n \log r$ comparisons, while the number of reads is trivially n. We remark that a rather obvious RAM algorithm using n equidistant buckets solves Longest Empty Interval (unsorted) in $O(n)$ time, but for comparison-based algorithms $\Theta(n \log r)$ is optimal, and the simple scheme also fails for similar problems in higher dimensions. The problem is also known as *max gap* and has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model [3, 9]. Our algorithms do not assume prior knowledge of r. Another practical advantage is their simplicity, however, several details leading to the constant factors are a bit tricky, and there remain gaps between the current upper and lower bounds. In the unsorted case, approximate solutions, i.e., large regions with few data points, can still be obtained in sublinear time. We give a grid-based method to analyze the performance of an obvious sampling method.

*Department of Computer Science and Engineering, Chalmers University, 41296 Göteborg, Sweden, email: ptr@cs.chalmers.se.
Supported by the Swedish Research Council (Vetenskapsrådet), grant no. 2007-6437, "Combinatorial inference algorithms – parameterization and clustering".

The last section informally discusses extensions to other geometric set families.

We conclude the introduction with some motivations and further related literature.

In the sorted case one may argue that the longest empty interval *could have been* computed on the fly, when the set has been sorted, and this makes up a minor part of the calculations. But what if distances in our huge sorted point sets *have not been* computed earlier, simply because there was no interest in such analysis? Then we want to solve the *actual* problem as efficiently as possible. There may also arise machine learning problems where we know that some unknown "empirical" function is monotone, values are not explicitly stored but can be queried by experiments, and we are mainly interested in large jumps of this function. In fact, Longest Empty Interval exhibits striking similarities to a well-known problem in combinatorial search: competitive group testing [6, 7].

In [4] we gave algorithms for finding at most s disjoint intervals of maximum total length that contain at most p data points (s, p are fixed parameters). Finding longest empty intervals in sorted point sets is part of the preprocessing. Then, it is proved that the optimal solutions are composed of such intervals from a certain candidate set whose size depends only on s and p, and it can be computed by dynamic programming. Only the time for preprocessing depends on n, therefore we save a significant fraction of the overall running time by log-time preprocessing. In range prediction applications as in [4], the data points come as previously sorted sets.

2 The Sorted Case

Theorem 1 Longest Empty Interval *can be solved using* $r\log(n/r)$ *reads, and this bound is optimal.*

Proof. The optimality argument is omitted.

The proposed algorithm maintains, in a linked list, the ordered sequence of data points x_k already read. In every step we take two consecutive data points in this list with currently largest distance, say x_a and x_b, read the data point $x_{\lfloor(a+b)/2\rfloor}$ and insert it in our list. We stop as soon as $a+1=b$. Since $x_b - x_a$ is the maximal distance in the sequence, we have found the longest empty interval at this moment.

To analyze the number of reads, think of this splitting process as a binary tree of segments of data points, in the obvious sense. One read is associated with every non-leaf node. Consider the tree upon termination of the algorithm. A *long* node represents an interval of length at least $1/r$, other nodes are called *short*. We prune the tree as follows. Any pair of short leaf siblings is removed, making their parent a leaf. The parent node is always long, since the algorithm has considered intervals by decreasing lengths and stopped at $1/r$. After pruning, one read is associated with every long node. Since the leaves represent pairwise disjoint intervals, at most r leaves are long nodes. Every long non-leaf node is on some path from the root to some long leaf (otherwise we could continue pruning). It follows that all reads are associated with nodes on paths to at most r of the leaves. The path length in the tree is trivially bounded by $\log n$. At most r nodes have depth $\log r$, and the remaining subpaths from level $\log r$ to the leaves have length at most $\log n - \log r$. Since at most r such paths exist, we get the claimed bound. □

However we have to worry about the other operations, too. Upon every read we also need two subtractions to get the lengths of the two new intervals. Thus, the method needs $2r\log(n/r)$ subtractions. The catch is that we need to know the longest interval for the next split. Using a heap for at most r interval lengths (the current leaves of the tree), we make, for every read, up to $4\log r$ length comparisons to include the two new interval lengths in the heap (and also $5\log r$ copy operations in the heap). Thus the method in this form costs $4r\log r\log(n/r)$ comparisons. An optimal number of reads is good if data access is very expensive, e.g., if data reside in some external memory. But usually the costs of reads, comparisons, and subtractions should be similar. Thus we will next aim at $O(r\log(n/r))$ operations in total, with small constant factors. We now propose a method that still uses binary search, but on the range of values rather than indices. The number of reads is only doubled.

Theorem 2 Longest Empty Interval *can be solved using* $2r\log(n/r)$ *reads,* $2r\log(n/r)$ *comparisons, and* $O(r)$ *subtractions.*

Proof. In the jth phase ($j = 1, 2, 3\ldots$), we declare every $i/2^j$ (i odd, $0 < i < 2^j$) a *grid point*. For every new grid point g, binary search finds k with $x_k \le g < x_{k+1}$. We call $[x_k, x_{k+1}]$ the *empty interval around* g. We compute the lengths of empty intervals around all grid points and determine the longest one.

Let p be the exponent with $1/2^p \le 1/r < 1/2^{p-1}$. Then, a longest empty interval (of length $1/r$) contains a grid point in phase p. Since we have computed the lengths of empty intervals around all grid points, $1/r$ is among these values, and it is the maximum length. Since every empty interval without grid points is entirely between two consecutive grid points, its length is at most $1/2^p \le 1/r$, hence we know at this moment that a longest empty interval is found.

In order to find the empty interval around any new grid point introduced in phase j, it suffices to do binary search on the data points between the two neighbored old grid points. (Recall that we already know the indices of the leftmost and rightmost data point in this range.)

Since all these search spaces do not overlap, we perform 2^{j-1} binary search procedures on a total of n elements in phase j. By concavity of log, the total number of search steps in phase j is maximized if all search spaces have equal size $n/2^{j-1}$. Summation over all phases yields the number of operations: $\sum_{j=1}^{p} 2^{j-1}(\log \frac{n}{2^{j-1}} + O(1)) = 2^p(\log n - p + O(1))$.

The worst case is $1/r < 1/2^{p-1}$, with an arbitrarily small difference. Now $2^p < 2r$ yields the upper bound of $2r \log(n/r)$ search steps. Every search step requires one read and one comparison. Subtractions are only used to compute the lengths of empty intervals around the $O(r)$ grid points. Only $O(r)$ comparisons are needed to determine the maximum length among them. □

The worst case in the above analysis suggests that randomization on the grid size might improve the constant factor in the number of reads. In fact, we obtain:

Theorem 3 Longest Empty Interval *can be solved using an expected number of* $(1/\ln 2)r\log(n/r)$ *reads,* $(1/\ln 2)r\log(n/r)$ *comparisons, and* $O(r)$ *subtractions. (Remark:* $1/\ln 2 < 1.4427$*.)*

Proof. We sample a random $t \in [1,2)$ according to some probability density function q that we specify below, multiply the grid point distances by t, and continue deterministically as in Theorem 2. For formal clarity: We construct the grid on an interval of length t including $[0,1]$, but then we ignore all grid points outside $[0,1]$.

As in Theorem 2, let p be the exponent with $1/2^p \le 1/r < 1/2^{p-1}$. If $t \le 2^p/r$ then we also have $t/2^p \le 1/r < t/2^{p-1}$. Now we argue, as in Theorem 2, that an empty interval of length $1/r$ is identified in phase p. However, since grid points outside the unit interval are ignored, we perform only $2^{j-1}/t$ binary search procedures on disjoint subsets of a set of n elements, in phase j. The total number of search steps in phase j is maximized if all search spaces have equal size $tn/2^{j-1}$. Summing over all phases we get $\frac{1}{t}\sum_{j=1}^{p} 2^{j-1}\left(\log \frac{tn}{2^{j-1}} + O(1)\right) = \frac{2^p}{t}(\log n - p + O(1))$.

If $t > 2^p/r$ then $t/2^{p+1} \le 1/r < t/2^p$. Still we can argue as above, but with $p+1$ in the role of p, which yields the result $(2/t)2^p(\log n - p + O(1))$.

Define $x := 2^p/r$, and note that $1 \le x < 2$. We express the number of reads as $(x/t)r\log(n/r)$ if $t \le 2^p/r$, and $2(x/t)r\log(n/r)$ if $t > 2^p/r$. Specifically, we use density $q(t) = 1/(t\ln 2)$ for sampling. (In fact, q is a density function, due to $\int_1^2 dt/t = \ln 2$). Thus we obtain in front of $r\log(n/r)$ the following expected factor: $x\left(\int_1^x \frac{1}{t}q(t)dt + 2\int_x^2 \frac{1}{t}q(t)dt\right) = \frac{x}{\ln 2}\left(\int_1^x \frac{1}{t^2}dt + 2\int_x^2 \frac{1}{t^2}dt\right) = \frac{x}{\ln 2}\left(\frac{1}{1} - \frac{1}{x} + \frac{2}{x} - \frac{2}{2}\right) = \frac{1}{\ln 2}$. The other bounds follow as in Theorem 2. □

It remains open, even in the randomized case, whether $r\log(n/r)$ reads are sufficient together with $O(1)r\log(n/r)$ other operations. More generally, a smooth trade-off between reads and comparisons would be nice. Apparently this would require to "bridge" somehow between binary search on indices and values.

3 The Unsorted Case

In order to solve Longest Empty Interval (unsorted), we have to read all n data points x_i, since any missing x_i could fall into the largest empty interval of the rest of the data set. Hence the number of reads is not interesting. We focus on comparisons and subtractions. Trivially, sorting the x_i solves the problem by $n\log n$ comparisons and n subtractions, but for $r \ll n$ we can avoid sorting and save almost a $\log n$ factor:

Theorem 4 Longest Empty Interval (unsorted) *can be solved using* $n(\log r + 3) + 4r$ *comparisons and* $O(r)$ *subtractions, and* $n\log r$ *is a lower bound for the number of comparisons.*

Proof. Again we perform binary search on $[0,1]$, inserting grid points $i/2^j$ (i odd) in phase j, but this time we divide the data points recursively into subsets situated between any two neighbored grid points. If j phases are needed, this costs altogether nj comparisons between data points and grid points. After each phase we check which of the mentioned subsets became empty. This step is simple: To every new grid point we attach a discrete variable that tells us whether some data point went to the left and to the right subset. As soon as we get some empty subset(s) in our partitioning, we know that the largest empty interval is formed by the rightmost data point in some nonempty subset and the leftmost data point in the next nonempty subset to the right. All candidates are found by n comparisons in total, because the linear order of subsets is known, and minimum resp. maximum search is done on disjoint subsets. If j is the final phase, at most 2^j subtractions yield the interval lengths, and 2^j further comparisons return the result.

Once more, let p be the exponent with $1/2^p \le 1/r < 1/2^{p-1}$. We detect an empty subset when two grid points hit the largest empty interval, which happens in phase $j \le p+1$. Hence $j < \log r + 2$, furthermore $2^j \le 2^{p+1} < 4r$. Summation of comparisons in binary search and candidate selection yields the bound. The lower bound argument is omitted. □

It is not possible to find exactly the largest empty interval in sublinear time. On the other hand, for statistical inference and data mining, a relaxed optimization goal is still appropriate: Find a large interval containing at most a given fraction of data points (as in [4]). Then we can sample from the data points and estimate the point numbers in intervals. The question is how reliable the inferred "sparse" intervals are.

For technical reasons we further modify the problem statement in two ways, without changing its "essence": Firstly, instead of a huge set of data points we assume an unknown *continuous* probability distribution on $[0,1]$ to sample from. Secondly, instead of searching for an interval with given probability mass q and maximum length L, we search for an interval with given L and minimum q. (Note that the length of an interval is "observable", whereas probability mass can only be estimated.) Now we can measure the performance simply by the competitive ratio q_A/q, where q_A is the probability mass of the interval selected by the algorithm, and q is the minimal probability mass among all intervals of length L. We get the following trade-off, with $\delta = q_A/q - 1$:

Theorem 5 *Given some $L < 1$ and an unknown probability distribution on the unit interval, let q be the minimum probability mass of the intervals of length L. Then one can, in $O(m \log m)$ time, sample an expected number of m points and specify an interval of length L with probability mass smaller than $(1+\delta)q$, subject to an error probability less than $\frac{h}{q}(1+1/\delta)\exp(-mq\frac{(\delta-2/h)^2}{4+2\delta})$, for any positive δ and h.*

Proof. Sample m points and take an interval A of length L with least number of sampled points. The probability of $q_A > (1+\delta)q$ is limited by a union bound, applied to a finite "grid" of intervals G so that every too heavy A contains some G. Details are omitted. □

After a slight refinement of the proof we can replace factor $\frac{h}{q}$ with the smaller $\frac{h}{L}$. The free parameter h may be choosen so as to minimize the error bound. In particular, taking $h = mq\delta$ gives the best asymptotics for large m. Here we obtain $m(1+\delta)\exp(-mq\frac{\delta^2}{4+2\delta})$. For a given sample size m, the bound can also be used to compute $1+\delta$ that are achievable with high probability, depending on q. For very small q, these δ are large, however, the "absolute" probability mass $q(1+\delta)$ of the returned interval is more interesting than the competitive ratio in this case.

4 Further Research: Other Geometric Set Families

It remains to improve the various complexity and probability bounds and to close the gaps. In this paper we have focused on intervals, but the ideas are much more general. In the final remarks we sketch some extensions to be considered in further research. The k longest empty intervals, as needed in [4], can be found by slight modifications of our strategies. Bounds are similar, when $1/r$ is redefined as the length of the k-th longest empty interval. For analoguous problems in d-space, e.g., the largest empty (axis-parallel) box in $[0,1]^d$, a scheme as in Theorem 4 still works, with some relaxation: Since we lack total order, we cannot get optimal results in $o(n \log n)$ time, but $(1-1/s)$-approximations in $O(n(\log r + \log s))$ time, where $1/r$ is the volume of the result. Hidden factors depend on d. The sampling approach of Theorem 5 works similarly for other geometric set families $\mathcal{F}$, too, once there is an efficient algorithm for finding large sets in $\mathcal{F}$ with few data points. A technical difficulty of the analysis is to define suitable "grids": For any probability distribution we need a finite family $\mathcal{G}$ so that every set of $\mathcal{F}$ has a subset in $\mathcal{G}$ with small loss of probability mass. Granularity can be chosen so as to minimize the union bound. The cardinality of $\mathcal{G}$ appears as a factor, but loss affects the negative exponent in the exp term. For unions of s intervals (s fixed) we can proceed as in Theorem 5, only the loss is multiplied by s, since we cut intervals at each end, and the cardinality of $\mathcal{G}$ goes as $(h/q)^{2s-1}$. For boxes in d-space we can simply use slices in the d axis directions. For families $\mathcal{F}$ like disks or balls, grid construction is possible, too, but more complex, since "heavy" borders of sets in $\mathcal{F}$ must be sliced.

References

[1] A. Aggarwal, S. Suri. Fast algorithms for computing the largest empty rectangle, *Symp. on Comput. Geometry 1987*, 278-290

[2] M.J. Atallah, G.N. Frederickson. A note on finding a maximum empty rectangle, *Discrete Applied Math.* 13 (1986), 87-91

[3] M. Ben-Or. Lower bounds for algebraic computation trees. *15th ACM STOC 1983*, 80-86

[4] A. Bergkvist, P. Damaschke. Fast algorithms for finding disjoint subsequences with extremal densities, *Pattern Recognition* 39 (2006), 2281-2292, abstract in: *16th ISAAC 2005*, *LNCS* 3827, 714-723

[5] B. Chazelle, L.R.S. Drysdale, D.T. Lee. Computing the largest empty rectangle, *SIAM J. Comp.* 15 (1986), 550-555

[6] D.Z. Du, H. Park. On competitive group testing, *SIAM Journal Comp.* 23 (1994), 1019-1025

[7] D.Z. Du, G. Xue, S.Z. Sun, S.W. Cheng. Modifications of competitive group testing, *SIAM Journal Comp.* 23 (1994), 82-96

[8] J. Edmonds, J. Gryz, D. Liang, R.J. Miller. Mining for empty rectangles in large data sets, *Theor. Computer Science* 296 (2003), 435-452

[9] D.T. Lee, Y.F. Wu. Geometric complexity of some location problems, *Algorithmica* 1 (1986), 193-211

[10] B. Liu, L.P. Ku, W. Hsu. Discovering interesting holes in data, *15th IJCAI 1997*, 930-935

[11] M. Orlowski. A new algorithm for the largest empty rectangle problem, *Algorithmica* 5 (1990), 65-73

Erratum for "Disjoint Segments have Convex Partitions with 2-Edge Connected Dual Graphs"

Nadia M. Benbernou* Erik D. Demaine† Martin L. Demaine‡ Michael Hoffmann§
Mashhood Ishaque¶ Diane L. Souvaine‖ Csaba D. Tóth**

A set of n disjoint line segments in the plane and a permutation π of the $2n$ segment endpoints define a partition of the plane into convex faces: extend the segments beyond their endpoints one-by-one in the order given by π until they hit another segment, a previous extension, or infinity. If no three segment endpoints are collinear, then every permutation π produces $n+1$ convex faces.

For convex partition, the *dual graph* is defined where the $n+1$ convex faces correspond to the vertices, and every segment endpoint corresponds to an edge between the two incident faces on opposite sides of the segment.

In [1], we presented a partition algorithm (see below) that, for a set S of n disjoint line segments, computes a nonempty subset $S' \subseteq S$ and a convex partition P' of S' such that each remaining segment in $S \setminus S'$ lies in the interior of a face of P'. We claimed that the dual graph of P' is 2-edge connected. This claim is false. Sometimes the dual graph of P' has a bridge (Fig 1).

Partition Algorithm. Input S.

- Pick a segment $s_0 = a_0b_0$ with an endpoint b_0 along $\mathrm{conv}(\cup S)$. Set $s := s_0$, $p := a_0$, $\gamma := 0$, $S' := \{s_0\}$, and $i := 1$.
- Repeat while $p \neq b_0$:

  Extend s beyond p into a ray $\overrightarrow{r}$ until it hits another segment, a previous extension, or to infinity.

  - If $\overrightarrow{r}$ hits a segment in $S \setminus S'$, denote it by $s_i = a_ib_i$ such that $\angle(\overrightarrow{r}, \overrightarrow{a_ib_i}) < 0 < \angle(\overrightarrow{r}, \overrightarrow{b_ia_i})$, let $\gamma_i = \gamma + \angle(\overrightarrow{r}, \overrightarrow{a_ib_i})$, put $S' := S' \cup \{s_i\}$, $s := s_i$, $p := a_i$, $\gamma := \gamma_i + \pi$, and $i := i + 1$.
  - Else, over all integers j, $0 \leq j < i$, such that $s_j \in S'$ has not been extended beyond b_j, pick one where the turning angle γ_j is maximal. Set $s := s_j$, $p := b_j$, and $\gamma := \gamma_j$.

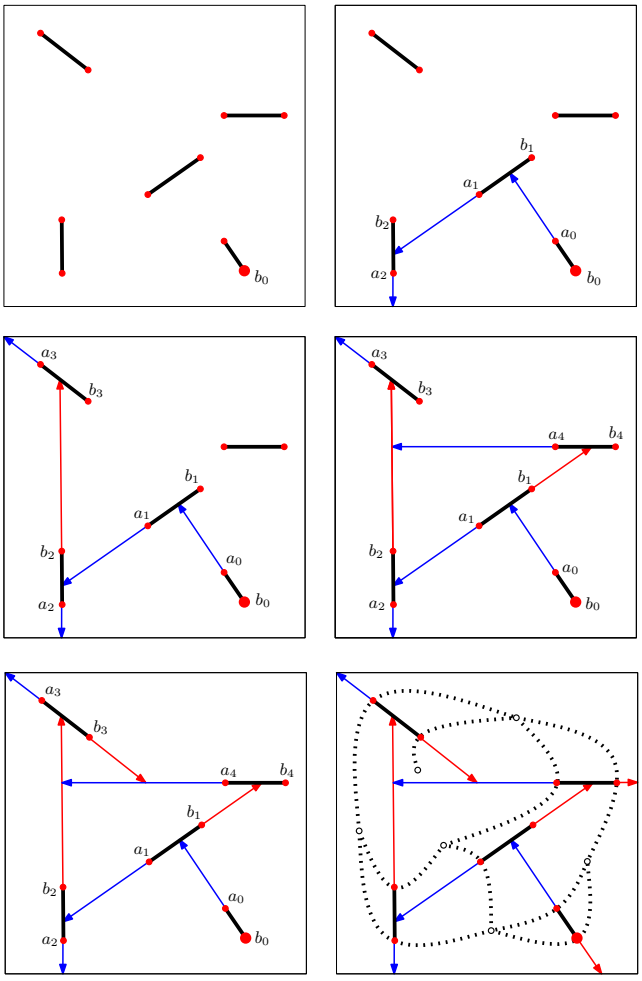


Figure 1: Steps of our partition algorithm for five input segments, and the resulting dual graph.

Specifically, the last phrase in the "proof" for our Lemma 4 is false. It is not true that after a ray $\overrightarrow{r}$ hits a segment a_ib_i, no extension can hit $\overrightarrow{r}$ from the right before the extension a_ib_i beyond b_i is drawn.

References

[1] N. M. Benbernou, E. D. Demaine, M. L. Demaine, M. Hoffmann, M. Ishaque, D. L. Souvaine, and Cs. D. Tóth, Disjoint segments have a convex partition with a 2-edge connected dual graph, in *Proc. 19th Canadian Conf. Comp. Geom.*, 2007, Ottawa, ON, pp. 13–16.

*Massachusetts Institute of Technology, nbenbern@mit.edu
†Massachusetts Institute of Technology, edemaine@mit.edu
‡Massachusetts Institute of Technology, mdemaine@mit.edu
§ETH Zürich, hoffmann@inf.ethz.ch
¶Tufts University, mishaq01@cs.tufts.edu
‖Tufts University, dls@cs.tufts.edu
**University of Calgary, cdtoth@ucalgary.ca

Author Index

Abam, Mohammad Ali 95
Abu Affash, Karim 147
Aichholzer, Oswin 75
Akl, Selim 127
Aloupis, Greg 79, 107
Alsalih, Waleed 199
Arkin, Esther 135

Barbay, Jérémy 47
Benbernou, Nadia 15, 223
Benton, Alex 35
de Berg, Mark 95
Bhattacharya, Bhargab B. 87
Bhattacharya, Binay 119
Bose, Prosenjit 55, 107, 167
Bremner, David 23

Cardinal, Jean 79
Chen, Eric Y, ... 47
Chitsaz, Hamid Reza 179
Collette, Sébastien 79

Daescu, Ovidiu 191
Damaschke, Peter 219
Das, Sandip 103
Demaine, Erik 183
Demaine, Erik D. 139, 223
Demaine, Martin L. 139, 223
Derryberry, Jonathan 163
Dorrigiv, Reza 215
Dujmović, Vida 107
Dumitrescu, Adrian 67, 71

Eigensatz, Michael 211
El-Khechen, Dania 131
Elbassioni, Khaled 171
Evans, William 207

Fabila-Monroy, Ruy 75
Fevens, Thomas 131
Flores-Peñaloza, David 75
Fraser, Maia 195
Fujimoto, Youichi 39

Ghodsi, Mohammad 51
Giesen, Joachim 211
Goswami, Partha P. 103
Graham, Ron 1
Gray, Chris 107
Guettler, Gerhard 19
Gupta, Prosenjit 7

Hackl, Thomas 75
Hart, George 135
Hart, Vi 139
Hoffmann, Michael 223
Hudson, Benoît 115
Huemer, Clemens 75
Hurtado, Ferran 79

Iacono, John 131
Ishaque, Mashhood 15, 223
Islam, Kamrul 127, 199

Janardan, Ravi 7
Janssen, Augustus J.E.M. 91
Jiang, Minghui 71
Jovanovic, Natasa 91

Karpinski, Marek 11
Katz, Matthew J. 147
Kesh, Deepanjan 43
Kim, Joondong 135
King, James 27
Korst, Jan 91
Kostitsyna, Irina 135
Kranakis, Evangelos 195
Kumar, Yokesh 7
Kurdia, Anastasia 191

Langerman, Stefan 79, 107, 167
LaValle, Steven M. 179
Lenchner, Jonathan 23
Liotta, Giuseppe 23
López-Ortiz, Alejandro 215
Lubiw, Anna 155

Magen, Avner 151
Mallows, Colin 19
Manjunath, Madhusudan 211
McCreath, Eric 83
Mehta, Shashank 43
Meijer, Henk 127, 143
Miller, Gary 175
Mitchell, Joseph 135
Moharrami, Mohammad 151
Motoki, Mitsuo 39
Mukhopadhyaya, Krishnendu 87

Núñez Rodríguez, Yurai 143, 199
Näher, Stefan 203
Nandy, Subhas 87
Nekrich, Yakov 11
Nouri Bygi, Mojtaba 51

O'Kane, Jason 179
O'Rourke, Joseph 35, 55, 79, 99, 183

Pal, Manjish 63
Paul, Christophe 23
Phillips, Todd 175
Poon, Sheung-Hung 95
Pouget, Marc 23

Rappaport, David 143
Razaghpour, Mina 155
Rote, Günter 131
Roy, Sasanka 167

Sabhnani, Girishkumar 135
Schaefer, Marcus 111
Schmitt, Daniel 203
Sedgwick, Eric 111
Sember, Jeff 207
Shamos, Michael I. 3
Sheehy, Don 163, 175
Shi, Qiaosheng 119
Shu, Chang 55
Sinha Mahapatra, Priya Ranjan 103
Skiena, Steven 135
Sleator, Danny 163
Smid, Michiel 7
Souvaine, Diane 15, 223
Speckmann, Bettina 107
Stefankovic, Daniel 111
Steiger, William 123
Stolpner, Svetlana 23

Türkoğlu, Duru 115
Taslakian, Perouz 99
Tiwary, Hans Raj 171
Toth, Csaba 223
Toussaint, Godfried 99
Tymoczko, Dmitri 5

Uehara, Ryuhei 31, 39
Urrutia, Jorge 75, 195

Wismath, Stephen 23
Woo, Maverick 163
Wuhrer, Stefanie 55
Wulff-Nilsen, Christian 59

Xiao, Henry 199

Zarrabi-Zadeh, Hamid 159
Zhao, Jihui 123