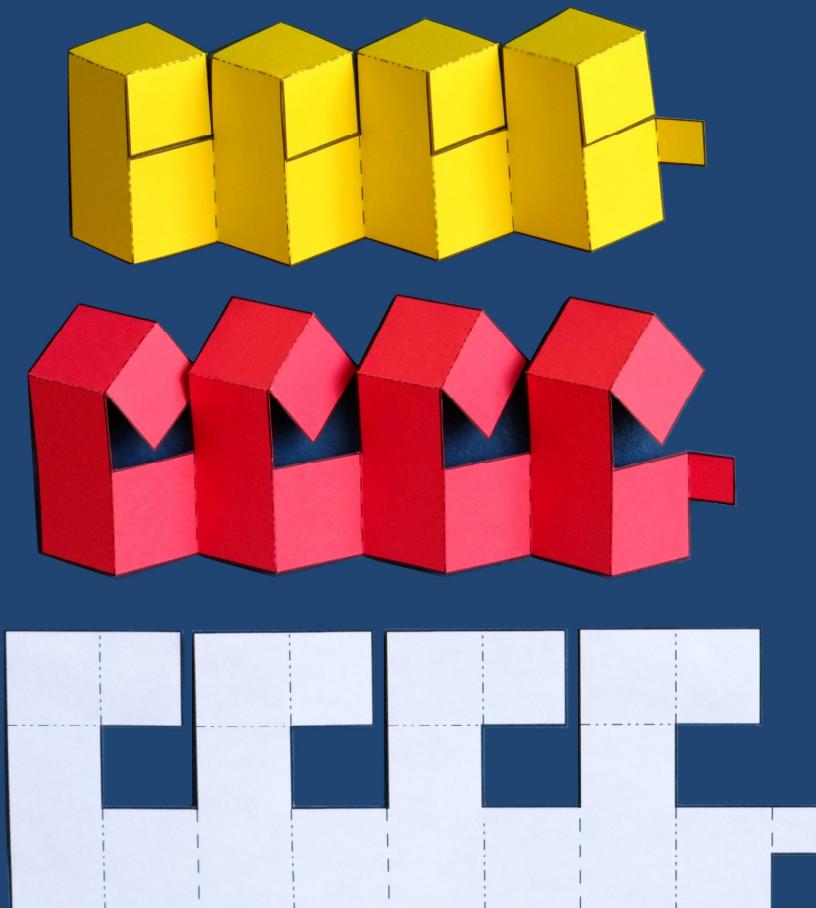


CCCG 2013

The 25th Canadian Conference on Computational Geometry



Waterloo, Ontario, Canada

August 8-10, 2013

Preface

This volume contains the proceedings of the 25th Canadian Conference on Computational Geometry (CCCG'13), held in Waterloo, Ontario on August 8-10, 2013. These papers are also available electronically at <http://www.cccg.ca> and at <http://cs.uwaterloo.ca/conferences/cccg2013>.

We thank Patrick Nicholson and Wendy Rush for preparing the conference site. We also thank Greg Aloupis for sharing his hard earned wisdom in organizing the two previous editions of CCCG, and Sebastian Collette and Peter Palfrader for technical support in preparing these proceedings.

We are grateful to the Program Committee, and external reviewers, for the hard work they did. They thoroughly examined all submissions and provided excellent feedback. Out of 81 papers submitted, 49 are contained in these proceedings, an acceptance rate of 60%. We thank the authors of all submitted papers, all those who have registered, and in particular Alla Sheffer, Sue Whitesides and Peter Widmayer for presenting plenary lectures.

Last but not least, we are grateful for sponsorship from the Fields Institute, the University of Waterloo and the Atlantic Association for Research in the Mathematical Sciences (AARMS). Their financial support has helped us to cover many costs as well as provide significant funding for travel to students, postdocs and invited speakers.

Alejandro López-Ortiz
Therese Biedl
Anna Lubiw
(Conference Organizers)



Copyrights of the articles in these proceedings are maintained by their respective authors. More information about this conference and about previous and future editions is available online at

<http://cccg.ca>

Invited Speakers

Sue Whitesides	University of Victoria
Alla Sheffer	University of British Columbia
Peter Widmayer	ETH Zürich

Program Committee

Peyman Afshani	Aarhus University
Hee-Kap Ahn	Pohang University of Science and Technology
Greg Aloupis	Université Libre de Bruxelles
Boaz Ben-Moshe	Ariel University
Mark de Berg	TU Eindhoven
Therese Biedl	University of Waterloo
Gruia Calinescu	Illinois Institute of Technology
Jean Cardinal	Université Libre de Bruxelles
Paz Carmi	Ben-Gurion University of the Negev
Timothy M. Chan	University of Waterloo
Mirela Damian	Villanova University
Vida Dujmović	McGill University
Adrian Dumitrescu	University of Wisconsin-Milwaukee
Sándor Fekete	TU Braunschweig
Akitoshi Kawamura	University of Tokyo
Alejandro López-Ortiz, Chair	University of Waterloo
Anna Lubiw	University of Waterloo
Anil Maheshwari	Carleton University
Lata Narayanan	Concordia University
Belen Palop	Universidad Valladolid
Michiel Smid	Carleton University
Bettina Speckmann	TU Eindhoven
Csaba Tóth	University of Calgary and CSUN
Ryuhei Uehara	Japan Advanced Institute of Science and Technology
Emo Welzl	ETH Zürich
Norbert Zeh	Dalhousie University

Organizing Committee

Therese Biedl, co-chair
Timothy Chan
Francisco Claude
Hella Hoffmann
Shahin Kamali
Alejandro López-Ortiz, co-chair
Anna Lubiw, co-chair
Daniela Maftuleac
Patrick Nicholson
Venkatesh Raman
Alejandro Salinger
Hamideh Vosoughpour
Gelin Zhou

Additional Reviewers

Luis Barba
Ahmad Biniaz
Ke Chen
Jean-Lou De Carufel
Erik D. Demaine
Martin Demaine
Matt Duckham
William Evans
Eli Fox-Epstein
Bernd Gärtner
Drik H.P. Gerrits
Anirban Ghosh
Arther van Goethem
Michael Hoffmann
Stefan Huber
Yoonho Hwang
Minghui Jiang
Pegah Kamousi
Matya Katz
Stephen Kiazyk
Sang-Sub Kim
Matias Korman
Marc van Kreveld
Vincent Kusters
Arthur Langerman
Wouter Meulemans
Pat Morin
Eunjin Oh
Dongwoo Park
Val Pinciu
Marcel Roeloffzen
Maria Saumell
Wanbin Son
Hans Raj Tiwary
Hamideh Vosoughpour
Sang Duk Yoon

Contents

Plenary talk - Thursday Aug. 8 9:00-10:00

Kinetic Data Structures on The Move

Sue Whitesides

11

Session 1A - Thursday Aug. 8 10:00-10:50

Weighted Straight Skeletons In The Plane

Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser and Peter Palfrader

13

Medial Residues of Piecewise Linear Manifolds

Erin Chambers, Tao Ju and David Letscher

19

Session 1B - Thursday Aug. 8 10:00-10:50

Morpion Solitaire 5D: a new upper bound 121 on the maximum score

Akitoshi Kawamura, Takuma Okamoto, Yuichi Tatsu, Yushi Uno and Masahide Yamato

25

Computational complexity and an integer programming model of Shakashaka

Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara and Yushi Uno

31

Session 2A - Thursday Aug. 8 11:15-12:30

One-Round Discrete Voronoi Game in \mathbb{R}^2 in Presence of Existing Facilities

Aritra Banik, Bhaswar Bhattacharya, Sandip Das and Satyaki Mukherjee

37

Zipper Unfoldability of Domes and Prismoids

Erik D. Demaine, Martin L. Demaine and Ryuhei Uehara

43

Map Folding

Rahnuma Islam Nishat and Sue Whitesides

49

Session 2B - Thursday Aug. 8 11:15-12:30

Partial Searchlight Scheduling is Strongly PSPACE-complete

Giovanni Viglietta

55

Set-Difference Range Queries

David Eppstein, Michael Goodrich and Joseph A. Simons

61

The Unified Segment Tree and its Application to the Rectangle Intersection Problem

David P. Wagner

67

Session 3A - Thursday Aug. 8 13:50-15:05**Covering Folded Shapes**

Oswin Aichholzer, Greg Aloupis, Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Michael Hoffmann, Anna Lubiw, Jack Snoeyink and Andrew Winslow **73**

Unfolding Face-Neighborhood Convex Patches: Counterexamples and Positive Results

Joseph O'Rourke **79**

Counting Triangulations Approximately

Victor Alvarez, Karl Bringmann, Saurabh Ray and Raimund Seidel **85**

Session 3B - Thursday Aug. 8 13:50-15:05**Aggregate-Max Nearest Neighbor Searching in the Plane**

Haitao Wang **91**

Data structures for incremental extreme ray enumeration algorithms

Blagoy Genov **97**

Fault Tolerant Clustering Revisited

Nirman Kumar and Benjamin Raichel **103**

Open Problems Session**Open Problems from CCCG 2012**

Joseph S. B. Mitchell **109**

Plenary talk - Friday Aug. 9 9:00-10:00**Quantifying Design: the geometric properties behind designer choices and human perception**

Alla Shaffer **115**

Session 4A - Friday Aug. 9 10:00-10:50**Universal Point Sets for Planar Graph Drawings with Circular Arcs**

Patrizio Angelini, David Eppstein, Fabrizio Frati, Michael Kaufmann, Sylvain Lazard, Tamara Mchedlidze, Monique Teillaud and Alexander Wolff **117**

Weighted Region Problem in Arrangement of Lines

Amin Gheibi, Anil Maheshwari and Jörg-Rüdiger Sack **123**

Session 4B - Friday Aug. 9 10:00-10:50**Combinatorics of Beacon Routing and Coverage**

Michael Biro, Jie Gao, Justin Iwerks, Irina Kostitsyna and Joseph S. B. Mitchell **129**

Privacy by Fake Data: A Geometric Approach

Victor Alvarez, Erin Chambers and László Kozma **135**

Session 5A - Friday Aug. 9 11:20-12:35**Stabbing Polygonal Chains with Rays is Hard to Approximate***Steven Chaplick, Elad Cohen and Gila Morgenstern*

141

Heaviest Induced Ancestors and Longest Common Substrings*Travis Gagie, Paweł Gawrychowski and Yakov Nekrich*

145

Maximum-Weight Planar Boxes in $O(n^2)$ Time (and Better)*Jérémie Barbay, Timothy M. Chan, Gonzalo Navarro and Pablo Pérez-Lantero*

151

Session 5B - Friday Aug. 9 11:20-12:35**An Optimal Algorithm Computing Edge-to-Edge Visibility in a Simple Polygon***Mikkel Abrahamsen*

157

Counting Carambolas*Maarten Löffler, André Schulz and Csaba Tóth*

163

Cell-Paths in Mono- and Bichromatic Line Arrangements in the Plane*Oswin Aichholzer, Jean Cardinal, Thomas Hackl, Ferran Hurtado, Matias Korman, Alexander Pilz, Rodrigo Silveira, Ryuhei Uehara, Birgit Vogtenhuber and Emo Welzl*

169

Session 6A - Friday Aug. 9 14:05-15:45**Optimal Data Structures for Farthest-Point Queries in Cactus Networks***Prosenjit Bose, Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari and Michiel Smid*

175

Cole's Parametric Search Technique Made Practical*Michael Goodrich and Paweł Pszona*

181

Polynomial Time Algorithms for Label Size Maximization on Rotating Maps*Yusuke Yokosuka and Keiko Imai*

187

Drawing some 4-regular planar graphs with integer edge lengths*Timothy Sun*

193

Session 6B - Friday Aug. 9 14:05-15:45**Hyperbanana Graphs***Christopher Clement, Audrey Lee-St.John and Jessica Sidman*

199

Theta-3 is connected*Oswin Aichholzer, Sang Won Bae, Luis Barba, Prosenjit Bose, Matias Korman, André van Renssen, Perouz Taslakian and Sander Verdonschot*

205

How to Cover Most of a Point Set with a V-Shape of Minimum Width*Boris Aronov, John Iacono, Özgür Özkan and Mark Yagnatinsky*

211

Computing Covers of Plane Forests*Luis Barba, Alexis Beingessner, Prosenjit Bose and Michiel Smid*

217

Session 7A - Friday Aug. 9 16:15-17:55**An Efficient Exact Algorithm for the Natural Wireless Localization Problem***Bruno Crepaldi, Pedro de Rezende and Cid de Souza*

223

On k-Enclosing Objects in a Coloured Point Set*Luis Barba, Stephane Durocher, Robert Fraser, Ferran Hurtado, Saeed Mehrabi, Debajyoti Mondal, Jason Morrison, Matthew Skala and Mohammad Abdul Wahid*

229

On the Rectangle Escape Problem*Sepehr Assadi, Ehsan Emamjomeh-Zadeh, Sadra Yazdanbod and Hamid Zarrabi-Zadeh*

235

Grid Proximity Graphs: LOGs, GIGs and GIRLs*River Allen, Laurie Heyer, Rahnuma Islam Nishat and Sue Whitesides*

241

Session 7B - Friday Aug. 9 16:15-17:55**Bounding the Locus of the Center of Mass for a Part with Shape Variation***Fatemeh Panahi and A. Frank van der Stappen*

247

Geometric Separators and the Parabolic Lift*Donald R. Sheehy*

253

How To Place a Point to Maximize Angles*Boris Aronov and Mark Yagnatinsky*

259

Spanning Colored Points with Intervals*Payam Khanteimouri, Ali Mohades, Mohammad Ali Abam and Mohammad Reza Kazemi*

265

Session 8A - Saturday Aug. 10 9:00-10:40**On Fence Patrolling by Mobile Agents***Ke Chen, Adrian Dumitrescu and Anirban Ghosh*

271

Face-Guarding Polyhedra*Giovanni Viglietta*

277

On k -Guarding Polygons*Daniel Busto, William Evans and David Kirkpatrick*

283

Geometric Red-Blue Set Cover for Unit Squares and Related Problems*Timothy M. Chan and Nan Hu*

289

Session 8B - Saturday Aug. 10 9:00-10:15**Convex hull alignment through translation***Michael Hoffmann, Vincent Kusters, Günter Rote, Maria Saumell and Rodrigo I. Silveira*

295

Faster approximation for Symmetric Min-Power Broadcast*Gruia Calinescu*

301

Planar Convex Hull Range Query and Related Problems*Nadeem Moidu, Jatin Agarwal and Kishore Kothapalli*

307

Plenary talk - Saturday Aug. 10 11:00-12:00**What's in a gaze? How to reconstruct a simple polygon from local snapshots***Peter Widmeyer*

311

Kinetic Data Structures on The Move

Sue Whitesides

University of Victoria, British Columbia, Canada

From its beginnings, computational geometry has studied problems concerning sets of points in the plane: which pair is closest? For each point, what is the closest neighbor? What are good computational methods for computing structures such as convex hulls, Voronoi diagrams, minimum spanning trees? When the points are moving, answering such questions presents new challenges. The area of kinetic data structures emerged in the late 1990's. Given today's ubiquity of smart mobile devices, problems on moving points and objects are all the more relevant to applications. This talk reviews the topic, including some recent results.

Weighted Straight Skeletons In The Plane

Therese Biedl*

Martin Held†

Stefan Huber‡

Dominik Kaaser†

Peter Palfrader†

Abstract

In this paper, we investigate the weighted straight skeleton from a geometric, graph-theoretical and combinatorial point of view. We start with a thorough definition, shed light on an ambiguity issue in the procedural definition, and propose solutions. We investigate the geometry of faces and the roof model and we discuss in which cases the straight skeleton is connected. Finally, we show that the weighted straight skeleton of even a simple polygon may be non-planar and may contain cycles, and we discuss under which circumstances the weighted straight skeleton still behaves similar to its unweighted pendant.

1 Introduction

The straight-skeleton $\mathcal{S}(P)$ of a simple polygon P is a skeleton structure that was introduced by Aichholzer et al. [1] to computational geometry about 20 years ago.¹ Its definition is based on a wavefront propagation process where the polygons edges move inwards with unit speed. The straight skeleton, roughly speaking, is the skeleton structure that results from the interference patterns of the wavefront edges. Aichholzer and Aurenhammer [2] later generalized the definition to planar straight-line graphs. Since their introduction a lot of applications appeared in different research areas and multiple algorithms to compute the straight skeleton are known [8].

Eppstein and Erickson [6] were the first to mention the *weighted straight skeleton* where the wavefront edges may move with arbitrary but fixed speed. They claim that their algorithm to compute the unweighted straight skeleton in $O(n^{8/5+\epsilon})$ time and space also works, without major changes, for weighted straight skeletons. Weighted straight skeletons have many applications: Barequet et al. [5] use weighted straight skeletons in order to define the initial wavefront topology for straight skeletons of polyhedra. Haunert and Sester [7] use the

weighted straight skeleton for topology-preserving area collapsing in geographic maps. Laycock and Day [11] and Kelly and Wonka [10] use weighted straight skeletons to model realistic roofs of houses. Aurenhammer [4] investigated fixed-share decompositions of convex polygons using weighted straight skeletons with specific positive weights.

Although algorithms, applications and even simple implementations [9] of weighted straight skeletons were published, only limited research was conducted on the weighted straight skeleton per se. The only known results are that the simple definition based on wavefront propagation may lead to ambiguities [10, 8] and that the lower envelope characterization by Eppstein and Erickson [6] does not apply. In this paper, we carefully define weighted straight skeletons, shed light on the ambiguity in the procedural definition, investigate geometric, graph-theoretical and combinatorial properties of weighted straight skeletons, and compare those with properties of unweighted straight skeletons. In particular, we show that weighted straight skeletons of simple polygons may have cycles and crossings. Furthermore, we investigate necessary conditions for the weights such that the straight skeleton of a simple polygon is a planar tree.

2 Preliminaries

The definition of the straight skeleton $\mathcal{S}(P)$ of a simple polygon P is based on a so-called wavefront propagation of P where all edges of P move inwards in parallel and with unit speed. The wavefront, denoted by $\mathcal{W}_P(t)$, for small t has the shape of a mitered offset-curve of P . As t increases, \mathcal{W}_P changes its topology. Such a change is called an event: An *edge event* happens if an edge e collapsed to zero length and vanishes. A *split event* happens when a reflex wavefront vertex reaches a wavefront edge e and splits the edge into parts. Either event causes local changes in the topology of the wavefront.

The straight skeleton $\mathcal{S}(P)$ is defined by set of loci traced out by the vertices of $\mathcal{W}_P(t)$ for all $t \geq 0$, see Fig. 1. Additionally, some loci are added to the straight skeleton in case of parallel edges as follows: (a) If two parallel edge e, e' that move in opposite direction become overlapping during a split event, then the region common to e and e' is added to the straight skeleton, while the region(s) that belongs to exactly one of them remains in the wavefront. (b) If two parallel edges e, e'

*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 1A2, Canada. Supported by NSERC. Research was done while the author was visiting Universität Salzburg. biedl@uwaterloo.ca

†Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria. [\[held,dominik,ppalfrad\]@cosy.sbg.ac.at](mailto:[held,dominik,ppalfrad]@cosy.sbg.ac.at)

‡Institute of Science and Technology Austria, 3400 Klosterneuburg, Austria. stefan.huber@ist.ac.at

¹Remarks on the history are given in [3].

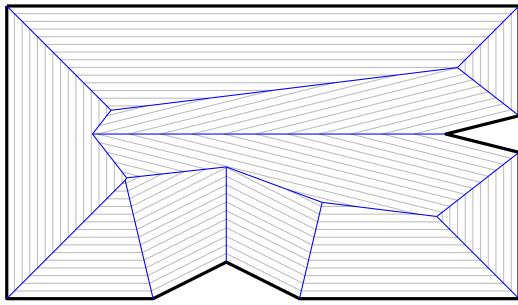


Figure 1: The straight skeleton $\mathcal{S}(P)$ (blue) of the input polygon P (bold) is defined by wavefronts (grey) emanated by P .

that move in the same direction become adjacent during an edge event, then their common endpoint is considered a vertex of the wavefront. We call this a *ghost vertex*, which moves perpendicular to e and e' .²

Each wavefront vertex traces out an *arc* of $\mathcal{S}(P)$. As wavefront vertices move along bisectors of edges of P , the arcs of $\mathcal{S}(P)$ are straight-line segments. Every event of \mathcal{W}_P belongs to a locus where arcs of $\mathcal{S}(P)$ meet and give rise to a *node* of $\mathcal{S}(P)$. See Figure 1 for an example. The straight skeleton $\mathcal{S}(P)$ is interpreted as a graph, and one can show that it is a tree [1]. Also, no two arcs of the straight skeleton cross since the wavefront moves inwards towards the unswept region. Hence $\mathcal{S}(P) \cup P$ is a planar straight-line graph. The inner faces of $\mathcal{S}(P) \cup P$ are called *straight-skeleton faces*.

For a polygon edge e , let the *wavefront fragments* of e at time t , denoted $e(t)$, be the union of segments of $\mathcal{W}_P(t)$ that originated from e ; set $e(t)$ may comprise none, one, or many segments depending on whether e participated in edge events and/or split events. We consider segments in $e(t)$ to be *open* line segments. Define $f(e) := \bigcup_{t>0} e(t)$ to be the *face* of edge e ; this is the region that was swept by e during the propagation. One can easily show that the faces of wavefront-edges are in 1-1-correspondence with the straight-skeleton faces. Also, $f(e)$ is monotone with respect to the line through e [1, 2], and its lower chain is convex [8]. Furthermore, the boundary of each face $f(e)$ corresponds to a cycle in $P \cup \mathcal{S}(P)$.³

2.1 Roof models

Aichholzer et al. [1] introduced the *roof model*, which is a handy way of interpreting the straight skeleton.

²One could also argue for omitting ghost vertices, hence effectively merging e and e' into one edge of the wavefront. But then, even in the unweighted case, the straight skeleton is not always connected.

³Aichholzer and Aurenhammer [2] extended the definition of straight skeletons to planar straight line graphs. Then some arcs are rays to infinity. We consider such arcs to meet at a node located at infinity. Thus, the property even holds in this case.

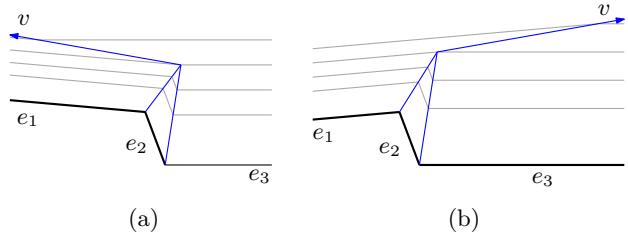


Figure 2: The definition of straight skeletons is ambiguous when two parallel wavefront edges with different weights become adjacent.

One considers the wavefront propagation embedded in three-space where the z -axis constitutes time. Then \mathcal{W}_P traces out a surface, namely the roof model $\mathcal{T}(P) := \bigcup_{t>0} \mathcal{W}_P(t) \times \{t\}$. Note that we can obtain $\mathcal{S}(P)$ from $\mathcal{T}(P)$ by projecting the edges of $\mathcal{T}(P)$ onto the plane $\mathbb{R}^2 \times \{0\}$. Vice versa, we can obtain $\mathcal{T}(P)$ from $\mathcal{S}(P)$ by lifting all nodes of $\mathcal{S}(P)$ by their orthogonal distance to the respective input edges of P (i.e., the time when they were swept by the wavefront). In other words, $\mathcal{T}(P)$ gives us the means to investigate \mathcal{W}_P over its entire lifespan.

The roof model is sometimes also called *terrain model*, since $\mathcal{T}(P)$ is a *terrain*, i.e., any line parallel to the z -axis intersects it in at most one point. As this property may be violated for the weighted version of straight skeletons, we prefer the term “roof model”.

2.2 Weighted straight skeletons

The weighted straight skeleton differs from the straight skeleton only in the speed $\sigma(e) \in \mathbb{R} \setminus \{0\}$ with which edge e moves in the wavefront. We call σ the *weight function* and $\sigma(e)$ the *weight* of e . The wavefront moves such that the fragments $e(t)$ of e at time t are on the line $\bar{e} + \sigma(e) \cdot n(e)$, where \bar{e} is the line through e , and $n(e)$ is the inward normal of e . We note that $\sigma(e)$ is not necessarily positive; for $\sigma(e) < 0$ edge e moves outward with $|\sigma(e)|$.⁴

All other definitions that we gave for (unweighted) straight skeletons, such as edge event, split event, $\mathcal{W}_P(t)$, $\mathcal{S}(P)$, $e(t)$, $f(e)$, $\mathcal{T}(P)$ carry over verbatim to the weighted straight skeleton. (We use an additional parameter σ in the names of these objects when we want to emphasize that we are discussing the weighted version.) The only exception to the “carrying over verbatim” is the change of topology during an edge event;

⁴One could expand the definition to zero weights, resulting in stationary wavefront edges. The roof model is then still well-defined, and the straight skeleton could be considered to be the projection of the edge graph of the roof model onto the ground plane. However, straight skeleton arcs may then degenerate into points and faces of edges may be a line segment. We forbid zero-weight edges here to avoid such degeneracies.

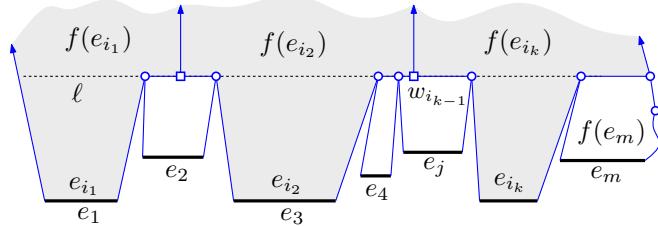


Figure 3: Multiple parallel wavefront edges become adjacent when simultaneously reaching the line ℓ .

here there exists one ambiguity that we discuss in detail in Section 3.

While the definitions carry over, it is not at all clear which of the properties of the straight skeleton (such as planarity, tree, $f(e)$ is monotone, $f(e)$ corresponds to faces) carries over to the weighted version. This is the main topic of the paper, and will be discussed in Section 4.

3 Ambiguity of definition

Presume that we have an edge event where edge e_2 disappears, leaving its adjacent edges e_1 and e_3 to become adjacent with common endpoint v . See Fig. 2. Some elementary computation shows that if γ is the angle spanned by e_1 and e_3 (on the side that the wavefront propagates to), and α_1 is the angle between e_1 and the arc traced by v , then

$$\cot \alpha_1 = \frac{\cos \gamma + \frac{\sigma(e_3)}{\sigma(e_1)}}{\sin \gamma}. \quad (1)$$

However, this equation only holds for $\sin(\gamma) \neq 0$, i.e., if e_1 and e_3 are not parallel. Worse, as γ monotonically approaches π , we may obtain two different limit cases:

$$\lim_{\gamma \nearrow \pi} \alpha_1 = 0 \quad \lim_{\gamma \searrow \pi} \alpha_1 = \pi, \quad (2)$$

see Fig. 2. Hence, as already alluded to by Kelly and Wonka [10] and Huber [8], the definition of the weighted straight skeleton is ambiguous whenever parallel wavefront edges with different weights are adjacent. In case $\sigma(e_1) = \sigma(e_3)$ and $\gamma = \pi$, we obtain $\lim_{\gamma \rightarrow \pi} \cot \alpha_1 = 0$ by de l'Hôpital's rule, and hence $\alpha_1 = \pi/2$. Similarly $\alpha_1 = \pi/2$ if $\sigma(e_1) = -\sigma(e_3)$ and $\gamma = 0$. But in all other cases, there is no unique definition of a straight skeleton. Note that the situation can become even more complicated if multiple parallel wavefront edges with different weights become consecutive, as in Fig. 3.

We see a few canonical ways to resolve this situation:

- The wavefront edge(s) with maximum speed dominate all involved events.

- The wavefront edge(s) with maximum absolute speed dominate all involved events. In case of a tie (e.g. one edge has speed $+1$, another one has speed -1 , and all others have speed in $[-1, +1]$) the inward-moving edge(s) win.
- The wavefront edge(s) with maximum absolute speed dominate all involved events. In case of a tie the outward-moving edge(s) win.
- Three more options are as above with “maximum” replaced by “minimum”.

We will describe the first resolution in more detail, the others are similar. Assume (as in Fig. 3) that at some time line ℓ contains part of the wavefront; say a maximal contiguous part of the wavefront on ℓ consists of vertices and edges $v_0, e_1, v_1, \dots, e_m, v_m$ (in order). Also assume for now that e_1, \dots, e_m all approach ℓ from the same half-plane. Let e_{i_1}, \dots, e_{i_k} be those edges among them that maximize $\sigma(e_{i_j})$. We create, for $j = 1, \dots, k-1$, a ghost vertex w_{i_j} halfway between v_{i_j} and $v_{i_{j+1}-1}$. Then replace the wavefront between v_0 and v_m by edges $(v_0, w_{i_1}), (w_{i_1}, w_{i_2}), \dots, (w_{i_{k-2}}, w_{i_{k-1}}), (w_{i_{k-1}}, v_m)$, i.e., edges e_{i_1}, \dots, e_{i_k} “take over” all other edges in this part of the wavefront. Edges e_{i_1}, \dots, e_{i_k} continue to propagate (they all had the same speed), while all other edges in e_1, \dots, e_m disappear.

If not all edges on ℓ approach from the same half-plane, then any non-empty line segment that is common to two edges in opposite direction becomes an arc of the weighted straight skeleton. We otherwise proceed as above: the edge(s) with the maximum speed “take over” all other adjacent edges of the wavefront that reside on ℓ .

4 Properties of the straight skeletons

In what follows, we will assume there never are two parallel edges of different weights that become consecutive during an edge event. Under this assumption the weighted straight skeleton is uniquely defined. But as we show now, even then many seemingly natural properties do not hold. Table 1 lists all results.

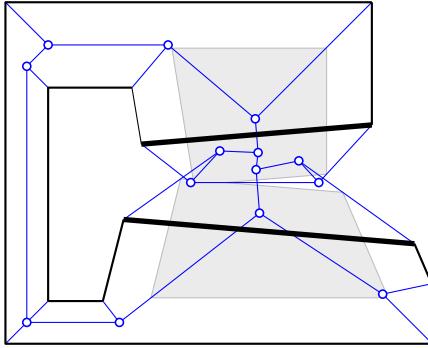
Lemma 1 *There exists a simple polygon P with weights chosen from $\{+1, -1\}$, such that $S(P, \sigma)$ has crossings and cycles.*

Proof. The polygon is shown in Fig. 4. \square

4.1 Terrains and crossings

We now want to show that if the weights are positive, then the straight skeleton has no crossing. For this (and other claims later) it will help to study when the roof model $\mathcal{T}(P, \sigma)$ is a terrain.

Property	Simple polygon			Polygon with holes		
	$\sigma \equiv 1$	σ positive	σ arbitrary	$\sigma \equiv 1$	σ positive	σ arbitrary
$\mathcal{S}(P)$ is connected	✓ [1]	✓ (Lem. 12)	✓ (Lem. 11)	✓ (Lem. 12)	✓ (Lem. 12)	✗ (Lem. 9)
$\mathcal{S}(P)$ has no crossing	✓ [1]	✓ (Lem. 4)	✗ (Lem. 1)	✓ (Lem. 4)	✓ (Lem. 4)	✗ (Lem. 1)
$f(e)$ is monotone w.r.t. e	✓ [1]	✗ (Lem. 8)	✗ (Lem. 8)	✓ (as in [1])	✗ (Lem. 8)	✗ (Lem. 8)
$\text{bd } f(e)$ is a simple polygon	✓ [1]	✓ (Lem. 6)	✗ (Lem. 7)	✓ (as in [1])	✗ (Lem. 5)	✗ (Lem. 5)
$\mathcal{T}(P, \sigma)$ is z -monotone	✓ [1]	✓ (Lem. 2)	✗ (Lem. 3)	✓ (Lem. 2)	✓ (Lem. 2)	✗ (Lem. 3)
$\mathcal{S}(P)$ has $n(\mathcal{S}(P)) - 1 + h$ arcs	✓ [1]	✓ (Lem. 13)	✗ (Lem. 1)	✓ (Lem. 13)	✓ (Lem. 13)	✗ (Lem. 1)

Table 1: Results for a simple polygon and a polygon with h holes.Figure 4: $\mathcal{S}(P, \sigma)$ of a simple polygon P may have crossings and cycles. A wavefront is shown in grey. All edges have weight $+1$, except the two bold edges, which have weight -1 .

Lemma 2 Let P be a polygon (possibly with holes). If $\sigma(e) > 0$ for all wavefront edges e , then the roof model $\mathcal{T}(P, \sigma)$ is a terrain and its z -projection equals P .

Proof. Because the weights are all positive, the wavefront edges emanated by P move towards the interior of P . After each event of $\mathcal{W}_{P, \sigma}$ the trajectories of the newly born wavefront vertices hence point to the area within P not yet swept by the wavefront. Hence, no wavefront vertex can ever reach a locus that has already been swept as this vertex would have met another wavefront edge before that and the vertex would have been annihilated. Therefore the wavefront $\mathcal{W}_{P, \sigma}$ stays within P and no locus of P is swept more than once by the wavefront. On the other hand, each locus of P is swept at least once, since otherwise the boundary of the unswept region would be the wavefront, and hence not empty yet. \square

Lemma 3 There exists a simple polygon such that if all edges are assigned weights in $\{+1, -1\}$, $\mathcal{T}(P, \sigma)$ is not a terrain.

Proof. The example shown in Fig. 4 contains loci that are swept more than once and hence $\mathcal{T}(P, \sigma)$ is not a terrain. In fact, it can be easily extended such that some loci are swept an arbitrary number of times. \square

Lemma 4 Let P be a polygon (possibly with holes). If $\sigma(e) > 0$ for all wavefront edges e , then $\mathcal{S}(P, \sigma)$ has no crossings.

Proof. This holds by Lemma 2, since the locus p of any crossing must have been covered at least twice by the wavefront. But then the line parallel to the z -axis through p would intersect $\mathcal{T}(P, \sigma)$ twice. \square

4.2 Faces of edges

We later want to argue that under some assumptions the straight skeleton is connected. To do so, we first study some properties of the faces of edges. Recall that $f(e) = \bigcup_{t>0} e(t)$, where $e(t)$ are the open line segments that result from edge e at time t . Clearly $f(e)$ is connected (no fragment of e suddenly appears during a wavefront process) and its boundary $\text{bd } f(e)$ consists of arcs of the straight skeleton.

Lemma 5 There exists a polygon P with holes, with weights chosen from $\{1, 3\}$, such $\text{bd } f(e)$ is not a simple polygon.

Proof. Polygon P is shown in Fig. 5 (include the dotted features), with edge e the bottommost horizontal edge. During the wavefront process, edge e gets split when it meets the hole. But since e moves faster than the edges of the hole, two fragments of e later re-combine. According to our definition of straight skeleton, a ghost vertex is created that traces the arc a in Fig. 5. The boundary of $f(e)$, viewed as a polygon, contains arc a twice and is not simple. \square

Lemma 5 used a polygon with holes. Whether for simple polygons a similar situation can arise depends on whether the weights are positive or not.

Lemma 6 Let P be a simple polygon and σ be an assignment of positive weights to the edges of P . Then $\text{bd } f(e)$ is a simple polygon for all wavefront-edges e .

Proof. (Sketch) Note that $f(e)$ is an open, simply-connected set: whenever a fragment splits, the pieces never re-merge due to the ghost-vertices. Hence $\text{bd } f(e)$

is a weakly simple polygon, and the only way that it could be not simple is by having a point p that is incident to three or more edges of the boundary of $f(e)$, see Fig. 5. This can only happen if two fragments s_1, s_2 in $e(t)$ become adjacent in the wavefront. By tracing from s_1 and s_2 back to e while residing inside $f(e)$, we can find a closed Jordan curve C inside $\text{cl } f(e)$ that encloses a point outside $\text{cl } f(e)$.

For positive weights the roof model $\mathcal{T}(P)$ projects to P (Lemma 2), so $C \subset \text{cl } f(e) \subset P$. Since P is simple, any point inside C also belongs to P . Since some point inside C does not belong to $\text{cl } f(e)$, therefore some other face $f(e')$ is inside C . But by planarity (Lemma 4) then the edge e' of this face is inside C as well. Since C contains no edges of P , therefore edge e' cannot be connected to the edges at the exterior face of P along edges of P . So P has a hole, a contradiction. \square

Lemma 7 *There exists a simple polygon P with weights chosen from $\{-1, +1, +3\}$ such that $\text{bd } f(e)$ is not a simple polygon for an edge e of P .*

Proof. The polygon is shown in Fig. 6. \square

If the polygon is simple and all weights are the same, then $\text{bd } f(e)$ is simple, because $f(e)$ is monotone with respect to the line through e [1]. We note here that $f(e)$ need not be monotone if we allow weights. This is already obvious from Fig. 5 and 6, but may happen even for simple polygons and positive weights.

Lemma 8 *There exists a simple polygon P with weights chosen from $\{1, 3\}$ such that for one wavefront edge e face $f(e)$ is not monotone with respect to the line through e .*

Proof. The face $f(e')$ in Fig. 5 (omit the dotted features) is not monotone. \square

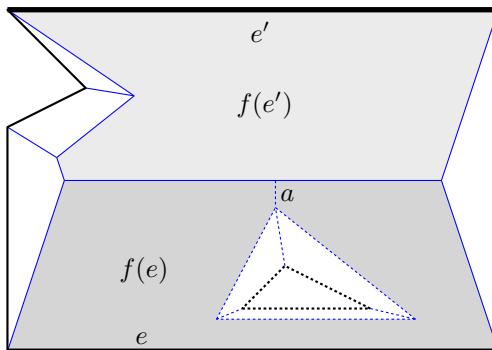


Figure 5: A polygon with hole may have a non-simple face. A simple polygon may have a non-monotone face. The dotted features are caused by the hole. The bold edges have weight 3, the others have weight 1.

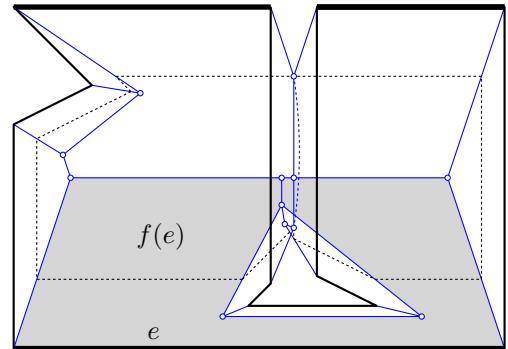


Figure 6: A simple polygon where face $f(e)$ (shaded) has a non-simple boundary. One wavefront is depicted by dotted lines. The bold edges have weight 3 and the two vertical edges that form the corridor have weight -1 . All other edges have weight 1. The arc between the two corridor edges geometrically coincides with other arcs.

4.3 Connectivity

The unweighted straight skeleton $\mathcal{S}(P)$ of a polygon P with holes is always connected. In fact, $\mathcal{S}(P)$ is even of the same homotopy type as P . The weighted straight skeleton, however, need not even be connected.

Lemma 9 *There exists a polygon P with holes such that $\mathcal{S}(P, \sigma)$ is not connected.*

Proof. If all weights are negative then $\mathcal{S}(P, \sigma)$ resides in each component of $\mathbb{R}^2 \setminus P$. \square

The following lemma serves as a tool to prove connectedness in the following. The lemma basically says that straight-skeleton features that are connected via the wavefront at any time are also connected within the final straight skeleton.

Lemma 10 *Let $\mathcal{S}_t(P, \sigma)$ denote the straight-skeleton features traced by \mathcal{W}_P until time t . If two points $p, q \in \mathcal{S}_t(P, \sigma)$ are path-connected on $\mathcal{S}_t(P, \sigma) \cup \mathcal{W}_{P, \sigma}(t)$ then they are path-connected on $\mathcal{S}(P, \sigma)$.*

Proof. We prove this lemma by induction on the events of \mathcal{W}_P in chronological order. We need to check that connectivity is maintained despite the changes of \mathcal{W}_P caused by an event. An event may simply remove a collapsed edge of $\mathcal{W}_P(t)$ (edge event), remove a collapsed component of $\mathcal{W}_P(t)$ (edge event), split a component of $\mathcal{W}_P(t)$ (split event), or merge components (split event). In any case a straight-skeleton node v is created to which arcs are incident that were traced by the vertices of each involved component of the wavefront. However, even if the wavefront is split into multiple components, each component remains connected to v by at least one arc that is traced by a vertex of each component. That is,

an event never disconnects a component from the node that is created by the event.⁵ \square

Lemma 11 $\mathcal{S}(P, \sigma)$ is connected for simple polygons P .

Proof. This holds even for negative weights as the initial wavefront of P consists of only one connected component. (Recall that infinite arcs are incident to a node at infinity.) \square

Lemma 12 $\mathcal{S}(P, \sigma)$ is connected for polygons with holes P and positive weights.

Proof. Consider the wavefront emanating from some hole. At some point, it either merges with some other wavefront during a split event; then the claim holds by induction (and Lemma 11) since $\mathcal{W}_{P, \sigma}$ then has fewer components. Or it collapses during an edge event. But this is impossible, since for positive weights the wavefront of the hole moves towards the inside of P and hence encloses ever more area. \square

4.4 Bounds on the edges

It is well-known that for simple polygons, the straight skeleton is a tree. It is not hard to verify that for positive weights, also the weighted straight skeleton is a tree. We show an even stronger statement, which bounds the number of edges even in the presence of holes.

Lemma 13 Let P be a polygon with h holes, and let σ be an assignment of positive weights to the edges of P . Then $\mathcal{S}(P)$ has $n(\mathcal{S}(P)) + h - 1$ arcs, where $n(\mathcal{S}(P))$ is the number of nodes of $\mathcal{S}(P)$ plus the number of vertices of P .

Proof. By Lemma 2, $\mathcal{T}(P, \sigma)$ is a terrain that projects to P . Denote by \mathcal{P} the polyhedron that is enclosed by $\mathcal{T}(P)$ and $\mathcal{T}(P)$ mirrored at the plane $z = 0$. Observe that \mathcal{P} is z -monotone as $\mathcal{T}(P)$ is z -monotone and its z -projection is P . Hence \mathcal{P} has exactly one handle for each hole of P , so the genus of \mathcal{P} is h .

Let $n(P)$ ($n(\mathcal{P})$, resp.) be the number of vertices of P (\mathcal{P} , resp.) and $m(P)$ ($m(\mathcal{P})$, $m(\mathcal{S}(P))$, resp.) be the number of edges/arcs of P (\mathcal{P} , $\mathcal{S}(P)$, resp.). We have $n(\mathcal{P}) = 2n(\mathcal{S}) - n$ and $m(\mathcal{P}) = 2m(\mathcal{S}) + n$. Also note that the number $f(\mathcal{P})$ of faces of \mathcal{P} is $2m(\mathcal{P}) = 2n(P)$, since every edge of P gives rise to one face of the roof model, and hence two faces of \mathcal{P} .

Since $\mathcal{S}(P)$ is connected by Lemma 12, so is the graph of \mathcal{P} and Euler's formula applies. Since \mathcal{P} has genus h , therefore $n(\mathcal{P}) - m(\mathcal{P}) + f(\mathcal{P}) = 2 - 2h$. This implies $2n(\mathcal{S}) - n - 2m(\mathcal{S}) - n + 2n = 2 - 2h$, hence $m(\mathcal{S}) = n(\mathcal{S}) - 1 + h$ as desired. \square

⁵Note that adding ghost vertices when fragments of the same edge re-combine is crucial here, otherwise no arc would emanate from the created skeleton node v .

5 Conclusion

In this paper, we studied properties of weighted straight skeletons, and shows that many seemingly natural properties do not necessarily hold for it, especially if negative weights are allowed. Hence caution must be used when applying weighted straight skeletons. We suspect that many of the applications have a special situation (e.g. convex polygons [4] or weights defined in a special way [5]) that imply that weighted straight skeletons behave “just like” the unweighted ones, but this remains a topic for future study.

References

- [1] O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gärtner. Straight Skeletons of Simple Polygons. In *Proc. 4th Internat. Symp. of LIESMARS*, pages 114–124, Wuhan, P.R. China, 1995.
- [2] O. Aichholzer and F. Aurenhammer. Straight Skeletons for General Polygonal Figures in the Plane. In A. Samoilenco, editor, *Voronoi's Impact on Modern Science, Book 2*, pages 7–21. Institute of Mathematics of the National Academy of Sciences of Ukraine, Kiev, Ukraine, 1998.
- [3] O. Aichholzer, H. Cheng, S. Devadoss, T. Hackl, S. Huber, B. Li, and A. Risteski. What Makes a Tree a Straight Skeleton? In *Proc. 24th Canad. Conf. Comput. Geom. (CCCG'12)*, pages 267–272, Charlottetown, P.E.I., Canada, Aug. 2012.
- [4] F. Aurenhammer. Weighted Skeletons and Fixed-Share Decomposition. *Comput. Geom. Theory and Appl.*, 40(2):93–101, July 2008.
- [5] G. Barequet, D. Eppstein, M. T. Goodrich, and A. Vaxman. Straight Skeletons of Three-Dimensional Polyhedra. In *Proc. 16th Annu. Europ. Symp. Algorithms*, pages 148–160, Karlsruhe, Germany, Sept. 2008.
- [6] D. Eppstein and J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete Comput. Geom.*, 22(4):569–592, 1999.
- [7] J.-H. Haunert and M. Sester. Area Collapse and Road Centerlines Based on Straight Skeletons. *GeoInformatica*, 12:169–191, 2008.
- [8] S. Huber. *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*. Shaker Verlag, Apr. 2012. ISBN 978-3-8440-0938-5.
- [9] T. Kelly. <http://code.google.com/p/campskeleton/>.
- [10] T. Kelly and P. Wonka. Interactive Architectural Modeling with Procedural Extrusions. *ACM Trans. Graph.*, 30(2):14:1–14:15, Apr. 2011.
- [11] R. Laycock and A. Day. Automatically Generating Large Urban Environments Based on the Footprint Data of Buildings. In *Proc. 8th Symp. Solid Modeling Applications*, pages 346–351, Seattle, WA, USA, June 2003.

Medial Residues of Piecewise Linear Manifolds

Erin W. Chambers*

Tao Ju†

David Letscher‡

Abstract

Skeleton structures of objects are used in a wide variety of applications such as shape analysis and path planning. One of the most widely used skeletons is the medial axis, which is a thin structure centered within and homotopy equivalent to the object. However, on piecewise linear surfaces, which are one of the most common outputs from surface reconstruction algorithms, natural generalizations of typical medial axis definitions may fail to have these desirable properties. In this paper, we propose a new extension of the medial axis, called the medial residue, and prove that it is a finite curve network homotopy equivalent to the original surface when the input is a piecewise linear surface with boundary. We also develop an efficient algorithm to compute the medial residue on a triangulated mesh, building on previously known work to compute geodesic distances.

1 Introduction

The medial axis of an object is a skeletal structure originally defined by Blum [1]. It is the set of points having more than one closest points (under the Euclidean distance metric) on the boundary of the object. The medial axis is centered within the object, homology equivalent to the object if it is an open bounded subset of \mathbb{R}^n [6], and (at least) one dimension lower than that of the object. These properties make the medial axis ideal for many applications including shape analysis and robotic path planning.

We are interested in defining a similar skeletal structure on a surface S (with boundary) that inherits the properties of the medial axis. Such a structure could then be used for applications such as shape analysis of surface patches as well as path planning in non-planar domains. We are particularly interested in the case when S is piecewise smooth, which is more representative of typical outputs of discrete surface reconstruction algorithms (e.g., triangulated meshes) than globally smooth surfaces.

*Department of Mathematics and Computer Science, Saint Louis University, echambe5@slu.edu. Research supported in part by NSF grant (CCF 1054779).

†Department of Computer Science and Engineering, Washington University in St. Louis, taoju@cse.wustl.edu. Research supported in part by NSF grant (IIS 0846072).

‡Department of Mathematics and Computer Science, Saint Louis University, letscher@slu.edu.

A natural approach would be to replace the Euclidean distances in the medial axis definition by geodesic distances over S [12]. Interestingly, as we will show in this paper (Section 3), several equivalent definitions of the medial axis may yield different structures when S is only piecewise smooth, and none of these definitions guarantees the two essential properties of the medial axis, namely being homotopy equivalent to the original surface and codimension one.

In this paper, we propose a new extension of the medial axis onto a piecewise linear surface S with boundary, which we call the *medial residue* (Section 4), and prove that the structure is a finite curve network that is always homotopy equivalent to S (Section 5). We also describe a quadratic-time algorithm to compute this structure on a piecewise flat surface with boundary embedded in Euclidean space (Section 6).

2 Background and Definitions

We assume the reader is familiar with classical definitions of manifold topology, which can be found in books such as [5, 9, 2]. We shall only review definitions that are specifically relevant to our work.

A *piecewise linear surface* is a 2-manifold (with boundary) with a piecewise linear structure, whose presentation consists of a finite number of triangles glued together along with an intrinsic distance metric on each triangle that is a linear map. Our algorithmic results work in a more restricted class of *piecewise flat surface*, where the piecewise linear structure comes from an embedding of a triangulation of M into \mathbb{R}^3 , so that each triangle will be isometric to a triangle in \mathbb{R}^2 .

Given a vertex v of a piecewise linear surface which is contained in more than two triangles, let $\{f_1, f_2, \dots, f_k\}$ be the faces to which v belongs, where $\theta_i(v)$ is the interior angle of f_i at vertex v . The *total angle* is the sum of all of these angles, $\theta(p) = \sum_i \theta_i(v)$. The *curvature* at v is the value $2\pi - \theta(p)$. A vertex is said to be *convex*, *flat* or *concave* if its curvature is positive, zero or negative.

A *curve* (or path) is (the image of) a map $p : [0, 1] \rightarrow M$; the length of the curve is generally the length of the image in M . A curve is a *geodesic* if it is locally shortest; in other words, no perturbation of the curve will result in a shorter curve. On a piecewise linear surface, geodesics and shortest paths are themselves piecewise linear maps. We say a curve γ *bisects* a piecewise differentiable curve X at time t if $\gamma(t) \in X$ and the two

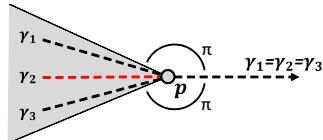
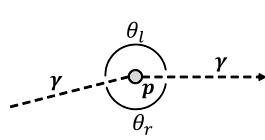


Figure 1: Left: Illustration of left and right curve angles. Right: at a concave vertex p , there may be infinitely many geodesic paths to the boundary (such as $\gamma_1, \gamma_2, \gamma_3$) sharing a common outgoing direction, but only one of them (γ_2) can be straight. Shaded region is the shadow rooted at p , made up of points whose shortest paths to the boundary go through p .

angles bounded by γ and the tangent of X at $\gamma(t)$ are equal. The *curve angles* θ_l and θ_r of a point p on a piecewise linear curve γ are the two angles to the left and right of the curve at p , where $\theta_l + \theta_r$ is the total vertex angle at that point p (see Figure 1 left).

A curve γ is considered *straight* if for each point $p \in \gamma$, the left and right curve angles are equal. This definition was introduced by Polthier and Schmies [10]. It is worth noting that Polthier and Schmies used the term “straight geodesic”, and not simply straight. However, their straight geodesics might in fact not be geodesic (for example, it can go through a convex vertex). In this paper, the term straight geodesic will be used to denote a curve that is both straight and geodesic. Note that although there may be infinitely many geodesic paths to the boundary that go through a concave vertex p , only one of them is straight (see Figure 1 right). We call the region made up of points whose shortest paths to the boundary go through p the *shadow* rooted at p (shaded region in Figure 1 right).

3 The Medial Axis

Let X be a shape in Euclidean space. There are a variety of equivalent ways in which the medial axis of X could be defined. We will consider the following three:

1. Most commonly, the medial axis is defined as the set of points *without a unique closest point* on the boundary of the shape: $\mathcal{M}^{CP} = \{x \in X \mid \nexists \text{ unique } y \in \partial X \text{ with } d(x, \partial X) = d(x, y)\}$
2. Alternatively, the medial axis is the set of points *without a unique shortest path* to the boundary of the shape: $\mathcal{M}^{SP} = \{x \in X \mid \exists \text{ shortest paths } \gamma_1 \neq \gamma_2 \text{ from } x \text{ to } \partial X\}$
3. The medial axis is also the set of points *without a unique direction for shortest paths* to the boundary of the shape. We say two paths γ_1 and γ_2 with $\gamma_1(0) = \gamma_2(0)$ start in the same direction if there exists some $\epsilon > 0$ such that for all

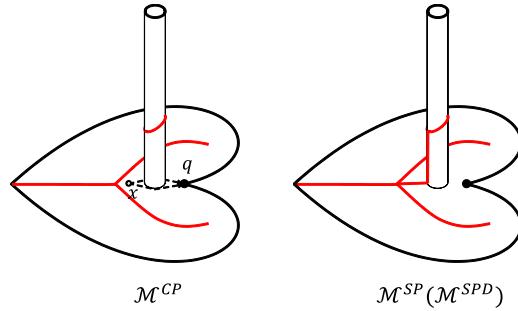


Figure 2: Example where \mathcal{M}^{CP} (red) is not homotopy equivalent to the surface (but \mathcal{M}^{SP} and \mathcal{M}^{SPD} are).

$t < \epsilon$, $\gamma_1(t) = \gamma_2(t)$ (or the curves can be reparameterized so that this holds): $\mathcal{M}^{SPD} = \{x \in X \mid \exists \text{ shortest paths } \gamma_1, \gamma_2 \text{ from } x \text{ to } \partial X \text{ that do not start in the same direction}\}$

We note that the above definitions are all equivalent when X is a smooth manifold in any dimension, but when X is piecewise smooth, these three definitions yield different structures. More precisely, if X is any path metric space (where distances are realized by shortest paths), then $\mathcal{M}^{CP} \subset \mathcal{M}^{SP}$ and $\mathcal{M}^{SPD} \subset \mathcal{M}^{SP}$. The fairly straightforward proof of this can be found in the full version of this paper. More importantly, there are situations where none of the three definitions satisfy the desired properties of being one dimension lower than and homotopy equivalent to X .

First, consider the heart-shaped surface in Figure 2, which has an interior hole on top of a cylindrical protrusion. Note that \mathcal{M}^{CP} excludes points like x in the picture, which has a single closest point q on the boundary (a C^0 corner point) but two shortest paths to q that go around the cylinder. As a result, \mathcal{M}^{CP} consists of two disconnected components. On the other hand, x is included in \mathcal{M}^{SP} and \mathcal{M}^{SPD} .

Next, consider the oval-shaped surface in Figure 3 (a). The surface has a concave vertex v with a large negative curvature that happens to have two shortest paths to two distinct boundary points (a non-generic situation). Since each point in the shadow rooted at v (shaded region in (b)) would have two distinct shortest paths to the boundary, both \mathcal{M}^{CP} and \mathcal{M}^{SP} include the 2-dimensional shadow region. On the other hand, since any point in the shadow has a unique shortest path direction (that follows the geodesic to v), the entire shadow is excluded in \mathcal{M}^{SPD} , and \mathcal{M}^{SPD} has an isolated vertex v that is disconnected from the rest of \mathcal{M}^{SPD} .

4 The Medial Residue

We now define our structure, called the medial residue, which is equivalent to existing definitions of the medial

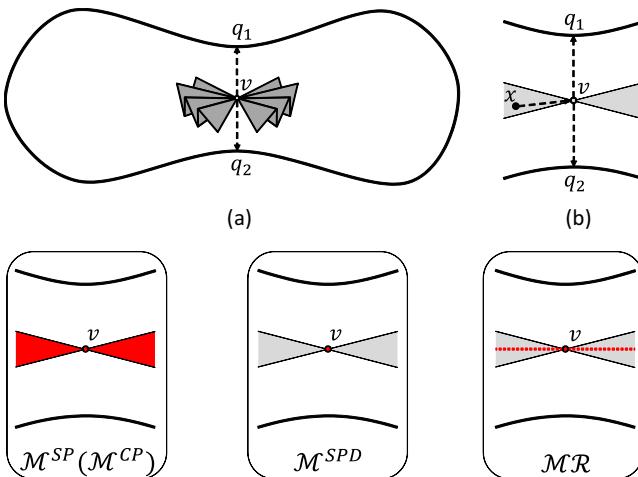


Figure 3: Top: a surface with a highly concave central vertex (a) and a zoom-in view (b). Bottom: different medial axis extensions (red): \mathcal{M}^{CP} and \mathcal{M}^{SP} are 2-dimensional, \mathcal{M}^{SPD} has an isolated vertex, and $\mathcal{M}\mathcal{R}$ is 1-dimensional and homotopy equivalent to the surface.

axis on a smooth manifold but possesses the desired properties of homotopy equivalence and co-dimension one on a piecewise linear surface. To make it clear that we are considering surfaces and not arbitrary manifolds from now on, we will use S instead of X to represent the shape.

We note that our medial residue is well defined on piecewise smooth manifolds, and that the majority of our results hold in these settings. However, our proof about homotopy and dimension holds only for piecewise linear surfaces, although we conjecture that the properties hold in more general settings as well.

The starting point of our definition is \mathcal{M}^{SPD} , which is more complete than \mathcal{M}^{CP} in our first example (Figure 2) and remains low dimension in the second example (Figure 3). Our goal is to add low-dimensional components to \mathcal{M}^{SPD} to restore the homotopy equivalence. Observe that, in our second example, the disconnection in \mathcal{M}^{SPD} takes place in the shadow rooted at a concave vertex $v \in \mathcal{M}^{SPD}$, where the shortest paths from a point x in the shadow to the boundary would agree for some time and then diverge at v . Since we cannot include the entire shadow, which is 2-dimensional, we wish to keep one representative curve. A natural choice of such curve would be one that is “centered” with respect to the two diverging shortest paths at v . More precisely,

Definition 1 The medial residue, $\mathcal{M}\mathcal{R}$ consists of any point $x \in S$ such that either $x \in \mathcal{M}^{SPD}$ or where there are two distinct shortest paths from x to the boundary, γ_1 and γ_2 , parameterized by arc length, which first intersect \mathcal{M}^{SPD} at $v = \gamma_1(t) = \gamma_2(t)$ such that

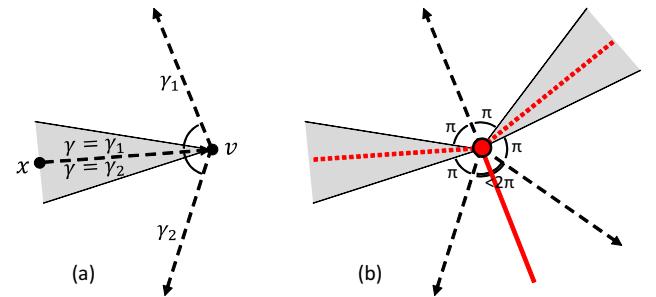


Figure 4: (a): illustration for the definition of a point $x \in \mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$. (b): a generic picture of $\mathcal{M}\mathcal{R}$ at a concave vertex with multiple shortest path directions (solid line is \mathcal{M}^{SPD} and dotted lines are $\mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$).

$\gamma = \gamma_1([0, t]) = \gamma_2([0, t])$ is straight and bisects the angle between the tangents of the two shortest paths from v to the boundary that are nearest to γ on its left and right side.

The definition for a point $x \in \mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$ is illustrated in Figure 4 (a). Note that, by definition, every point on the common segment γ of the shortest paths from x to the boundary is also included in $\mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$. In fact, $\mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$ consists of straight geodesics that bisect shortest path directions at concave vertices of \mathcal{M}^{SPD} . Figure 4 (b) gives a generic picture of $\mathcal{M}\mathcal{R}$ at a concave vertex of \mathcal{M}^{SPD} . The multiple shortest path directions divide the local neighborhood of the vertex radially into *sectors*. Each sector is bisected either by a curve in \mathcal{M}^{SPD} (solid red line), if the sector’s interior angle is less than 2π , or otherwise by a curve in $\mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$ (dotted red line).

Since any point in $\mathcal{M}\mathcal{R} \setminus \mathcal{M}^{SPD}$ has two distinct shortest paths, we have $\mathcal{M}^{SPD} \subset \mathcal{M}\mathcal{R} \subset \mathcal{M}^{SP}$. Since both \mathcal{M}^{SPD} and \mathcal{M}^{SP} are equivalent when S is a smooth manifold, this implies that our medial residue is also equivalent to the other definitions we mentioned earlier in a smooth manifold.

5 Medial Residue on Piecewise Linear Surfaces

In this section, we give sketches of proofs of Theorem 2 and the related lemmas, showing that the medial residue is homotopy equivalent to the original surface; full proofs of each appear in the full version of this paper. As previously mentioned, while we only prove this for piecewise linear surfaces (the main focus of this work), we conjecture that it also holds for piecewise smooth surfaces and higher dimensional manifolds as well.

Theorem 2 If S is a piecewise linear surface with boundary then the medial residue of S is a finite graph that is a deformation retract of S .

To prove this theorem, we will construct a deformation retract by incrementally “eroding” from the boundary, stopping at potentially interesting points along the way. To begin this process, we must understand what a neighborhood of the boundary of S looks like. Let $S_t = \{x \in S \mid d(x, \partial S) \geq t\}$; in other words, S_t is the set of points whose distance from the boundary of S is no less than t . The boundary of S_t is precisely the points with distance t to ∂S .

Our first step is to prove that shortest paths to the boundary are of finite complexity; in other words, they cannot cross the triangulation an unbounded number of times. If our PL surface is a flat embedding in \mathbb{R}^3 , this will follow easily since edges in the triangulation are shortest paths (and no two shortest paths can cross twice), but we must also have a bound for arbitrary PL surfaces. We note that variants of the following proof have been used in the normal surface community for at least 20 years; however, we are unaware of any published reference for such a bound on the number of possible intersections, so we have included it for completeness.

Proposition 3 *The number of intersections between any shortest paths and the underlying triangulation of an arbitrary PL-surface is $\leq |E| \cdot \frac{2\pi}{\delta} \cdot \max_{e \in E} \frac{l(e)}{c_e}$, where δ is the minimum angle at any vertex of the triangulation, $l(e)$ is the length of the edge e , and $c(e)$ is the minimum distance between any pair of points on opposite edges of the quadrilateral formed by the two faces adjacent to an edge e .*

Next, we want to understand what the boundary of the surface looks like at each stage of the erosion process. Locally, ∂S_t consists of a union of straight edges and circular arcs. The straight edges correspond to points whose shortest paths to the boundary do not pass through a vertex of the triangulation, and the circular arcs to points whose shortest paths pass through a vertex. The previous proposition can be used to show that there are finitely many arcs and line segments in ∂S_t .

Lemma 4 *Given a piecewise linear S , for all but finitely many values of t , ∂S_t is a curve. In the cases where ∂S_t is not a curve, ∂S_t is a graph.*

Notice that \mathcal{M}^{SPD} consists of points that have multiple shortest paths directions to the boundary. The above results allow us to bound the combinatorial types of these shortest paths. The points equidistant from the boundary in each shortest path direction are built locally from lines segments and circular arcs. So in a small neighborhood \mathcal{M}^{SPD} consists of the intersection of two curves that are either lines or circles. Hence, \mathcal{M}^{SPD} is built from lines, circles and parabolas. This leads to the following result:

Lemma 5 *If S is piecewise linear, then \mathcal{M}^{SPD} is a finite graph.*

Now we are ready to describe the deformation retract, which immediately implies that \mathcal{MR} is homotopy equivalent to the original PL surface. We will build our deformation retract based on an erosion process which intuitively “pauses” at times $\{t_1, \dots, t_k\}$, where each t_i corresponds to one or more of these three possibilities:

1. There is a vertex v of the triangulation of S with $d(v, \partial S) = t_i$.
2. ∂S_{t_i} is not a disjoint union of curves but instead forms a graph.
3. There is a vertex v of \mathcal{M}^{SPD} with $d(v, \partial S) = t_i$.

The previous lemmas imply that the set of t_i ’s is finite. We will consider the sets S_{t_i} based on our level sets at times $\{t_1, \dots, t_k\}$ described above, as well as the “slice” between two of our level sets, $C_i = (\overline{S_{t_i}} \setminus \overline{S_{t_{i+1}}})$. The following lemma actually shows how we can construct the deformation retract.

Lemma 6 *For each t_i , $S_{t_{i+1}} \cup \mathcal{MR}$ is a deformation retract of $S_{t_i} \cup \mathcal{MR}$.*

Proof. Consider the slice region C_i between two level curves. One of several cases could occur depending on what happens on the boundaries of this region, as illustrated in Figure 5.

The first case is that portions of the boundary ∂S_{t_i} meet at a convex corner. This is shown in Figure 5(a), where the shortest paths are shown on the left and the deformation retract on the right. At such a corner point v , there is a segment of \mathcal{M}^{SPD} going from ∂S_{t_i} to $\partial S_{t_{i+1}}$, which bisects the convex corner at v . Shortest paths from points on this segment hit ∂S_{t_i} near v . The deformation retract follows these curves.

The second case is that portions of the boundary ∂S_{t_i} meet at a concave corner, see Figure 5(b). Note that the concave corner v must contain a shadow rooted at v where there is a cone of shortest paths going through v . By definition of \mathcal{MR} , if $v \in \mathcal{MR}$ then the bisector of the shadow will be in $\mathcal{MR} \setminus \mathcal{M}^{SPD}$. However, the deformation retract cannot simply follow the shortest paths exactly, as this would not be continuous at v ; observe in the figure that points near v are taken to opposite sides of the bisector and do not move continuously. Instead, very near this point, the deformation retract will take points to either the bisector or the full shadow, as shown on the right in Figure 5(b). Note that the reparameterization continuously deforms points from ∂S_{t_i} onto the union of $\partial S_{t_{i+1}}$ and \mathcal{MR} .

In the third case, consider points $v \in \partial S_{t_i}$ where the ∂S_{t_i} is smooth. A single shortest path passes through v . If $v \notin \mathcal{MR}$, the deformation retract simply follows this path backwards, as shown in Figure 5(c). Otherwise, if $v \in \mathcal{MR}$, there is a segment of a bisector in \mathcal{MR} that contains v and continues in a direction that

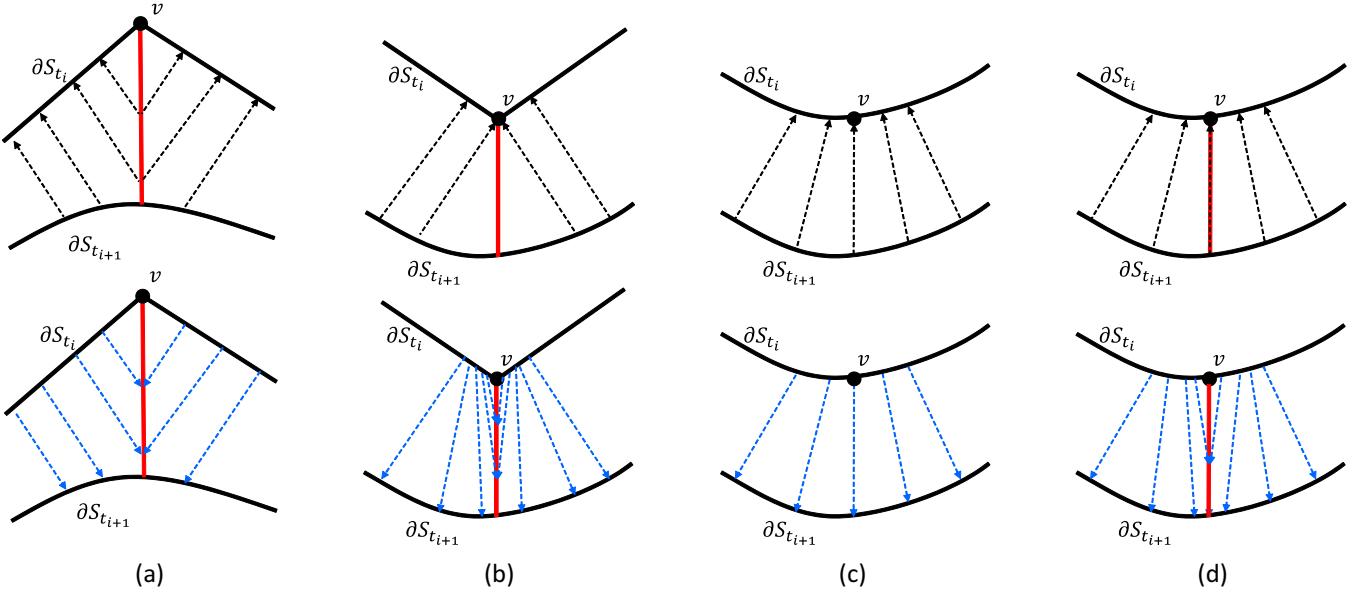


Figure 5: The shortest paths (black arrows), medial residue (red lines) and deformation retract (blue arrows) at the points in the slice region.

is perpendicular to ∂S_{t_i} . In this situation (as in the second case above), the shortest paths cannot be used as a deformation retract as it would not be continuous at that v . However, a similar re-parameterization as in the second case can be used in a local neighborhood of this portion of \mathcal{MR} to construct a deformation retract, as shown in Figure 5(d).

Note that it is possible that ∂S_{t_i} is a graph, in which case the deformation retract described in the three cases above can be applied to individual components of C_i that are incident to a point $v \in \partial S_{t_i}$. \square

Finally, to show that \mathcal{MR} is a finite graph, we observe that the set of concave vertices in \mathcal{M}^{SPD} and the set of sectors around each such vertex are both finite on a PL surface, which implies that $\mathcal{MR} \setminus \mathcal{M}^{SPD}$ consists of a finite number of straight geodesic paths that bisect these sectors. Together with our previous lemma that \mathcal{M}^{SPD} is a finite graph, this completes the proof of Theorem 2.

6 Algorithm

We next give an overview of our algorithm to compute the medial residue on a piecewise flat surface with triangle faces in \mathbb{R}^3 , a commonly used discretization in many applications. (Further details of the algorithm can be found in the full version of the paper.)

We first recall some essential properties of shortest paths on a triangulated surface S [7, 4]. We assume the boundary ∂S consists of vertices and edges of some triangle faces. A shortest path p that connects any point

$x \in S$ to the boundary ∂S originates either from a vertex or an interior point of an edge. In the latter case, p is orthogonal to that originating edge. The path p may go through some vertex of S , and if it does, both the left and right curve angles made by p at that vertex are greater than or equal to π . Away from the vertices, p is a straight line segment after unfolding the triangles that p goes through onto a plane. We call the last vertex visited by p before reaching x the *root* of p . If p does not go through any vertex, the root is the originating vertex or edge on ∂S . The *last edge sequence* of p is the (possibly empty) sequence of edges that p goes through between the root and x .

The starting point of our algorithm is a subdivision of each triangle face into regions where the shortest paths have a common combinatorial structure. Given a face f , a root r (being either a vertex or edge), and an edge sequence E , a *cell* is the set of points $x \in f$ such that some shortest path from x to ∂S has root r and last edge sequence E . The curve segments that bound the cells (including both interior segments on f and the segments on the edges of f) form a graph, which is called the *subdivision graph*. The subdivision can be computed using an easy extension of existing methods [8, 7, 4] in $O(n^2 \log n)$ time and $O(n^2)$ space.

Given the subdivision graph, our algorithm first identifies a subset of the graph as \mathcal{M}^{SPD} , then adds in the bisectors to form the complete \mathcal{MR} . Both steps can be done in $O(n^2)$ time and space, where n is the number of triangles of the surface. The overall process, taking into account the creation of the face subdivision, can be

done in $O(n^2 \log n)$ time and $O(n^2)$ space. We assume exact arithmetic is used to precisely compute distances and angles.

6.1 Computing \mathcal{M}^{SPD}

First, we observe two relations between \mathcal{M}^{SPD} and a subdivision graph:

Lemma 7 \mathcal{M}^{SPD} is a subset of the subdivision graph.

Lemma 8 Let h be a segment in the subdivision graph, then either all interior points of h lie in \mathcal{M}^{SPD} or none of them does.

The algorithm simply goes through each element (a vertex or a segment open at its ends) of the subdivision graph. For each element l , it picks an arbitrary point $x \in l$ and gathers shortest path directions at x by examining each incident cell of l . l is included in \mathcal{M}^{SPD} as soon as two distinct shortest path directions are found.

Since computing the shortest path direction given a cell takes constant time, the complexity of the algorithm is proportional to the number of pairs of an element and an incident cell, which is linear to the number of elements in the subdivision graph. The algorithm uses a data structure that maintains adjacency between cells and subdivision graph elements, which is again linear to the complexity of the graph. Hence computing \mathcal{M}^{SPD} can be done in $O(n^2)$ time and $O(n^2)$ space.

6.2 Computing $\mathcal{MR} \setminus \mathcal{M}^{SPD}$

We use a tracing algorithm to compute bisectors that make up $\mathcal{MR} \setminus \mathcal{M}^{SPD}$. For each sector bounded by shortest path directions at some concave vertex $v \in \mathcal{M}^{SPD}$, we start tracing a straight and shortest path from v in the bisecting direction of the sector. Tracing proceeds in a cell-by-cell manner, creating straight line segments within each cell and maintaining straightness while marching to the next cell. Tracing ends when the path hits a segment or vertex of the subdivision graph that belongs to \mathcal{M}^{SPD} .

Tracing within a cell involves intersecting a line with several low-degree algebraic curves. Since the intersection of a cell with a shortest path to the boundary is a single line segment [8], tracing in a cell can be done in time linear to the number of segments of the cell. Marching from one cell to the next can be done in constant time using an adjacency structure. To bound the complexity of tracing all bisectors, the key is to observe that each cell can contain a non-trivial portion of at most one bisector. This is because only a cell whose shortest paths to the boundary are rooted at some vertex may contain a bisector rooted at the vertex, and the angle made by any two bisectors rooted at a vertex is at least 2π . So the total tracing time for all bisectors

is bounded by the sum of number of segments over all cells, which is $O(n^2)$. Tracing uses $O(n^2)$ space since it adds only a constant amount of additional data per element of the subdivision graph.

References

- [1] H. Blum. A transformation for extracting new descriptors of form. *Models for the Perception of Speech and Visual Form*, pages 362–80, 1967.
- [2] Manfredo P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.
- [3] Jeff Erickson and Amir Nayyeri. Tracing compressed curves in triangulated surfaces. In *Proceedings of the 2012 symposium on Computational Geometry*, SoCG ’12, pages 131–140, New York, NY, USA, 2012. ACM.
- [4] M. Fort and J.A. Sellares. Generalized source shortest paths on polyhedral surfaces. In *Proceedings of the 23rd European Workshop on Computational Geometry*, pages 186–189, March 2007.
- [5] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.
- [6] André Lieutier. Any open bounded subset of \mathbb{R}^n has the same homotopy type as its medial axis. *Computer-Aided Design*, 36(11):1029 – 1046, 2004. Solid Modeling Theory and Applications.
- [7] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, August 1987.
- [8] David Mount. Voronoi diagrams on the surface of a polyhedron. Technical report, Dept. of Computer Science, Univ. of Maryland, Baltimore, MD, 1985.
- [9] James Munkres. *Topology*. Pearson, 2000.
- [10] Konrad Polthier and Markus Schmies. Straightest geodesics on polyhedral surfaces. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH ’06, pages 30–38, New York, NY, USA, 2006. ACM.
- [11] Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comput.*, 14(2):448–468, 1985.
- [12] F.-E. Wolter and K.-I. Fries. Local and global geometric methods for analysis interrogation, reconstruction, modification and design of shape. In *Proceedings of the International Conference on Computer Graphics*, CGI ’00, pages 137–151, Washington, DC, USA, 2000. IEEE Computer Society.

Morpion Solitaire 5D: a new upper bound of 121 on the maximum score

Akitoshi Kawamura* Takuma Okamoto† Yuichi Tatsu† Yushi Uno§ Masahide Yamato†

Abstract

Morpion Solitaire is a pencil-and-paper game for a single player. A move in this game consists of putting a cross at a lattice point and then drawing a line segment that passes through exactly five consecutive crosses. The objective is to make as many moves as possible, starting from a standard initial configuration of crosses. For one of the variants of this game, called 5D, we prove an upper bound of 121 on the number of moves. This is done by introducing *line-based analysis*, and improves the known upper bound of 138 obtained by potential-based analysis.

Keywords: *pencil-and-paper game, lattice points, line-based analysis.*

1 Introduction

Morpion Solitaire, also known as *Join Five*, is a game played alone with a pencil and paper, and it is popular in several countries [4]. A move in this game consists of drawing a cross and a line segment on an infinite square lattice. The line segment has to pass through exactly five consecutive crosses including the one that has just been placed. The objective is to make as many moves as possible starting from a given initial configuration. We call the number of moves the *score*. There are two variants of this game according to how two line segments can touch each other.

Demaine et al. [6] studied generalizations of the game and their computational complexity, and show that a generalized Morpion Solitaire is NP-hard and that its maximum score is hard to approximate. Another target of interest is the maximum scores or their lower and upper bounds. Recently, computing maximum scores was used as a test problem to evaluate the effectiveness of the Monte-Carlo tree search method, which has been attracting rising attention as a promising approach in game programming [5, 9].

In this paper, we focus on the 5D variant of the game, and show improved upper bounds on the maximum score. We first show that the known upper bound

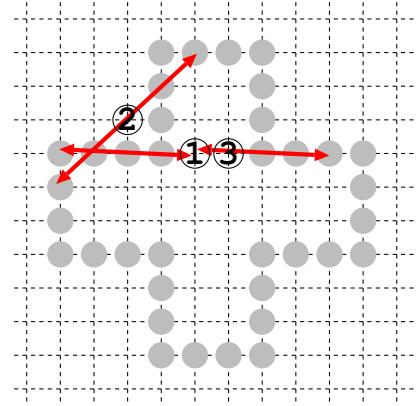


Figure 1: The standard initial board layout for Morpion Solitaire 5D and 5T, and an example of the first three moves. Each cross placed in these moves is denoted by a number surrounded by a circle. Move 3 is allowed in 5T (touching) but not in 5D (disjoint).

of 138 can be improved to 136 by pushing on the existing potential-based approach. Next we introduce a line-based approach and further improve the bound to 121. We also try to organize and present related results, since there are relatively few research papers on this topic.

2 Rules and Records

2.1 Rules

Morpion Solitaire is played on an infinite square lattice. Initially 36 crosses are drawn on lattice points so that they form a large cross shape with edge length 4 as shown in Figure 1. In this figure, a cross is denoted by a circle. (In this paper, the length of a line segment means the number of crosses covered by it.)

A *move* consists of the following two steps applied in this order. The objective of this game is to maximize the number of moves.

1. Draw a new *cross* on a lattice point which is empty (no cross exists) on the current board.
2. Draw a segment of length 5 (called a *line*) that passes through exactly five consecutive crosses including the one drawn in step 1 of this move. Here, the line can be drawn in either one of the four directions, vertical, horizontal, or diagonal. Two lines in the same direction may not overlap.

*Department of Computer Science, University of Tokyo, Japan. kawamura@is.s.u-tokyo.ac.jp

†Graduate School of Science, Osaka Prefecture University, Japan. {ss301002,sr301023,sr301036}@edu.osakafu-u.ac.jp

§Graduate School of Science, Osaka Prefecture University, Japan. uno@mi.s.osakafu-u.ac.jp

There are two variants of this game depending on whether two lines in the same direction can touch (5T) or have to be disjoint (5D) (Figure 1). We mainly discuss about 5D in this paper.

When a line L passes a cross C , we say that L *covers* the cross or the lattice point on which it is drawn. We sometimes call a board after move N a board at move N . Also we sometimes denote a cross and a line drawn in move N by C_N and L_N , respectively.

2.2 Records

The above definition of the game can be extended to αD and αT , where the lines have length α and the edges of the large cross in the initial configuration have length $\alpha - 1$, however, the maximum scores are known for all variants except $\alpha = 5$. For 3T and 3D, the maximum scores are not bounded, as there are sequences of moves that can be repeated infinitely [6]. For 6T and 6D, we can easily see that the maximum score is 12. For 4T and 4D, there used to be gaps between the maximum achieved scores and the upper bounds in the past, but in 2007, 62 and 35 moves were achieved for 4T and 4D, respectively [7], and these scores were proved to be optimal in 2008 [4].

Table 1 [4] shows the current maximum scores of 5T and 5D. We briefly explain how the records of these two variants have been developed.

5T. Bruneau achieved 170 in 1976 by hand [2]. In 2010, by computer, Akiyama, Komiya and Kotani [1] used Monte-Carlo tree search to achieve 145 and 146, which were still less than human’s record at that time. From 2010 to 2011, also by computer, Rosin achieved 172, beating human’s record [3]. Rosin [9] improved the record to 177 in 2011, and the current record is 178 [10]. An upper bound of 705 on the maximum score is known [6].

5D. According to Demaine et al. [6], 68 moves was achieved by hand in 1999. Cazenave [5] established 80 in 2008, and then Rosin [9] improved it to 82 in 2010, both by computers. As for upper bounds, Demaine et al. [6] showed 141 in 2006 [6] and Karjalainen showed 138 in 2011 [8].

Recent records of maximum scores of both 5T and 5D were obtained by computers. The framework used for this was Monte-Carlo tree search or its extensions,

Table 1: Records on Morpion Solitaire 5T and 5D: their maximum achieved scores and proven upper bounds.

game type	best achieved score	upper bound
5T	178	705
5D	82	138

which are known to produce excellent results in designing computer programs, for example, for playing Shogi or Go against humans.

Hereafter, in this paper, we focus only on 5D variant and aim to improve the upper bound on its maximum score, which is known to be 138.

3 Potential-based Analysis of Upper Bounds

The known upper bound of 138 on the maximum score of Morpion Solitaire 5D is obtained by arguments using ‘potentials’. In this section, we explain potentials and the related results, and then show that the upper bound can be improved to 136 by a more detailed analysis based on this approach.

3.1 Preceding Research

The notion of potential in the analysis of Morpion Solitaire seems to have been originally introduced in folklore discussions and was used by Demaine et al. [6]. The *potential* of a cross on a board is the number of additional lines that can cover it. Since a cross can be covered by at most four lines (in the vertical, horizontal and two diagonal directions), the potential of a cross C is formally given by

$$4 - (\text{number of lines that cover } C).$$

We define the *total potential* of a board to be the sum of the potentials of all crosses on that board.

Now we can observe the following three facts about Morpion Solitaire 5D.

Observations

- (i) The total potential of the initial board is 144.
- (ii) The total potential decreases at least by 1 in every move.
- (iii) At any time, playing the next move requires at least a total potential 4.

We have (i) because initially there are 36 crosses, each of which has potential 4. We have (ii) because step 1 of a move in 5D adds 4 to the total potential, and step 2 decreases the potential by 5.

Demaine et al. [6] showed the following upper bound based on the above three observations.

Theorem 1 ([6]) *The number of moves in Morpion Solitaire 5D cannot exceed 141.*

To see this, let M be the maximum score (the number of moves). The total potential after $M - 1$ moves must be at least 4, that is, $144 - (M - 1) \geq 4$.

Karjalainen [8] improved this argument and obtained the following result by showing that the total potential at any time is at least 6.

Theorem 2 ([8]) *The number of moves in Morpion Solitaire 5D cannot exceed 138.*

To see this, let M be the maximum score and consider the last three moves. The crosses drawn in the last three moves M , $M - 1$ and $M - 2$ are eventually covered by one line, by at most two lines, and by at most three lines, respectively. In other words, those crosses have potentials 3, ≥ 2 , and ≥ 1 , respectively, at the end of the game. This implies $144 - M \geq 6$, and thus $M \leq 138$.

3.2 Improvements

We next show some small improvements of maximum scores in the framework of potential-based analysis. Our improvements are obtained by focusing on the last four moves. We denote the potential of a cross C on a board by $p(C)$.

Lemma 3 *The sum of the potentials of the three crosses that are drawn in the last three moves is greater than or equal to 7.*

Proof. Consider the board at move N . According to the arguments for Theorem 2, $p(C_N) = 3$, $p(C_{N-1}) \geq 2$ and $p(C_{N-2}) \geq 1$ hold for crosses C_N , C_{N-1} and C_{N-2} at moves N , $N - 1$ and $N - 2$, respectively. Here, $p(C_N) = 3$, $p(C_{N-1}) = 2$ and $p(C_{N-2}) = 1$ cannot be satisfied simultaneously. Suppose they can. Then line L_{N-1} has to cover cross C_{N-2} as well as C_{N-1} , and line L_N has to cover both crosses C_{N-2} and C_{N-1} as well as cross C_N , and this forces such two lines L_{N-1} and L_N to overlap. This contradicts the rules of Morpion Solitaire, and thus $p(C_N) + p(C_{N-1}) + p(C_{N-2}) > 6$ holds. \square

Lemma 3 alone improves an upper bound to 137, and we can save one more move.

Theorem 4 *The number of moves in Morpion Solitaire 5D cannot exceed 136.*

Proof. Let M be the maximum score, and consider a board at move $M - 1$. First, we can see that in order that move M is feasible, there exists a cross C other than C_{M-1} , C_{M-2} and C_{M-3} with $p(C) \geq 1$. Then we determine the total potential of board $M - 1$ by a case analysis; whether line L_M drawn in move M covers all three crosses C_{M-1} , C_{M-2} and C_{M-3} , or not.

Case 1: line L_M covers all crosses C_{M-1} , C_{M-2} and C_{M-3} . In this case, three crosses C_{M-1} , C_{M-2} and C_{M-3} lie on a common lattice line. Since no two lines can overlap, line L_{M-2} that covers C_{M-3} and line L_{M-1} that covers both C_{M-2} and C_{M-3} are not compatible. Hence, $p(C_{M-1}) = p(C_{M-2}) = p(C_{M-3}) = 3$ holds. This, together with the fact that there exists a cross C with $p(C) \geq 1$ other than C_{M-1} , C_{M-2} and C_{M-3} guarantees $p(C_{M-1}) + p(C_{M-2}) + p(C_{M-3}) + p(C) \geq 10$.

Case 2: line L_M does not cover at least one of crosses C_{M-1} , C_{M-2} or C_{M-3} . In this case, there must exist two different crosses C and C' with $p(C) \geq 1$ and $p(C') \geq 1$. Therefore, together with Lemma 3,

$p(C_{M-1}) + p(C_{M-2}) + p(C_{M-3}) + p(C) + p(C') \geq 9$ holds.

To put both cases together, the total potential of an arbitrary board of move $M - 1$ is greater than or equal to 9. That is, $144 - (M - 1) \geq 9$ holds, which implies $M \leq 136$. \square

4 Line-based Analysis of Upper Bounds

In this section, we introduce a new approach for deriving better upper bounds, which we call the line-based analysis. It is based on the relationship between the number of lines on a board and the number of lattice points they cover.

The following observation is easy but crucial.

Fact After N moves, there are $N + 36$ crosses and N lines.

Let $c(N)$ denote the minimum number of lattice points that are covered by N lines of length 5 in an arbitrary layout on a board (lattice plane). Then in order for a board of move N to be feasible (realizable), it has to satisfy that $c(N) \leq N + 36$. Conversely, for N that satisfies $c(N) > N + 36$, such a move N is infeasible. Here, since this game proceeds move by move, if a board of move N is infeasible then all boards of moves greater than N are infeasible. Hence, these observations imply the following property.

Property (Board Infeasibility Condition) If there exists N that satisfies $c(N) > N + 36$, then an upper bound on the maximum score is $N - 1$.

In the subsequent discussions, we derive new upper bounds on the maximum score by fully utilizing this property. In this case, however, since it is not easy to obtain $c(N)$ directly, we compute a lower bound $c'(N)$ on $c(N)$, and we try to find N that satisfies the Board Infeasibility Condition for that $c'(N)$.

4.1 An Upper Bound of 132

Here, we count the number of lattice points covered by lines by focusing on lines in one direction among four that we draw arbitrarily. Then we have the following lower bound on $c(N)$.

Claim 5 *For any move N , $c(N) \geq \lceil \frac{N}{4} \rceil \times 5$ holds.*

Proof. Since we draw N lines in all, there is a direction in which at least $\lceil \frac{N}{4} \rceil$ lines are drawn. They cover at least $\lceil \frac{N}{4} \rceil \times 5$ lattice points. \square

By Claim 5, we have the following upper bound.

Theorem 6 *The number of moves in Morpion Solitaire 5D cannot exceed 132.*

Proof. In case that $N = 133$, $c(N) \geq \lceil \frac{133}{4} \rceil \times 5 = 170$ holds according to the claim. On the other hand, $N + 36 = 169$ and this N satisfies the Board Infeasibility Condition. \square

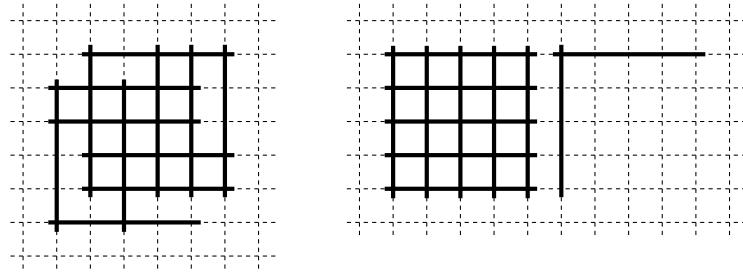


Figure 2: Six lines in each of two directions cover at least 34 lattice points.

4.2 An Upper Bound of 121

In the previous arguments, to count the number of lattice points covered by lines, we focused on the lines only in one direction. By considering two directions, we will obtain a tighter lower bound on $c(N)$. We first prove a technical lemma.

Lemma 7 *Suppose that $5k + \beta$ ($k \geq 0; 5 > \beta \geq 0$) lines of length 5 are drawn in each of two different directions (among the possible four). Then they cover at least $(5k + \beta) \times 5 + 5\beta - \beta^2$ lattice points.*

Proof. We assume without loss of generality that the two different directions are vertical and horizontal. We color vertical lattice lines on the board periodically with different five colors, and consider the situation where $5k + \beta$ lines are drawn arbitrarily along the vertical and horizontal directions on that board. Notice here that the number of lattice points covered by line is the same in both the vertical and horizontal directions, that is $(5k + \beta) \times 5$. Then we can observe that

- (i) in all the lattice points covered by lines drawn in horizontal directions, there are exactly $5k + \beta$ points colored in each one of five colors, and
- (ii) if we classify the lattice points covered by vertical lines by their colors, there are at least $5k + 5$ points in some β colors out of five.

Therefore, at least $\beta(5 - \beta)$ out of $5k + 5$ lattice points are not covered by horizontal lines. Consequently, these lines cover $(5k + \beta) \times 5 + \beta(5 - \beta)$ lattice points. \square

Figure 2 shows two different layouts of lines where this lemma holds for $k = 1$. Moreover, Lemma 7 can be generalized as follows for different lengths of lines.

Lemma 8 *Suppose that $k\alpha + \beta$ ($k \geq 0; \alpha > \beta \geq 0$) lines of length α are drawn in each direction of two different directions on board. Then they cover at least $\alpha(k\alpha + \beta) + \beta\alpha - \beta^2$ lattice points.*

In the following claim, we use Lemma 7 with $\beta = 1$. That is, if we draw $5k + 1$ lines in each of two different directions, they cover at least $(5k + 1) \times 5 + 4$ lattice points.

Claim 9 *For a move N , if $N \not\equiv 1 \pmod{4}$ and $\lceil \frac{N}{4} \rceil \equiv 1 \pmod{5}$, then $c(N) \geq \lceil \frac{N}{4} \rceil \times 5 + 4$.*

Proof. If the maximum number of lines drawn in a certain direction is greater than or equal to $\lceil \frac{N}{4} \rceil + 1$, the number of lattice points covered by some line is at least $\lceil \frac{N}{4} \rceil \times 5 + 5$ and the statement trivially holds. So suppose otherwise, that is, the maximum number of lines drawn in one direction is equal to $\lceil \frac{N}{4} \rceil$. Since $N \not\equiv 1 \pmod{4}$, at least $\lceil \frac{N}{4} \rceil$ lines are drawn in more than one direction. Since this number $\lceil \frac{N}{4} \rceil$ equals $5k + 1$ for some k by assumption, we can apply Lemma 7 to conclude that the lines drawn in these two directions cover at least $\lceil \frac{N}{4} \rceil \times 5 + 4$ lattice points. This implies the desired inequality. \square

Using this fact, we obtain a new upper bound.

Theorem 10 *The number of moves in Morpion Solitaire 5D cannot exceed 121.*

Proof. When $N = 122$, since $122 \equiv 2 \pmod{4}$ and $\lceil \frac{122}{4} \rceil = 1 \pmod{5}$, the hypothesis of Claim 9 is satisfied, and thus $c(122) \geq 31 \times 5 + 4 = 159$. Since this exceeds $N + 36 = 158$, we have the Board Infeasibility Condition. \square

4.3 Remarks

We mention that a similar argument to Claim 9 holds when $N \not\equiv 1 \pmod{4}$ and $\lceil \frac{N}{4} \rceil \equiv 2$ or $3 \pmod{5}$. In this case, if the maximum number of lines drawn in a certain direction is $\lceil \frac{N}{4} \rceil$, the number of lattice points covered by some line is at least $\lceil \frac{N}{4} \rceil \times 5 + 6$. On the other hand, if the maximum number of lines drawn in a certain direction is equal to or greater than $\lceil \frac{N}{4} \rceil + 1$, that is at least $\lceil \frac{N}{4} \rceil \times 5 + 5$. So putting these two cases together, we have $c(N) \geq \lceil \frac{N}{4} \rceil \times 5 + 5$. However, such N that satisfies this hypothesis and the Board Infeasibility Condition is at least 126, and thus we know that an upper bound on the maximum score can be improved to 125 at best.

We also note a limitation of this approach of trying to use $c(N)$: we cannot obtain an upper bound smaller than 102 by proving the Board Infeasibility Condition. This is because we have $c(N) \leq N + 36$ for all $N \leq 102$. Figure 3 proves this inequality for $N = 102$, and we can also easily confirm that it holds for all smaller N .

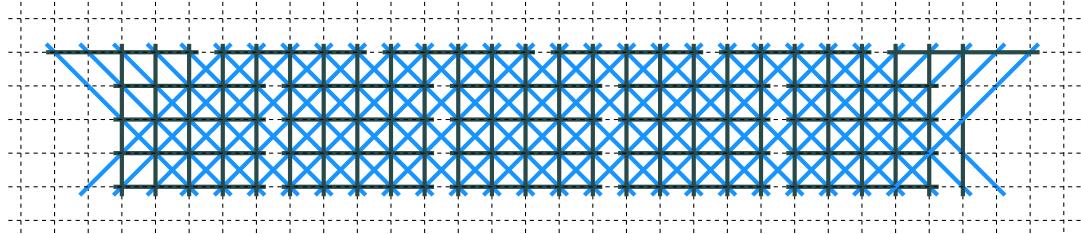


Figure 3: 102 lines cover 138 lattice points.

5 Conclusion

Although the ultimate goal of this game is to achieve the true maximum score, there are some other interesting questions.

It is possible that some idea based on our line-based analysis can further improve the upper bound on the maximum score of 5D. For example, we may consider more than two directions in those arguments. Also we may somehow take the initial layout of 36 crosses into account, which we did not in this paper.

We can also try to apply our line-based analysis to 5T. There are variants of 5T called 5T+ and 5T++, defined by relaxing the original rules about the relationship between the numbers of crosses and the lines; see Boyer's web page [4]. On this web page, he shows how to play 317 moves in the 5T++ variant, and expects that this number may be the best possible (and hence may give an upper bound for 5T). Our line-based approach may help prove upper bounds close to this.

References

- [1] H. Akiyama, K. Komiya and Y. Kotani. Nested Monte-Carlo search with AMAF heuristic. *Proc. International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 172–176 (2010).
- [2] P. Berloquin. Mini-morpion et nouveaux problèmes de dominos. *Science and Vie*, pp. 130–131 (1976).
- [3] C. Boyer. Morpion Solitaire, le record est enfin battu!, *Science and Vie*, pp. 144–147 (2010).
- [4] C. Boyer. Morpion Solitaire. <http://www.morpionsolitaire.com/>
- [5] T. Cazenave. Nested Monte-Carlo search. *Proc. 26th International Joint Conference on Artificial Intelligence*, pp. 456–461 (2009).
- [6] E. D. Demaine, M. L. Demaine, A. Langerman and S. Langerman. Morpion Solitaire. *Theory of Computing Systems*, 39(3), pp. 439–453 (2006).
- [7] H. Hyyrö and T. Poranen. New heuristics for Morpion Solitaire. <http://www.sis.uta.fi/~tp54752/pub/morpion-article.pdf> (2007).
- [8] P. Karjalainen. Bounding the upper limit of moves in the game of Morpion Solitaire 5D. <http://www.morpionsolitaire.com/Karjalainen.pdf> (2011).
- [9] C. D. Rosin. Nested rollout policy adaptation for Monte Carlo tree search. *Proc. 27th International Joint Conference on Artificial Intelligence*, pp. 649–654 (2011).
- [10] C. D. Rosin. A new Morpion Solitaire record via Monte-Carlo search. <http://www.chrisrosin.com/morpion/>

Computational complexity and an integer programming model of Shakashaka

Erik D. Demaine*

Yoshio Okamoto†

Ryuhei Uehara‡

Yushi Uno§

Abstract

Shakashaka is a pencil-and-paper puzzle proposed by Guten and popularized by the Japanese publisher Nikoli (like Sudoku). We determine the computational complexity by proving that Shakashaka is NP-complete, and furthermore that counting the number of solutions is #P-complete. Next we formulate Shakashaka as an integer programming (IP) problem, and show that an IP solver can solve every instance from Nikoli’s website within a second.

Keywords: integer programming, NP-completeness, pencil-and-paper puzzle, Shakashaka

1 Introduction

The puzzle *Shakashaka* is one of many pencil-and-paper puzzles (such as the famous Sudoku) popularized by Japanese publisher Nikoli. Shakashaka was proposed by Guten in 2008, and since then, has become one of the main Nikoli puzzles.

An instance of Shakashaka consists of an $m \times n$ rectangular board of unit squares. Each square is either white or black, and some black squares contain a number. A candidate solution to the puzzle consists of filling in some of the white squares with a black half-square (isosceles right triangle filling half the area) in one of the four ways: □, ▲, ▢, ▣. We call such squares *b/w squares*; white squares may also be left entirely white. Each number in a black square specifies the number of b/w squares that should be among four (vertically or horizontally adjacent) neighbors of the black square. (A black square without a number allows any number of b/w neighbors.) The objective of the puzzle is to fill the white squares in the given board while satisfying the above constraints and so that the remaining white area consists only of (empty) squares and rectangles. An example of the puzzle Shakashaka in [1] is shown in Figure 1(a), and its (unique) solution is given in Figure 1(b).

*Computer Science and Artificial Intelligence Laboratory, MIT, edemaine@mit.edu

†Graduate School of Informatics and Engineering, The University of Electro-Communications (UEC), okamotoy@uec.ac.jp

‡School of Information Science, JAIST, uehara@jaist.ac.jp

§Graduate School of Science, Osaka Prefecture University (OPU), uno@mi.s.osakafu-u.ac.jp

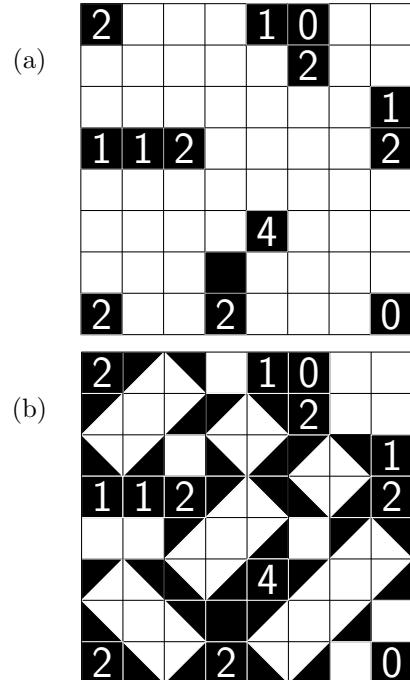


Figure 1: An instance of the puzzle Shakashaka and its solution ([1])

As mentioned in the literature [2], a lot of pencil-and-paper puzzles have been shown NP-complete. However, the computational complexity of Shakashaka has not yet been studied. In this paper, we prove that Shakashaka is NP-complete, by a reduction from planar 3SAT. Because our reduction preserves the number of solutions, we also prove that counting the number of solutions to a Shakashaka puzzle is #P-complete.

Next we show how to formulate Shakashaka as a 0-1 integer programming problem (a linear programming problem in which all variables are restricted to be 0 or 1). Although integer programming is one of Karp’s 21 NP-complete problems, there are many efficient solvers from a practical point of view. For example, recent solvers run around one billion times faster than those from 1991 [3]. Therefore, once we can formulate a puzzle as a 0-1 integer linear programming problem, we can hope to use these solvers to solve the puzzle efficiently in practice.

Some authors have proposed integer-programming formulations of several puzzles before, mainly for the

didactic purposes [4, 5, 6, 7, 8]. The formulation of Shakashaka is not so straightforward because we have to avoid forming nonrectangular orthogonal shapes or nested rectangles. We show that our formulation characterizes the constraints of Shakashaka. We also perform computational experiments, and observe that each instance from Nikoli’s website can be solved within one second.

2 Preliminaries

Let us begin with a formal definition of the puzzle Shakashaka. An instance I of *Shakashaka* is a rectangular board of size $m \times n$. Each unit square is colored either *white* or *black*. A black square may contain a *number* $i \in \{0, \dots, 4\}$. A *solution* of the instance I is a mapping from the set of white squares in I to the set $\{\square, \blacksquare, \blacksquare, \blacksquare, \blacksquare\}$ satisfying the following conditions:

1. Each white square mapped to \square is left uncolored (white), while each square mapped to $\blacksquare, \blacksquare, \blacksquare$, or \blacksquare is colored black and white as indicated (and called a *b/w square*).
2. Each black square that contains the number i has exactly i b/w squares among its four neighbors.
3. Each connected white area forms a white rectangle (or square).

Computationally, Shakashaka is a decision problem: for a given instance, does it have a solution? The *counting version* of Shakashaka asks to compute the number of distinct solutions to the given instance.

3 NP-completeness of Shakashaka

In this section, we prove the following theorem:

Theorem 1 *Shakashaka is NP-complete.*

The proof is by a reduction from planar 3SAT, one of the well-known NP-complete problems [9]. Let F be an instance of planar 3SAT. That is, F consists of a set $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$ of m clauses over n variables $\mathcal{V} = \{x_1, x_2, \dots, x_n\}$, where each clause C_i consists of three literals, and the graph $G = (\mathcal{C} \cup \mathcal{V}, \mathcal{E})$ is planar, where \mathcal{E} contains an edge $\{C_i, x_j\}$ if and only if literal x_j or \bar{x}_j is in the clause C_i .

Now we show a reduction from F to an instance I of Shakashaka. The key idea is to use the pattern shown in Figure 2. For the pattern in Figure 2(a), we have two choices for filling the 2×2 white squares as shown in Figure 2(b). Essentially, this works as a “wire” to propagate a signal. We regard the 2×2 square containing the four white unit squares in Figure 2(b) as representing “0,” and the big diamond containing four (different) b/w squares in 2(b) as representing “1.” That is, the

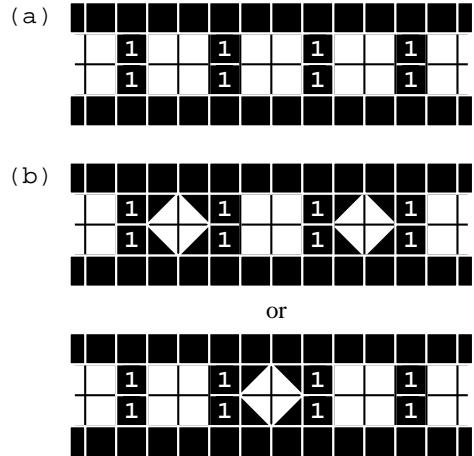


Figure 2: Basic pattern

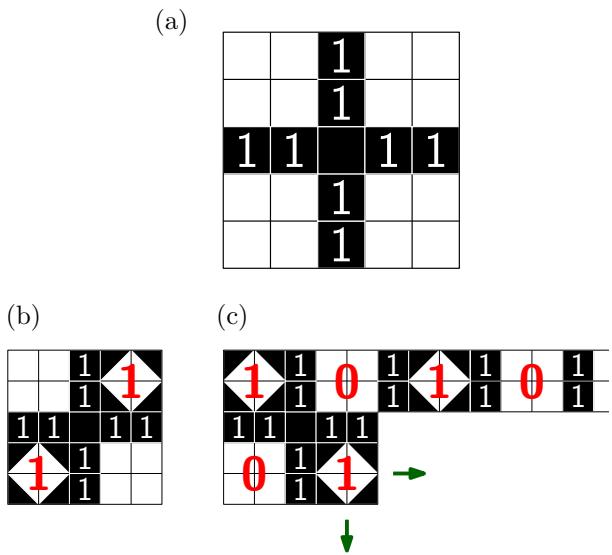


Figure 3: Variable gadget

“wire” pattern propagates a signal using the parity in two different ways. Using the terminology of [2], we need the gadgets of “variable,” “split,” “corner,” and “clause.” We describe these gadgets one by one.

Variable gadget: Figure 3(a)¹ shows the variable gadget. It is easy to see that we have two ways to fill the pattern as in Figure 3(b–c). It can propagate its value by the wire gadget as in the figure. It is also easy to obtain the negation of the variable by taking the value at the appropriate position of the wire.

Split gadget/corner gadget: Figure 4 shows the split and corner gadgets. Using the split gadget, we can increase the degree of the output of a variable gadget.

¹Hereafter, each pattern is assumed to be surrounded by black squares.

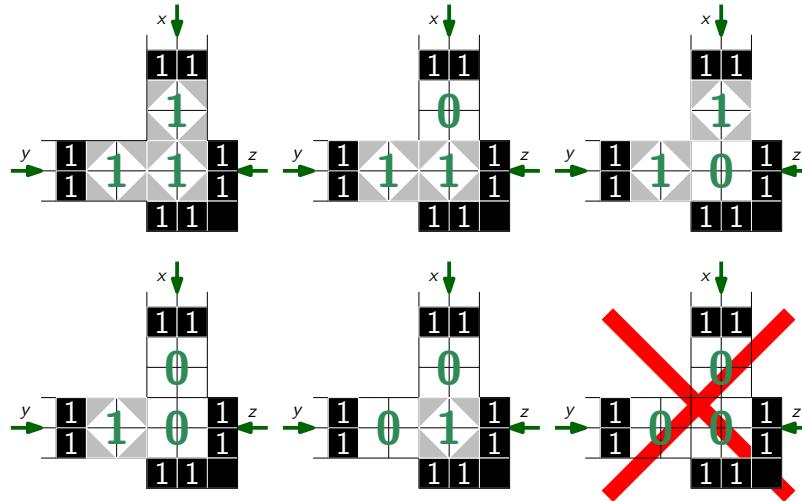


Figure 6: Feasible cases and infeasible case of a clause gadget

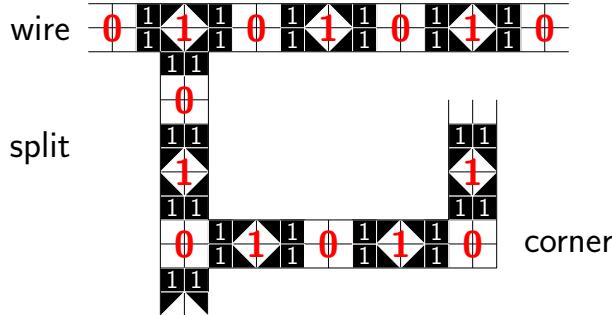


Figure 4: Split and corner gadgets

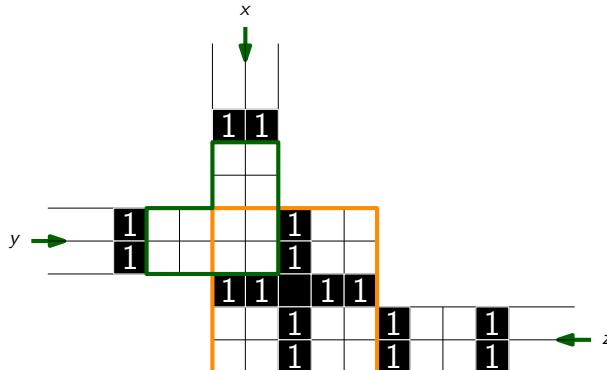


Figure 5: Clause gadget

Clause gadget: Figure 5 shows the clause gadget for a clause $C = \{x, y, z\}$. According to the values of x, y, z , we have eight possible cases. Among them, only the case $x = y = z = 0$ violates the condition of Shakashaka (Figure 6).

The gadgets for wire, variable, split, and corner are aligned properly because they are designed to fit into

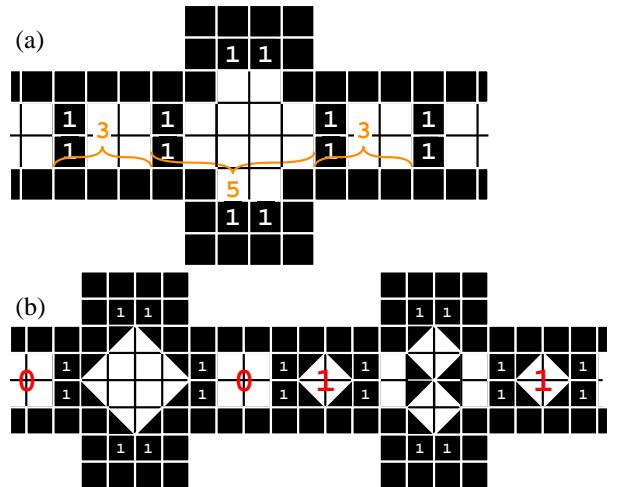


Figure 7: Parity gadget

a 3×3 square tiling. However, at a clause gadget, we have to change the positions of wires to fit the gadget. To shift the position, we use a “parity” gadget shown in Figure 7(a). Joining copies of the gadget in a straightforward way, we can change the position of a wire arbitrarily (Figure 7(b)). An example of a construction of Shakashaka for the instance $f = C_1 \vee C_2$, where $C_1 = \{x, \bar{y}, w\}$ and $C_2 = \{y, \bar{z}, \bar{w}\}$ is depicted in Figure 8.

It is easy to see that the resulting Shakashaka has a solution if and only if the original formula F is satisfiable. It is clear that the reduction can be done in polynomial time, and Shakashaka is in the class NP. Therefore, Shakashaka is NP-complete.

Our reduction is parsimonious, i.e., it preserves the number of solutions. That is, the number of satisfying assignments to the original CNF formula is equal

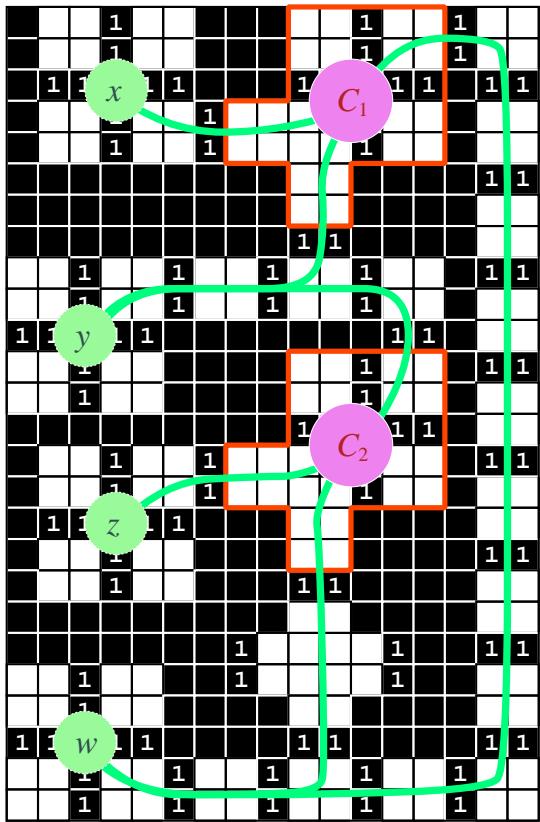


Figure 8: An example for $C_1 = \{x, \bar{y}, w\}$ and $C_2 = \{y, \bar{z}, \bar{w}\}$

to the number of solutions to the resulting instance of Shakashaka. Because the counting version of planar 3SAT is $\#P$ -complete [10], we have the following corollary:

Corollary 2 *The counting version of Shakashaka is $\#P$ -complete.*

4 Integer Programming Formulation

We formulate Shakashaka in terms of a 0-1 integer program. Recall that an instance I of Shakashaka consists of a rectangular board of size $m \times n$. We identify each square by $(i, j) \in \{1, 2, \dots, m\} \times \{1, 2, \dots, n\}$ in the natural way.

Variables: For each white square (i, j) , we will use five 0-1 variables $x[i, j, \square]$, $x[i, j, \blacksquare]$, $x[i, j, \blacksquare]$, $x[i, j, \blacksquare]$, and $x[i, j, \blacksquare]$. Exactly one of these variables has value 1, and the rest are 0, according to the following meaning:

$$\left\{ \begin{array}{ll} x[i, j, \square] = 1 & \text{means that } (i, j) \text{ remains white,} \\ x[i, j, \blacksquare] = 1 & \text{means that } (i, j) \text{ is filled with } \blacksquare, \\ x[i, j, \blacksquare] = 1 & \text{means that } (i, j) \text{ is filled with } \blacksquare, \\ x[i, j, \blacksquare] = 1 & \text{means that } (i, j) \text{ is filled with } \blacksquare, \\ x[i, j, \blacksquare] = 1 & \text{means that } (i, j) \text{ is filled with } \blacksquare. \end{array} \right.$$

We construct a linear system $S(I)$ with the variables $x[i, j, *]$ such that the solutions of the instance I of Shakashaka are in bijection with the solutions of $S(I)$. To this end, we set up five types of linear constraints as described below.

Constraint A (at most one triangle in each white square): In a solution to I , each white square either remains white, or is filled with one of the four black isosceles right triangles. We map this condition to the following linear equality:

$$\begin{aligned} x[i, j, \square] + x[i, j, \blacksquare] + x[i, j, \blacksquare] \\ + x[i, j, \blacksquare] + x[i, j, \blacksquare] = 1 \end{aligned} \quad (1)$$

for each i and j where (i, j) is a white square.

Proposition 3 *Let $S_A(I)$ be the linear system that consists of Constraint A. Then any feasible solution of $S_A(I)$ gives the mapping from each white square to exactly one of \square , \blacksquare , \blacksquare , \blacksquare , or \blacksquare .*

Constraint B (neighbors of black squares): Next we look at the black squares (i, j) . First we consider the case that (i, j) contains no number. In this case, (i, j) gives some restrictions to its white neighbors. For example, suppose that $(i-1, j)$ is white. Then, if $(i-1, j)$ is \blacksquare or \blacksquare , these two squares make a 45° white corner between them. Thus $(i-1, j)$ must be \blacksquare , \blacksquare , or \square . Hence, in this case, the equation (1) for $(i-1, j)$ can be replaced by

$$x[i-1, j, \square] + x[i-1, j, \blacksquare] + x[i-1, j, \blacksquare] = 1 \quad (2)$$

and we can fix $x[i-1, j, \blacksquare] = x[i-1, j, \blacksquare] = 0$.

On the other hand, when a black square (i, j) has a number k , it must have k b/w squares as its neighbor. This restriction is described by the following equation:

$$\begin{aligned} x[i-1, j, \blacksquare] + x[i-1, j, \blacksquare] + x[i+1, j, \blacksquare] \\ + x[i+1, j, \blacksquare] + x[i, j-1, \blacksquare] + x[i, j-1, \blacksquare] \\ + x[i, j+1, \blacksquare] + x[i, j+1, \blacksquare] = k, \end{aligned} \quad (3)$$

where $x[i, j, *]$ is regarded as 0 if (i, j) is black. We also fix $x[i-1, j, \blacksquare] = x[i-1, j, \blacksquare] = x[i+1, j, \blacksquare] = x[i+1, j, \blacksquare] = x[i, j-1, \blacksquare] = x[i, j-1, \blacksquare] = x[i, j+1, \blacksquare] = x[i, j+1, \blacksquare] = 0$ to avoid the 45° white angle.

Constraint C (sequences of triangles): Next we turn to the restrictions to make each connected white area a rectangle. Suppose $x[i, j, \blacksquare] = 1$. In this case, the white triangle at (i, j) can be orthogonal if and only if either $x[i, j+1, \blacksquare] = 1$ or $x[i+1, j+1, \blacksquare] = 1$. Therefore, we obtain the following constraint:

$$x[i, j, \blacksquare] \leq x[i, j+1, \blacksquare] + x[i+1, j+1, \blacksquare]. \quad (4)$$

Moreover, when $x[i, j, \blacksquare] = x[i+1, j+1, \blacksquare] = 1$, $(i, j+1)$ must remain white, or $x[i, j+1, \blacksquare] = 1$. (When $(i, j+1)$ is \blacksquare , we have a parity problem; we cannot enclose this area by extending this pattern. The other cases are also prohibited.) This implies the following constraint:

$$x[i, j, \blacksquare] + x[i+1, j+1, \blacksquare] \leq x[i, j+1, \blacksquare] + 1. \quad (5)$$

We add the similar constraints for the other directions. Then, we have the following proposition:

Proposition 4 Let $S_C(I)$ be the linear system that consists of Constraints A, B, and C, and fix any feasible solution of $S(I)$. Then, each angle on the boundary of each connected white area given by the mapping is 90° .

Constraint D (exclusion of concave corners): By Proposition 4, any feasible solution to Constraints A, B, and C produces a pattern consisting of orthogonal white polygons. However, this does not yet exclude concave corners. By Equation 4, no b/w square forms a part of a concave corner. Thus, a concave corner may be produced by only white squares. Suppose that $x[i, j, \blacksquare] = x[i+1, j, \blacksquare] = x[i, j+1, \blacksquare] = 1$. Then, $(i+1, j+1)$ must be \blacksquare or must remain white. Thus we add the following constraints (for all possible directions):

$$\begin{aligned} &x[i, j, \blacksquare] + x[i+1, j, \blacksquare] + x[i, j+1, \blacksquare] \\ &\leq x[i+1, j+1, \blacksquare] + x[i+1, j+1, \blacksquare] + 2. \end{aligned} \quad (6)$$

We now have the following proposition:

Proposition 5 Let $S_D(I)$ be the linear system that consists of Constraints A, B, C, and D, and fix any feasible solution of $S(I)$. Then every connected part of a boundary of a white area is a convex orthogonal polygon, i.e., a rectangle.

Constraint E (Exclusion of Nested White Rectangles): The last problem is that the linear system so far may produce nested rectangles. (Two rectangles are *nested* if one properly contains another.) We suppose that both of (i, j) and $(i+k, j+k)$ are \blacksquare . Then, to avoid nesting, we must have \blacksquare between them. That is, we must have \blacksquare at $(i+k', j+k')$ for some $0 < k' < k$. And it is not difficult to see that this is a necessary and sufficient condition to avoid nested rectangles. This observation gives us the following constraint:

$$\begin{aligned} &x[i, j, \blacksquare] + x[i+k, j+k, \blacksquare] \\ &\leq \sum_{0 < k' < k} x[i+k', j+k', \blacksquare] + 1. \end{aligned} \quad (7)$$

Combining all propositions and observations above, we conclude the following:

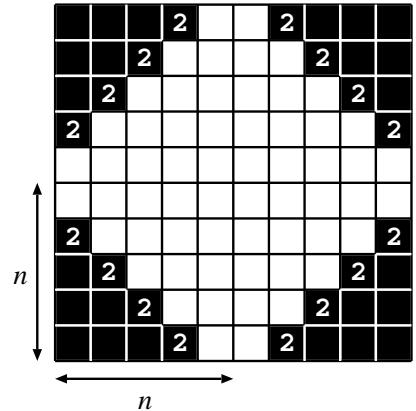


Figure 9: An artificial example of the puzzle Shakashaka.

Theorem 6 Let I be an instance of Shakashaka, and $S(I)$ be the linear system that consists of Constraints A–E. Then, a feasible solution of $S(I)$ gives a solution to I , and vice versa.

5 Experimental Results

In this section, we describe our experimental results. The IP solver we used is SCIP 3.0.0 [11]² (Binary: Windows/PC, 32bit, cl 16, intel 12.1: statically linked to SoPlex 1.7.0, Ipopt 3.10.2, CppAD 20120101.3). The machine we used was a laptop (Intel Core2 Duo P8600@2.40GHz with RAM 4GB on Windows Vista Business SP2). Each of the ten instances at nikoli.com³ was solved in less than one second in our experiments (Table 1).

We also looked at another instance at nikoli.com, which was prepared for a competition. The board has size 31×45 , the level is Extreme, and the number of white squares is 1230. A solution was obtained in 2.63 seconds.

The other examples are artificial ones (see Figure 9); for each $n = 1, 2, \dots$, the board of size $2n \times 2n$ consists of $4 \times \sum_{i=1}^{n-1} i = 2n(n-1)$ black squares, and $4 \times (n-1)$ black squares contain the number 2 as shown in the figure. Each of them has a unique solution. The experimental results for the artificial ones for $n = 2, 3, \dots, 40$ are shown in Figure 10. For $n = 40$, the solution is obtained in 19.86 seconds. A simple regression shows that the computation time is roughly proportional to 1.18^n .

6 Concluding Remarks

In this paper, we proved that Shakashaka is NP-complete. In our reduction, the black squares contain

²<http://scip.zib.de/>

³<http://www.nikoli.com/ja/puzzles/shakashaka/>

Problem	Size	Level	# of white squares	Time (sec)
1	10×10	Easy	76	0.02
2	10×10	Easy	77	0.03
3	10×10	Easy	82	0.03
4	10×18	Easy	131	0.07
5	10×18	Medium	156	0.09
6	10×18	Medium	144	0.07
7	14×24	Medium	297	0.21
8	14×24	Hard	295	0.19
9	20×36	Hard	645	0.84
10	20×36	Hard	632	0.91

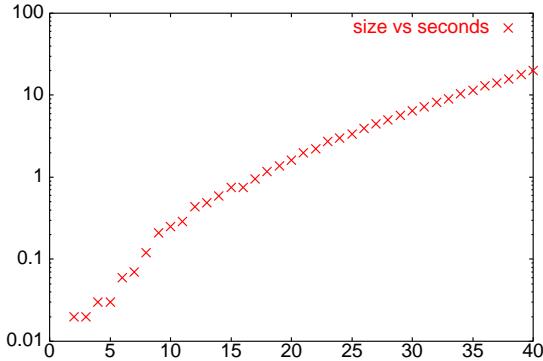
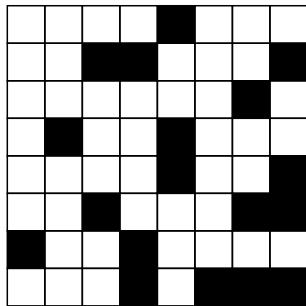
Table 1: Experimental results for the instances at nikoli.comFigure 10: Seconds for the artificial examples ($n = 2, 3, \dots, 40$).

Figure 11: An instance of Shakashaka without numbers

only the number 1 (or remain blank). An interesting question is to determine the computational complexity of Shakashaka with no numbers in the black squares. Figure 11 shows a nontrivial example, which has a unique solution. There are two natural questions in this Shakashaka puzzle. How many black squares are required to have a unique solution in an $m \times n$ board? Can this restricted Shakashaka be solved in polynomial time?

References

- [1] Nikoli, Shakashaka 1, vol.151, Pencil and Paper Puzzle Series, Nikoli, Jan. 2012.
- [2] R. A. Hearn and E. D. Demaine, Games, Puzzles, and Computation, A K Peters Ltd., 2009.
- [3] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. D. Mittelmann, T. K. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter, “MILP LIB 2010,” *Math. Program. Comput.*, vol.3, no.2, pp. 103–163, 2011.
- [4] A. Bartlett, T. P. Chartier, A. N. Langville, and T. D. Rankin, “Integer Programming Model for the Sudoku Problem,” *J. of Online Mathematics and its Applications*, vol.8, Article ID 1798, 2008.
- [5] R. A. Bosch, “Painting by Numbers,” *Optima*, vol.65, pp.16–17, 2001.
- [6] M. J. Chlond, “Classroom Exercises in IP Modeling: Su Doku and the Log Pile,” *INFORMS Transactions on Education*, vol.5, pp.77–79, 2005.
- [7] W. J. M. Meuffles and D. den Hertog, “Puzzle—Solving the *Battleship* Puzzle as an Integer Programming Problem,” *INFORMS Transactions on Education*, vol.10, no.3, pp.156–162, 2010.
- [8] L. Mingote and F. Azevedo, “Colored Nonograms: An Integer Linear Programming Approach,” *Proceedings of EPIA 2009 LNAI* vol. 5816, pp.213–224, Springer-Verlag 2009.
- [9] D. Lichtenstein, “Planar Formulae and Their Uses,” *SIAM J. on Computing*, vol.11, no.2, pp.329–343, 1982.
- [10] N. Creignou and M. Hermann, “Complexity of generalized satisfiability counting problems,” *Information and Computation*, vol.125, pp.1–12, 1996.
- [11] T. Achterberg, “SCIP: Solving Constraint Integer Programs,” *Mathematical Programming Computation*, vol.1, pp.1–41, 2009.

One-Round Discrete Voronoi Game in \mathbb{R}^2 in Presence of Existing Facilities

Aritra Banik*

Bhaswar B. Bhattacharya†

Sandip Das‡

Satyaki Mukherjee§

Abstract

In this paper we consider a simplified variant of the discrete Voronoi Game in \mathbb{R}^2 , which is also of independent interest in competitive facility location. The game consists of two players P1 and P2, and a finite set U of users in the plane. The players have already placed two sets of facilities F and S , respectively in the plane. The game begins by P1 placing a new facility followed by P2 placing another facility, and the objective of both the players is to maximize their own total payoffs. When $|F| = |S| = m$, this corresponds to the last round of the $(m + 1)$ -round discrete Voronoi Game in \mathbb{R}^2 . In this paper we propose polynomial time algorithms for obtaining optimal strategies of both the players under arbitrary locations of the existing facilities F and S . We show that the optimal strategy of P2, given any placement of P1, can be found in $O(n^2)$ time, and the optimal strategy of P1 can be found in $O(n^8)$ time.

1 Introduction

The main objective in any facility location problem is to judiciously place a set of facilities serving a set of users such that certain optimality criteria are satisfied. Facilities and users are generally modeled as points in the plane. The set of users (demands) is either *discrete*, consisting of finitely many points, or *continuous*, that is, a region where every point is considered to be a user. We assume that the facilities are equally equipped in all respects, and a user always avails the service from its nearest facility. Consequently, each facility has its *service zone*, consisting of the set of users that are served by it. For a set U of users, finite or infinite, and a set F of facilities, define for every $f \in F$, $U(f, F)$ as the set of users in U that are served by the facility f . In such a scenario, when the users choose the facilities based on the nearest-neighbor rule, the optimization criteria is to maximize the cardinality or the area of the service zone depending on whether the demand region is discrete or continuous, respectively.

*Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India, aritrabanik@gmail.com

†Department of Statistics, Stanford University, USA, bhaswar.bhattacharya@gmail.com

‡Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India, sandipdas@isical.ac.in

§Indian Statistical Institute, Bangalore, India, mail.satyaki.mukherjee@gmail.com

The game-theoretic analogue of such competitive problems for continuous demand regions is a situation where two players place two disjoint sets of facilities in the demand region. A player p is said to own a part of the demand region that is closer to the facilities owned by p than to the other player, and the player which finally owns the larger area is the winner of the game. The area a player owns at the end of the game is called the payoff of the player. In the *one-round game* the first player places m facilities following which the second player places another m facilities in the demand region. In the *m -round game* the two players place one facility each alternately for m rounds in the demand region.

Ahn et al. [1] studied a one-dimensional Voronoi Game, where the demand region is a line segment. They showed that when the game takes m rounds, the second player always has a winning strategy that guarantees a payoff of $1/2 + \varepsilon$, with $\varepsilon > 0$. However, the first player can force ε to be arbitrarily small. On the other hand, in the one-round game with m facilities, the first player always has a winning strategy. The one-round Voronoi Game in \mathbb{R}^2 was studied by Cheong et al. [7], for a square-shaped demand region. They proved that for any placement W of the first player, with $|W| = m$, there is a placement B of the second player $|B| = m$ such that the payoff of the second player is at least $1/2 + \alpha$, where $\alpha > 0$ is an absolute constant and m large enough. Fekete and Meijer [9] studied the two-dimensional one-round game played on a rectangular demand region with aspect ratio ρ . The Voronoi Game, for which the underlying space is a graph, was considered by Bandyapadhyay et al. [3].

In the discrete regime, the possible demand set is generally modeled as a finite graph, and users and facilities are restricted to lie on the nodes of the graph. As before, the players alternately chose nodes (facilities) from the graph, and all vertices (customers) are then assigned to closest facilities based on the graph distance. The payoff of a player is the number of customers assigned to it. Dürr and Thang [8] showed that deciding the existence of a Nash equilibrium for a given graph is NP-hard. Recently, Teramoto et al. [13] studied the same problem and considered following very restricted case: the game arena is an arbitrary graph, the first player occupies just one vertex which is predetermined, and the second player occupies m vertices in any way. They

proved that in this strongly restricted discrete Voronoi Game it is NP-hard to decide whether the second player has a winning strategy. They also proved that for a given graph G and the number r of rounds determining whether the first player has a winning strategy on G is PSPACE-complete. The discrete Voronoi Game for path graphs was studied by Kiyomi et al. [10].

Recently, Banik et al. [4] considered the discrete Voronoi Game where the universe is modeled as \mathbb{R} , and the distance between the users and the facilities are measured by their Euclidean distance. The problem consists of a finite user set $U \subset \mathbb{R}$, with $|U| = n$, and two players Player 1 (P1) and Player 2 (P2) each having $m = O(1)$ facilities. At first, P1 chooses a set $\mathcal{F}_1 \subset \mathbb{R}$ of k facilities following which P2 chooses another set $\mathcal{F}_2 \subset \mathbb{R}$ of m facilities, disjoint from \mathcal{F}_1 . The payoff of P2 is defined as the cardinality of the set of points in U which are closer to a facility owned by P2 than to every facility owned by P1. The payoff of P1 is the number of users in U minus the payoff of P2. The objective of both the players is to maximize their respective payoffs. The authors showed that if the sorted order of the points in U along the line is known, then the optimal strategy of P2, given any placement of facilities by P1, can be computed in $O(n)$ time. Also, for $m \geq 2$ the optimal strategy of P1 can be computed in $O(n^{m-\lambda_m})$ time, where $0 < \lambda_m < 1$, is a constant depending only on m . The discrete Voronoi Game for polygonal domains were considered by Banik et al. [5].

The discrete Voronoi Game when the user set consists of a finite set of points in \mathbb{R}^2 poses a major challenge. To the best of our knowledge, this problem has never been addressed before, and answering rather simple questions about this game is rather difficult. In this paper we consider a simplified variant of the discrete Voronoi Game in \mathbb{R}^2 , which is also of independent interest in competitive facility location. The game consists of two players P1 and P2 and a finite set U of users in the plane. Moreover, the two players have already placed a set of facilities F and S , respectively, in the plane. The game begins by P1 placing a new facility followed by P2 placing another facility. The objective of both the players is to maximize their respective payoffs.

For any placement of facilities A by P1 and B by P2, the payoff of P2, $\mathcal{P}_2(A, B)$ is defined as the cardinality of the set of points in U which are closer to a facility owned by P2 than to every facility owned by P1, that is, $\mathcal{P}_2(A, B) = |\bigcup_{f \in B} U(f, A \cup B)|$. Similarly, the payoff of P1, $\mathcal{P}_1(A, B)$ is $|\bigcup_{f \in A} U(f, A \cup B)|$, $|U \setminus \mathcal{P}_2(A, B)|$. Now, the One Round Discrete Voronoi Game in \mathbb{R}^2 in Presence of Existing Facilities can be formally stated as follows:

One Round Discrete Voronoi Game in \mathbb{R}^2 in Presence of Existing Facilities: Given a set U of n users, and two sets of facilities F and S owned by two competing

players P1 and P2, respectively, at first P1 chooses a facility f_1 following which P2 chooses another facility f_2 such that

- (a) $\max_{f'_2 \in \mathbb{R}^2} |\mathcal{P}_2(F \cup \{f_1\}, S \cup \{f'_2\})|$ is attained at the point f_2 .
- (b) $\max_{f \in \mathbb{R}^2} \nu(f)$ is attained at the point f_1 , where $\nu(f) = n - \max_{f'_2 \in \mathbb{R}} |\mathcal{P}_2(F \cup \{f\}, S \cup \{f_2\})|$.

The quantity $\nu(f_1)$ is called the optimal payoff of P1 and f_1 is the optimal strategy of P1.

In this paper we develop algorithms for the optimal strategies of the two players in the above game. Hereafter, we shall refer to this version of the Voronoi Game as $G_n(F, S)$. Note that when $|F| = |S| = m$ the situation described in the $G_n(F, S)$ game is identical to the last round of the $(m + 1)$ -round discrete Voronoi Game in \mathbb{R}^2 . Therefore, this problem takes the first non-trivial step towards solving the discrete Voronoi Game problem in \mathbb{R}^2 . Moreover, as mentioned before, this problem is of independent interest in competitive facility location. In any growing economy the expansion of the service zone is of utmost importance. However, because of some implied constraint it is never possible to place all your facilities at once. So it is of utmost importance to find a strategy which will guide how to place a set of facilities in a sequential manner, as the market grows. The $G_n(F, S)$ game is an instance of such a problem. Imagine there are 2 competing companies are providing a service to a set of users in a city. Suppose both these companies already have their respective service centers located in different parts of the city. Now, if both of them wish to open a new service center with the individual goal to maximize their total payoff, then the problem is an instance of the Voronoi Game described above. Though the $G_n(F, S)$ game, as described above, has never been studied before, if both F and S are empty, then it is a well-known fact that optimal strategy of P1 in the $G_n(F, S)$ game is at the halfspace median of U [12], which can be computed in $O(n \log^3 n)$ time [11]. However when the sets F and S are non-empty the problem becomes immensely more complicated. In this paper we propose polynomial time algorithms for obtaining optimal strategies of both the players in the $G_n(F, S)$ game.

The optimal strategy of P2, given any placement of P1, is identical to the solution of the **MaxCov** problem studied by Cabello et al. [6]. Suppose we are given a set of users U , existing facilities F and S , and any placement of a new facility f by P1. Let $U_1 \subseteq U$ denote the subset of users that are served by P1, in presence of F , S , and f . For every point $u \in U_1$, consider the *nearest facility disk* C_u centered at u and passes through the facility in $F \cup \{f\}$ which is closest to u . Note that a new facility s placed by P2 will serve any user $u \in U_1$ if and only if $s \in C_u$. If $\mathcal{C} = \{C_u | u \in U_1\}$, the optimal

strategy for P2, given any placement f of P1, is to place the new facility at a point where maximum number of disks in \mathcal{C} overlap. This is the problem of finding the maximum depth in an arrangement of n disks, and can be computed in $O(n^2)$ time [2].

Therefore, the main challenge in the $G_n(F, S)$ game lies in finding the optimal strategy of P1. In this paper, we provide a complete characterization of the event points and obtain a polynomial time algorithm for obtaining an optimal placement of P1:

Theorem 1 *Given a set U of n users, two sets of facilities F and S owned by two competing players P1 and P2, respectively, the optimal strategy of P1 in the $G_n(F, S)$ game can be found in $O(n^8)$ time.*

2 Understanding the Optimal Strategy of P1

In the $G_n(F, S)$ game, we are given a set U of n users, two sets of facilities F and S owned by two competing players P1 and P2 respectively. Observe that the set of facilities F and S will divide the set of users U into two groups U_F and U_S where U_F is the set of users served by the facilities placed by P1 and U_S is the set of users served by the facilities placed by P2.

Let f be any new placement by P1. Denote the set of users served by f , by $U_{FS}(f)$. More formally,

$$U_{FS}(f) = \{u_i | d(u_i, f) < d(u_i, f'), \forall f' \in F \cup S\}$$

Further let the users served by the set of facilities F and S after placement of f be $U_{F \setminus f}$ and $U_{S \setminus f}$, that is,

$$U_{F \setminus f} = \bigcup_{f' \in F} U(f', F \cup S \cup \{f\})$$

and

$$U_{S \setminus f} = \bigcup_{f' \in S} U(f', F \cup S \cup \{f\}).$$

Hence, any facility f by P1 will divide the set of users into three disjoint sets $U_{FS}(f)$, $U_{F \setminus f}$ and $U_{S \setminus f}$. Now any new placement s by P2 can serve a subset of users from all these three sets. For any placement of facility s by P2, let $U_f(s) \subset U_{FS}(f)$ be the set of users such that for all $u_i \in U_f(s)$, $d(u_i, s) < d(u_i, f)$. Similarly define the set of users $U_{F \setminus f}(s) \subset U_{F \setminus f}$ such that for all $u_j \in U_{F \setminus f}(s)$, $d(u_j, s) < d(u_j, f_k)$ for all $f_k \in F$.

Observe that for any placement f and s by P1 and P2 respectively the payoff of P2 will be equal to

$$\mathcal{P}_2(F \cup \{f\}, S \cup \{s\}) = |U_{S \setminus f}| + |U_f(s)| + |U_{F \setminus f}(s)|$$

For any placement of facility f by P1 define the *effective depth* of f , $\delta(f)$ as

$$\delta(f) = |U_{S \setminus f}| + \max_{s \in \mathbb{R}^2} (|U_f(s)| + |U_{F \setminus f}(s)|)$$

The optimal strategy of P1 is to find a point f_1 such that $\delta(f_1) = \arg \min_{f \in \mathbb{R}^2} \delta(f)$, that is the point of minimum effective depth. In order to do that we will subdivide \mathbb{R}^2 into a polynomial many cells such that the effective depth of all points in each cell is the same.

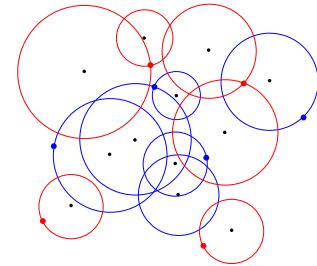


Figure 1: Arrangement of the set of circles \mathcal{C}_{FS}

Consider the set of circles \mathcal{C}_{FS} where each circle $C \in \mathcal{C}_{FS}$ is centered at some user u_i and passing through the facility closest to u_i among the set of facilities $F \cup S$ (see Figure 1 where facilities placed by P1 are shown in red and the facilities placed by P2 shown in blue). Denote the arrangement of the set of circles \mathcal{C}_{FS} by $\mathcal{A}(\mathcal{C}_{FS})$. We also include the lines joining any pair of users into \mathcal{C}_{FS} .

For any placement of facility x by P1 and for any user u_i , let $C_i(x)$ be the circle centered on u_i and passing through the facility closest to u_i from the set of facilities $F \cup S \cup \{x\}$. Consider all such circles $\mathcal{C}(x)$.

Let x and y be two points that belong to the same cell of $\mathcal{A}(\mathcal{C}_{FS})$ but $\delta(x) \neq \delta(y)$. Now as x and y belong to the same cell of $\mathcal{A}(\mathcal{C}_{FS})$ therefore $U_{S \setminus x} = U_{S \setminus y}$. That means for the placement of facilities x and y , $\max_{s \in \mathbb{R}^2} (|U_x(s)| + |U_{F \setminus x}(s)|) \neq \max_{s \in \mathbb{R}^2} (|U_y(s)| + |U_{F \setminus y}(s)|)$.

Now observe that for any placement of facility x , $\max_{s \in \mathbb{R}^2} (|U_x(s)| + |U_{F \setminus x}(s)|)$ denotes the maximum number of circles among the set of circles $\mathcal{C}(x)$ that can be pierced by a single point. Hence for each cell $\lambda \in \mathcal{A}(\mathcal{C}_{FS})$ if we can subdivide λ further such that in each sub-cell of λ for all points x maximum number of circles among the set of circles $\mathcal{C}(x)$ that can be pierced by a single point remains fixed, then we are done.

Lemma 2 *If x, y belong to some cell of $\mathcal{A}(\mathcal{C}_{FS})$ with $\delta(x) \neq \delta(y)$, then there exist three users $u_i, u_j, u_k \in U_{FS}(x) \cup U_{F \setminus x}$ such that $C_i(x) \cap C_j(x) \cap C_k(x) \neq \emptyset$ and $C_i(y) \cap C_j(y) \cap C_k(y) = \emptyset$ or vice versa.*

Proof. Without loss of generality assume $\delta(x) > \delta(y)$. For any placement of facility f by P1, U_f be the maximum cardinality set of users served by P2. As x and y belong to the same cell of $\mathcal{A}(\mathcal{C}_{FS})$, $U_x \not\subseteq U_{FS}(x)$. Also we can assume that the cardinality of U_x and U_y is at least three. We shall prove this result by contradiction. Suppose that for every three users $u_i, u_j, u_k \in$

$U_{FS}(x) \cup U_{F \setminus x}$, such that $C_i(x) \cap C_j(x) \cap C_k(x) \neq \emptyset$ we also have $C_i(y) \cap C_j(y) \cap C_k(y) \neq \emptyset$.

Therefore, for any three users $u_i, u_j, u_k \in U_x$, $C_i(x) \cap C_j(x) \cap C_k(x) \neq \emptyset$. By assumption, this implies $C_i(y) \cap C_j(y) \cap C_k(y) \neq \emptyset$. Therefore, by Helly's theorem, $\bigcap_{u_i \in U_x} C_i(y) \neq \emptyset$, which means that the number of users that can be served by P2 by placing one facility from the set of users $U_{FS}(y) \cup U_{F \setminus y}$ is at least $|U_x|$. Therefore, $\delta(y) \geq |U_x| + |U_{S \setminus y}| = |U_x| + |U_{S \setminus x}| = \delta(x)$, which is a contradiction and the result holds. \square

In light of lemma 2 we define, for each triplet of users $u_i, u_j, u_k \in U$, and any placement of facility $x \in \mathbb{R}^2$ by P1, the indicator variable,

$$\beta_{ijk}(x) = \begin{cases} 1 & \text{if } C_i(x) \cap C_j(x) \cap C_k(x) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Let $\beta(x)$ be the 3-dimensional array with cardinality $|U| \times |U| \times |U|$ where each cell $\beta_{ijk}(x)$ is defined as above. From Lemma 2 and the above definition, the following observation is immediate.

Observation 1 *If x, y belong to the same cell of $\mathcal{A}(C_{FS})$ and the two arrays $\beta(x)$ and $\beta(y)$ are equal in every coordinate, then $\delta(x) = \delta(y)$.*

Our next goal is to tessellate $\mathcal{A}(C_{FS})$ into a finer set of cells such that for any two points x and y on the same cell $\beta(x) = \beta(y)$. This implies that for any three users $u_i, u_j, u_k \in U$, either $C_i(x) \cap C_j(x) \cap C_k(x) \neq \emptyset$ or $C_i(x) \cap C_j(x) \cap C_k(x) = \emptyset$, for all points x in a fixed finer cell of the tessellation. Observation 1 would then imply that for all points in a cell the effective depth remains constant. Hence, by checking each cell once we can find the point with minimum effective depth.

Therefore, for each cell of $\mathcal{A}(C_{FS})$, we want a further subdivision such that for every point x in the fixed subdivided cell $C_i(x) \cap C_j(x) \cap C_k(x) \neq \emptyset$, for every triplet of users $u_i, u_j, u_k \in U$. Note that for any placement x by P1 and a user $u \in U \setminus U_{S \setminus x}$, either u is served by x or by some existing facility in F . The following definition distinguishes these two cases:

Definition 2.1 *Given any placement x by P1 and a user $u \in U \setminus U_{S \setminus x}$, the circle $C(x)$ is called an old circle if it is centered at u and passes through some facility $f_j \in F$, where f_j be the facility closest to u among the set of facilities $F \cup \{x\}$, that is, $u \in U_{F \setminus x}$. The circle $C(x)$ is called a new circle if it is centered at u and passes through x , that is $u \in U_{FS}(x)$.*

For every three fixed users $u_i, u_j, u_k \in U$ and a fixed point $x \in \mathbb{R}^2$, denote by $N_{ijk}(x)$ the subset of the circles in $\{C_i(x), C_j(x), C_k(x)\}$, which are new. For $S \subseteq \{u_i, u_j, u_k\}$, define the following sets:

$$\Gamma_{ijk}(S) = \{x \in \mathbb{R}^2 : C_i(x) \cap C_j(x) \cap C_k(x) = \emptyset \text{ and } C_a(x) \in N_{ijk}(x) \text{ if } u_a \in S\}$$

Moreover, for $z \in \{0, 1, 2, 3\}$, let $\Gamma_{ijk}^z = \bigcup_{S:|S|=z} \Gamma_{ijk}(S)$, where the union is taken over all sets $S \subseteq \{u_i, u_j, u_k\}$ such that $|S| = z$.

Lemma 3 *Let D_a be the circle centered at u_a passing through the facility in $F \cup S$ closest to u_a , for $a \in 1, 2, \dots, n$. Then for three fixed users $u_i, u_j, u_k \in U$, we have*

- (a) $\Gamma_{ijk}(\emptyset) = (D_i \cup D_j \cup D_k)^c$.
- (b) *For $S = \{u_i, u_j, u_k\}$, $\Gamma_{ijk}(S)$ is the interior of the triangle formed by u_i, u_j and u_k .*
- (c) *For $S = \{u_k\}$, $\Gamma_{ijk}(S)$ is the interior of the circle centered at u_k and passing through the point in $D_i \cap D_j$ closest to u_k .*

Proof. It is easy to show (a) and (b) from the definitions.

For proving (c), let p_c be the point in $D_i \cap D_j$ which is closer to u_k . Hence, for all points p in the open disk D , centered at u_k and passing through p_c , $C_i(x) \cap C_j(x) \cap D = \emptyset$, and for all points p outside the open disk D , $p_c \in C_i(x) \cap C_j(x) \cap D$. Therefore, if $S = \{u_k\}$, then $\Gamma_{ijk}(S) = D$. \square

As it turns out, when S consists of 2 elements then the sets $\Gamma_{ijk}(S)$ has a complicated geometric structure. The following lemma provides a complete characterization of the set $\Gamma_{ijk}(S)$ where $|S| = 2$.

Lemma 4 *For $S = \{u_i, u_j\}$, $\Gamma_{ijk}(S)$ is an open set bounded by $O(1)$ circular arcs and line segments. As a consequence, the boundary of $\Gamma_{ijk}(S)$ can be computed in constant time.*

We prove this lemma in the next section. Then using Lemma 3 and Lemma 4 we show how the proof of Theorem 1 can be completed.

3 Proof of Lemma 4 and Theorem 1

In this section we prove Lemma 4. The proof is rather technical, and requires careful analysis of the geometry of the points. Using this lemma, we then complete the proof of Theorem 1.

3.1 Proof of Lemma 4

In this section we will characterize $\Gamma_{ijk}(S)$ for $S = \{u_i, u_j\}$ and will complete the proof of Lemma 4. Hence given three users, say u_i, u_j and u_k we want to characterize the set of points $\Gamma_{ijk}(S)$ for $S = \{u_i, u_j\}$ such that for any point x in $\Gamma_{ijk}(S)$ if P1 places a facility at x , two circles $C_i(x)$ and $C_j(x)$ will pass through x and $C_k(x)$ will pass through some other facility and

$C_i(x) \cap C_j(x) \cap C_k(x) = \emptyset$. For notational simplicity, throughout this section we shall denote the region $\Gamma_{ijk}(S)$ as X and $C_k(x)$ as C . It can be shown that X is bounded and open. In the following lemma, we characterize the boundary of X . The proof is involves standard geometric arguments, and can be found in the full version of the paper.

Lemma 5 *Any point p belongs to the boundary of X , $\partial(X)$ if and only if $C_i(p) \cap C_j(p) \cap C$ is a singleton set.*

□

From Lemma 5 we know that $\partial(X)$ is the set of points p such that $C_i(p) \cap C_j(p) \cap C$ is a singleton set. Next we will find all such points. Observe that there can be two cases.

Case 1: Suppose the intersection of any two of the three circles, $C_i(p), C_j(p)$ and C is single point, say x , and the third circle contains x . Now suppose we want to find the set of points p such that $C_i(p) \cap C$ is a single point and $C_j(p)$ contains that point. Let the point closest to u_i in C be p_i . Now the set of points for which $C_i(p) \cap C$ is a single point is the circle $C_i(p_i)$ and for all point $p \in C_i(p_i)$, $C_i(p) \cap C = \{p_i\}$. But the circle $C_j(p)$ must contain p_i . Hence the set of points p such that $C_i(p) \cap C$ is a single point and $C_j(p)$ contains that point, is $C_i(p_i) \setminus C_j(p_i)$ (see Figure 2). Similarly we can find the points p such that $C_j(p) \cap C$ is a single point and $C_i(p)$ contains that point. Set of points p for which $C_i(p) \cap C_j(p)$ is a single point is the set of points on the line segment joining u_i and u_j , $[u_i, u_j]$. But C must contain the point p . Hence the set of points for which $C_i(p) \cap C_j(p)$ is a single point and C contains that point, is equal to $[u_i, u_j] \cap C$ (see Figure 3).

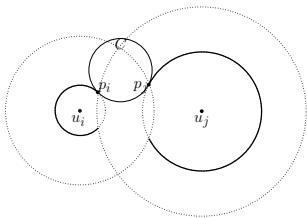


Figure 2: $\partial(X)$ when $C_i(p) \cap C$ is a single point

Case 2: The intersection of any two of the three circles is not a singleton, but $C_i(p) \cap C_j(p) \cap C$ is a singleton set. For any point p , define p_r to be the reflection of p on the line joining u_i and u_j . Observe that for any point p , the circle $C_i(p)$ and $C_j(p)$ intersects at points p and p_r . Hence, if for any point p , $C_i(p) \cap C_j(p) \cap C$ is a singleton, then $C_i(p_r) \cap C_j(p_r) \cap C$ is also a singleton. Now, we want

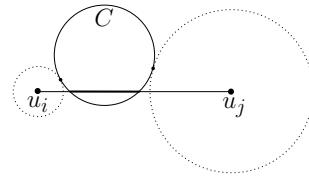


Figure 3: $\partial(X)$ when $C_i(p) \cap C_j(p)$ is a single point and C contains that point

to find the set of points p such that none of the pairwise intersections of the three circles $C_i(p), C_j(p)$ and C are singletons, but $C_i(p) \cap C_j(p) \cap C$ is a singleton set. Observe that p or p_r must belong to the boundary of C . Without loss of generality we will find the set of points p on the boundary of C such that $C_i(p) \cap C_j(p) \cap C$ is a singleton set. Now consider the line ℓ_i , joining u_i and the center of C (see Figure 4). Observe that for any point p on the boundary of C , C and $C_i(p)$ will intersect at p and p^{-1} where p^{-1} is the reflection of p with respect to the line ℓ_i . Suppose that among p and p^{-1} , p^{-1} is closer to u_j . In that case $C_i(p) \cap C_j(p) \cap C$ is not a singleton because p^{-1} belongs to $C_i(p) \cap C_j(p) \cap C$. Hence all the points on one side of ℓ_i are not in $\partial(X)$. Similarly consider the line ℓ_j , joining u_j and the center of C . All the points in one side of ℓ_j is also not in $\partial(X)$. Remaining points are shown in bold in Figure 4. It can be also shown that any point p on the region shown in Figure 4 is in $\partial(X)$.

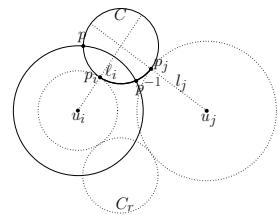


Figure 4: $\partial(X)$ when the intersections of none of the two circles are singletons, but $C_i(p) \cap C_j(p) \cap C$ is a singleton set.

This completes the proof of Lemma 4. The structure of X in the cases where the line joining u_i and u_j does not intersects C , and intersects C , are shown in Figure 5 and Figure 6, respectively.

3.2 Proof of Theorem 1

In this section using Lemma 3 and Lemma 4 we complete the proof of Theorem 1. Recall, from Section 2, the definition of Γ_{ijk}^z , for $S \subseteq \{u_i, u_j, u_k\}$, and $z \in \{0, 1, 2, 3\}$. We now define $\Gamma^z = \{\Gamma_{ijk}^z : u_i, u_j, u_k \in U\}$. Consider the tessellation of the plane induced by the

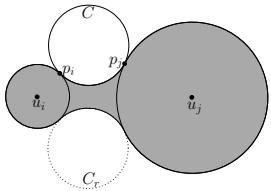


Figure 5: X when the line joining u_i and u_j does not intersect C .

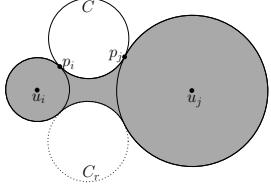


Figure 6: X when the line joining u_i and u_j intersects C .

collection of the sets Γ^z , for $z \in \{0, 1, 2, 3\}$ and \mathcal{C}_{FS} . From Lemma 3 we know that \mathcal{C}_{FS} and Γ^0 consists of $O(n)$ circles, and Γ^3 consists $O(n^2)$ lines (set of lines passing through each pair of users in U). Lemma 3 also shows that Γ^1 consists of $O(n^3)$ circles. From Lemma 4 we know for $S \subseteq \{u_i, u_j, u_k\}$, with $|S| = 2$, $\Gamma_{ijk}(S)$ is an open set bounded by $O(1)$ circular arcs and line segments. Therefore, Γ^2 also consists of $O(n^3)$ circles and line segments. Hence, the arrangement generated by Γ^z , for $z \in \{0, 1, 2, 3\}$ and \mathcal{C}_{FS} consists of $O(n^3)$ circles and line segments. The effective depth of any cell in this tessellation is constant. Moreover, the effective depth of a cell is the maximum depth in an arrangement of a set of $O(n)$ circles, which can be computed in $O(n^2)$ time [2]. Hence, by checking the effective depth of all the $O(n^6)$ cells, the minimum effective depth can be obtained in $O(n^8)$ time. Thus, the optimal strategy of P1 in the $G_n(F, S)$ game can be found in $O(n^8)$ time.

References

- [1] Hee-Kap Ahn, Siu-Wing Cheng, Otfried Cheong, Mordecai J. Golin, and René van Oostrum. Competitive facility location: the Voronoi game. *Theor. Comput. Sci.*, 310(1-3):457–467, 2004.
- [2] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- [3] Sayan Bandyapadhyay, Aritra Banik, Sandip Das, and Hirak Sarkar. Voronoi game on graphs. In Subir Kumar Ghosh and Takeshi Tokuyama, editors, *WALCOM*, volume 7748 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2013.
- [4] Aritra Banik, Bhaswar B. Bhattacharya, and Sandip Das. Optimal strategies for the one-round discrete Voronoi game on a line. *Journal of Combinatorial Optimization*, pages 1–15, 2013, to appear.
- [5] Aritra Banik, Sandip Das, Anil Maheshwari, and Michiel H. M. Smid. The discrete voronoi game in a simple polygon. In Ding-Zhu Du and Guochuan Zhang, editors, *COCOON*, volume 7936 of *Lecture Notes in Computer Science*, pages 197–207. Springer, 2013.
- [6] Sergio Cabello, José Miguel Díaz-Báñez, Stefan Langerman, Carlos Seara, and Inmaculada Ventura. Facility location problems in the plane based on reverse nearest neighbor queries. *European Journal of Operational Research*, 202(1):99–106, 2010.
- [7] Otfried Cheong, Nathan Linial, and Sariel Har-Peled. The one-round Voronoi game. In *Discrete Comput. Geom.*, pages 97–101, 2002.
- [8] Christoph Dürr and Nguyen Kim Thang. Nash equilibria in voronoi games on graphs. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 17–28. Springer, 2007.
- [9] Sándor P. Fekete and Henk Meijer. The one-round Voronoi game replayed. *Comput. Geom.*, 30(2):81–94, 2005.
- [10] Masashi Kiyomi, Toshiki Saitoh, and Ryuhei Uehara. Voronoi game on a path. *IEICE Transactions*, 94-D(6):1185–1189, 2011.
- [11] S. Langerman and W. Steiger. Optimization in arrangements. *Proc. 20th International Symposium on Theoretical Aspects of Computer Science*, pages 50–61, 2003.
- [12] J. Matoušek. Computing the center of planar point sets. *Computational Geometry: Papers from the DIMACS Special Year*, pages 221–230, 1991.
- [13] Sachio Teramoto, Erik D. Demaine, and Ryuhei Uehara. Voronoi game on graphs and its complexity. In Sushil J. Louis and Graham Kendall, editors, *CIG*, pages 265–271. IEEE, 2006.

Zipper Unfolding of Domes and Prismoids

Erik D. Demaine*

Martin L. Demaine*

Ryuhei Uehara†

Abstract

We study Hamiltonian unfolding—cutting a convex polyhedron along a Hamiltonian path of edges to unfold it without overlap—of two classes of polyhedra. Such unfoldings could be implemented by a single zipper, so they are also known as zipper edge unfoldings. First we consider domes, which are simple convex polyhedra. We find a family of domes whose graphs are Hamiltonian, yet any Hamiltonian unfolding causes overlap, making the domes Hamiltonian-ununfoldable. Second we turn to prismoids, which are another family of simple convex polyhedra. We show that any nested prismoid is Hamiltonian-unfoldable, and that for general prismoids, Hamiltonian unfoldability can be tested in polynomial time.

Keywords: edge unfolding, Hamiltonian-unfolding, zipper unfolding, paper folding, dome, prismoid.

1 Introduction

A common way to make a polyhedron from paper is to fold and glue a planar polygonal shape, called a *net* of the polyhedron. The characterization of polyhedra and their nets has been investigated since Dürer used nets to represent polyhedra in his 1525 book (see [DO07, O'R11]). One long-standing open problem is whether every convex polyhedron can be developed into a flat nonoverlapping polygonal shape by cutting only along its edges. Such a development is called an *edge unfolding* of the polyhedron. So far, very special classes of edge-unfoldable convex polyhedra are known: polyhedra of at most six vertices [DiB90], pyramids, prisms, prismoids, and domes [O'R01, DO07, O'R08].

In any edge unfolding, the cut edges produce a spanning tree of the graph representing the combinatorial structure of the convex polyhedron. One possible approach to the open problem is to restrict the cutting spanning tree to be a simple path. Because the path should visit (or cut) every vertex exactly once, the cutting edges produce a Hamiltonian path along the edges of the polyhedron. This restricted type of edge unfolding is called a *Hamiltonian unfolding* [DDL⁺10]. From an industrial point of view, such an unfolding can be re-



Figure 1: Zipper folding bags (Top: Spiral Wristlets (<http://www.cathayana.com/su13.htm>)). Bottom: ZipIt Monster (<http://www.zipitstore.com>))

alized by a zipper, and there are several products based on this idea (Figure 1).

From the graph-theoretical point of view, the Hamiltonian unfolding problem is related to the Hamiltonian path problem on a graph representing the vertices and the edges of the polyhedron. More precisely, if a polyhedron is Hamiltonian-unfoldable, then its corresponding graph must have a Hamiltonian path. Recently, Demaine et al. [DDL⁺10] found that all Archimedean solids are Hamiltonian-unfoldable. On the other hand, a rhombic dodecahedron does not have a Hamiltonian-unfolding because its corresponding graph has no Hamiltonian path [DDL⁺10]. As far as the authors know, all Hamiltonian-ununfoldable polyhedra have been proved in this combinatorial way, by showing that their corresponding graphs are not Hamiltonian.

On the other hand, the difficulty of edge unfolding convex polyhedron comes from the fact that we have

*Computer Science and Artificial Intelligence Laboratory, MIT, {edemaine, mdemaine}@mit.edu

†School of Information Science, JAIST, uehara@jaist.ac.jp

no general strategy to check whether its development causes an overlap no matter how it is cut along its edges. That is, to solve the open problem negatively, we have to find a convex polyhedron that causes an overlap by edge unfolding along any spanning tree. In this sense, a natural question arises: is there a convex polyhedron whose corresponding graph has a Hamiltonian path, yet any Hamiltonian unfolding causes overlap?

Our results. Our first result is an affirmative answer to the natural question. We show a family of convex polyhedra, which are simple domes, such that an overlap occurs in every Hamiltonian unfolding. Each of our domes has many Hamiltonian paths on its corresponding graph. Thus we can say that a graph-theoretic approach is not enough to tackle the open problem even for quite simple convex polyhedra.

Extending this result, for any fixed integer k , we show that there exists a family of domes that cannot be edge-unfolded by any cutting tree of degree at most k . That is, we show that, if the degree of the spanning tree of cuts is bounded, there exist infinitely many convex polyhedra that cannot be edge-unfolded. Hamiltonian unfoldings are the special case when the degree bound k is 2.

Next we turn to prismoids; a *prismoid* is the convex hull of two parallel convex polygons whose corresponding angles are equal. If one of these polygons contains the other in the projection orthogonal to the parallel planes, then the prismoid is *nested*. We give positive results about prismoids. First we show that any nested prismoid can be unfolded by a Hamiltonian unfolding. This result is based on band unfolding of nested prismoids developed in [ADL⁺08]. Second we show how to determine whether a general prismoid can be Hamiltonian-unfolded in polynomial time. This result is based on counting of the number of Hamiltonian paths of a general prismoid. We conjecture that any (general) prismoid can be Hamiltonian-unfolded, but this problem remains unsolved.

2 Hamiltonian-Ununfoldable Dome

For any integer $n \geq 3$, a *dome* is a convex polyhedron that consists of a (convex) polygonal *base*, and n (convex) polygonal *sides*, each of which shares a distinct edge with the base (see, e.g., [DO07]).

First we state a technical lemma.

Lemma 1 *For a positive integer n , let $\theta = \frac{2\pi}{n}$. Let T be an isosceles triangle with apex angle θ . Two arms of T are of unit length. We place eight copies of T as in Figure 2, where bold edges are shared by two triangles. Then the triangles T_4 and T_8 overlap for any $n > 12$.*

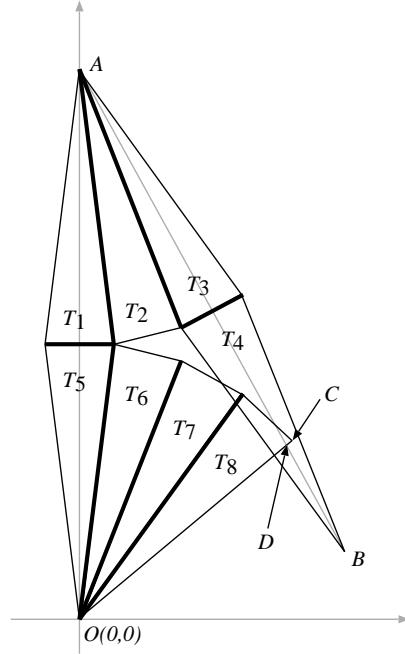


Figure 2: Overlapping triangles

Proof. We put the origin $O = (0, 0)$ on the apex of T_5 , and the y axis on the line joining the apices of T_1 and T_5 . Let A and B be the apices of T_1 and T_4 , respectively. Then we can compute

$$A = \left(0, 2 \sin \frac{\theta}{2}\right), B = \left(2 \sin \frac{\theta}{2} \sin 2\theta, 2 \sin \frac{\theta}{2}(1 - \cos 2\theta)\right).$$

On the other hand, let C be the furthest base angle point of T_8 from T_5 . Then we have

$$C = \left(\cos \frac{7\theta}{2}, \sin \frac{7\theta}{2}\right).$$

Now consider the intersection point D on two lines AB and OC . (Precisely, two lines containing AB and OC .) Then both T_4 and T_8 contain the point D if $|OD| < 1$. By a simple computation, we obtain

$$D = \left(\frac{2 \sin \frac{\theta}{2}}{\cot 2\theta + \cot \frac{7\theta}{2}}, \frac{2 \sin \frac{\theta}{2} \tan 2\theta}{\tan 2\theta + \tan \frac{7\theta}{2}}\right)$$

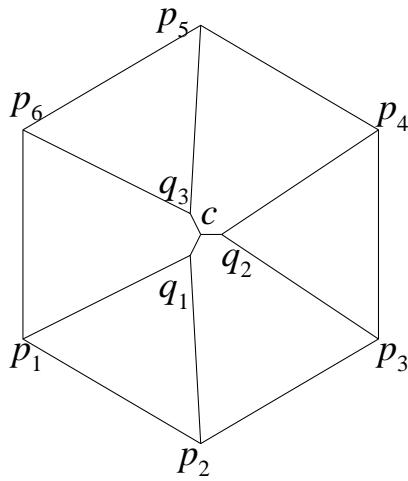
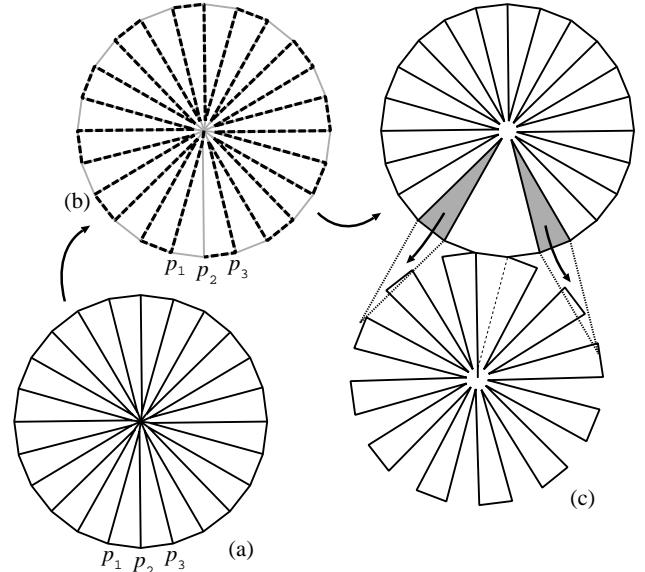
and hence $|OD|^2$ equals

$$4 \sin^2 \frac{\theta}{2} \left(\frac{1}{(\cot 2\theta + \cot \frac{7\theta}{2})^2} + \frac{\tan^2 2\theta}{(\tan 2\theta + \tan \frac{7\theta}{2})^2} \right),$$

which is less than 1 for any $n > 12$. \square

Theorem 2 *There exists an infinite sequence of domes that are Hamiltonian-ununfoldable.*

Proof. For each integer $n > 1$, we construct a dome $D(n)$ as follows. The base $B(n)$ is a regular $2n$ -gon.

Figure 3: The top view of $D(3)$ Figure 4: One possible development of $D(12)$

Let p_1, p_2, \dots, p_{2n} be the vertices of $B(n)$. The dome $D(n)$ has an apex c that is on the central perpendicular of $B(n)$. The height of c is very small. We put a small circle C centered at c , and put n points q_1, q_2, \dots, q_n on C such that these points form a regular n -gon. To simplify, we assume that the height of c and the radius of C are almost 0. Then we join and make edges $\{p_{2i-1}, q_i\}$ and $\{p_{2i}, q_i\}$ for each $i = 1, 2, \dots, n$. We rotate the circle C so that each triangle $q_i p_{2i-1} p_{2i}$ is an isosceles triangle. We also join c to the q_i for each $i = 1, 2, \dots, n$. Figure 3 shows the top view of the dome $D(n)$ for $n = 3$. Now we show that $D(n)$ is Hamiltonian-ununfoldable for $n > 12$.

Suppose that $D(n)$ is Hamiltonian-ununfoldable by cutting along edges in P . Then P is a Hamiltonian path on $D(n)$. For vertex v we use $\deg_P(v)$ to denote the number of edges incident to v in P . That is, $\deg_P(v) = 1$ for two endpoints and $\deg_P(v) = 2$ for the other vertices because P is a Hamiltonian path. Thus $\deg_P(c)$ is one or two, and almost all vertices q_i have $\deg_P(q_i) = 2$. This implies that for almost all vertices q_i , the path (p_{2i-1}, q_i, p_{2i}) is a part of P . That is, most isosceles triangles will be flipped along their base lines like petals of a flower.

We have two cases. First, we suppose that c is an endpoint of P . Without loss of generality, we can assume that the path (c, q_1, p_1) is in P . Then, because c has no other cut except along (c, q_1) , P contains all subpaths (p_{2i-1}, q_i, p_{2i}) with $1 < i \leq n$ (except $i = 1$). Then we have only two possible ways to make a Hamiltonian path. One is $(c, q_1, p_1, p_{2n}, q_n, p_{2n-1}, \dots, p_4, q_2, p_3, p_2)$, and the other one is $(c, q_1, p_1, p_2, p_3, q_2, p_4, \dots, p_{2n-1}, q_n, p_{2n})$.

The first subcase is illustrated in Figure 4(b) and (c). We first cut along the dotted path in Figure 4(b). Then we flip the *lid*, which consists of all pentagons and one

triangle $p_1 p_2 q_1$ (Figure 4(c)). Now the other triangles have to be flipped, however, the gray triangles overlap with the lid by Lemma 1 if the circle C and the height of the dome are sufficiently small and $n > 12$. Therefore, we cannot develop in this case without overlap. The second subcase is easier: one triangle closer to the lid again overlaps the flipped lid. Therefore, when c is an endpoint of P , every development causes an overlap.

Now we turn to the next case: c is not an endpoint of P . We now assume that the path (q_i, c, q_1, p_1) is in P without loss of generality for some i . When q_i is an endpoint, almost the same argument as the first case works. If $q_i = q_2$ or $q_i = q_n$, one of two petals overlaps, but in other cases, two petals again overlap the flipped lid. Therefore, we consider the case (p_j, q_i, c, q_1, p_1) , where $j = 2i - 1$ or $j = 2i$. If we remove the vertices $\{p_j, q_i, c, q_1, p_1\}$ from the graph obtained from the dome $D(n)$, it is easy to see that the graph is disconnected into two parts. We call the graph induced by $\{p_2, p_3, \dots, p_{j-1}, q_2, q_3, \dots, q_{i-1}\}$ the *right graph*, and the other graph induced by $\{p_{j+1}, p_{j+1}, \dots, p_{2n}, q_{i+1}, \dots, q_n\}$ the *left graph*. Then, clearly, P consists of three parts; P_r for the right graph, P_l for the left graph, and the subpath (p_j, q_i, c, q_1, p_1) joining P_r and P_l . Now we take the larger graph P' between P_r and P_l , apply the same argument as the first case on P' with (p_j, q_i, c, q_1, p_1) , and again obtain an overlap. \square

In the Hamiltonian unfolding, each vertex has degree at most 2 on the cutting path. This can be generalized to any integer $k \geq 2$:

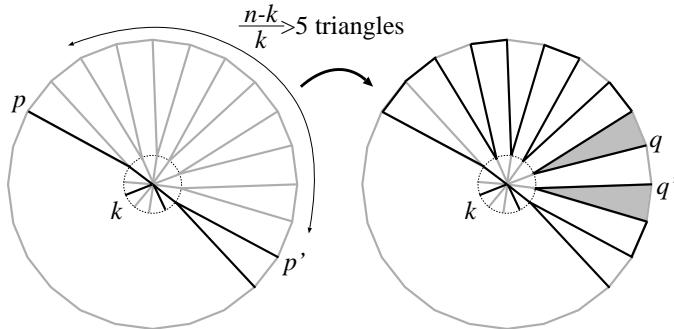


Figure 5: Maximum degree bounded case

Theorem 3 For any positive integer $k \geq 2$, there exists an infinite number of domes that are edge-ununfoldable when the maximum degree of the cutting tree at each vertex is bounded at most k .

We note that all vertices of the dome $D(n)$ have degree 3 except the central vertex c . That is, the cutting tree in Theorem 3 has only one vertex of degree greater than 3.

Proof. We consider the dome $D(n)$ for any $n > 6k$. Let T be any spanning tree of $D(n)$ with maximum degree at most k . We show that the development of $D(n)$ by cutting the edges in T causes an overlap. By definition, the central vertex c has degree at most k . Let T_c be the subtree of T induced by the vertices $\{c\} \cup N_T(c) \cup N_T(N_T(c))$, where $N_T(v)$ is the neighbor set of v on T , and $N_T(N_T(c)) = \cup_{q \in N_T(c)} N_T(q)$. Then, T_c has at most $2k$ leaves because each q_i may have two leaves from p_{2i-1} and p_{2i} in T_c . However, by the expected value argument, we have at least $(n - k)/k > 5$ consecutive triangles on the boundary of the base between two leaves p and p' of T_c (Figure 5). They are cut along T as was done in the proof of Theorem 2. Precisely, all pentagons between p and p' form a lid, and it is then flipped at one boundary edge, say $\{q, q'\}$ (Figure 5). When the triangles between p and p' are flipped, two triangles sharing q and q' (gray triangles in Figure 5) will overlap with the lid by Lemma 1. \square

3 Hamiltonian-Unfoldability of a Prismoid

A *prismoid* is a convex hull of two parallel convex polygons with matching angles. If one of these polygons contains the other in the projection orthogonal to the parallel planes, the prismoid is *nested*. In a nested prismoid, the larger polygon is called the *base* and the other polygon is called the *top*. In general prismoids, we arbitrary name the two parallel convex polygons base and top. The other surface is called the *band*. Because the top and base have matching angles with parallel edges, the band consists of trapezoids.

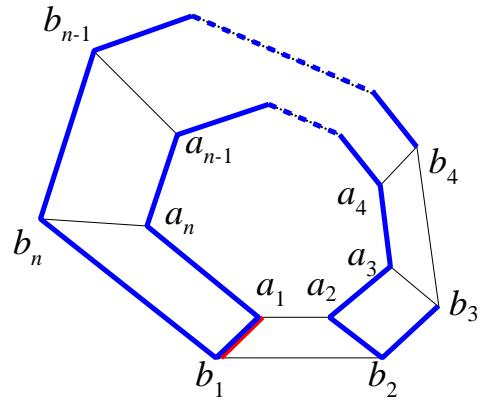


Figure 6: Hamiltonian-unfolding of a nested prismoid (1): (a_1, b_1) is the edge allowing us to unfold the band, and b_3b_4 is the first “acute” edge from b_1b_2 .

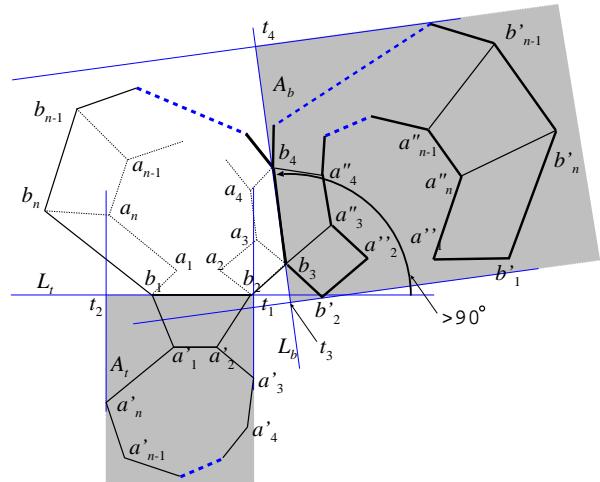


Figure 7: Hamiltonian-unfolding of a nested prismoid (2): the top and the band is fliped and separated from the base by the lines L_t and L_b , respectively.

3.1 Nested Prismoid

Theorem 4 Any nested prismoid has a Hamiltonian unfolding.

Proof. In [ADL⁺08], it is shown that the band of any nested prismoid can be unfolded. That is, the band has at least one edge (not included in base and top) such that by cutting along the edge and unfolding continuously all faces of the band can be placed into a plane without intersection. Let the top and base polygons be $P_t = (a_1, a_2, \dots, a_n)$ and $P_b = (b_1, b_2, \dots, b_n)$, and suppose that the edge (a_1, b_1) allows us to unfold the band.

Then our Hamiltonian unfolding consists of $(b_{i+1}, b_{i+2}, \dots, b_n, b_1, a_1, a_n, a_{n-1}, \dots, a_3, a_2, b_2, b_3, \dots, b_i)$

for some i with $i \geq 2$ (Figure 6). The index i is the first index such that the total turn angle from the vector $\overrightarrow{b_1 b_2}$ to the vector $\overrightarrow{b_i b_{i+1}}$ is greater than 90° . (Intuitively, the vertex b_{i+1} is the first vertex coming back to b_1 . We note that i can be n .) We fix the base in the plane. Then the unfolding can be regarded as two “flipping” (Figure 7): one is the flipping of the top along the axis (b_1, b_2) with the trapezoid $a_1 a_2 b_2 b_1$ as a hinge, and the other one is the flipping of the band (except the trapezoid $a_1 a_2 b_2 b_1$) along the axis (b_i, b_{i+1}) with the trapezoid $a_i a_{i+1} b_{i+1} b_i$ as a hinge. Let $P'_t = (a'_1, a'_2, \dots, a'_n)$ be the flipped top, and $Q = (b'_2, \dots, b'_{i-1}, b_i, b_{i+1}, b'_{i+2}, \dots, b'_n, b'_1, a''_1, a''_2, \dots, a''_n, a''_2)$ be the flipped band (except the trapezoid $b_1 b_2 a'_2 a'_1$). Let L_t and L_b be the line segments that contain $b_1 b_2$ and $b_i b_{i+1}$, respectively.

Now we prove that the Hamiltonian unfolding causes no overlap. We define the area A_t as the union of the rays ℓ perpendicular to L_t such that the endpoint of ℓ is on L_t and ℓ has a nonempty intersection with the flipped top (the left gray area in Figure 7). Let t_1 and t_2 be the rightmost and the leftmost points on L_t , respectively. For L_b and the flipped band, we also define A_b in a similar way. Let t_3 be the point on L_b closest to L_t . Then, it is easy to see that the flipped top is included in A_t and the flipped band is included in A_b .

We will show that A_b is above the line L_t , and hence A_t and A_b are separated by L_t . We have two cases. The first case is that the angle between the vector $\overrightarrow{b_1 b_2}$ and the vector $\overrightarrow{b_i b_{i+1}}$ is less than 180° as in Figure 7. This case is easy; the point t_3 closest to L_t is the intersection of L_b and the perpendicular to L_b that passes through b'_1 or b'_2 . In the worst case, t_3 is at t_1 . In this case, A_t and A_b have an intersection at this point, but this is the only point shared by A_t and A_b . Thus we can see that the Hamiltonian unfolding causes no overlap. Next we assume that the angle between the vector $\overrightarrow{b_1 b_2}$ and the vector $\overrightarrow{b_i b_{i+1}}$ is greater than 180° . In the case, we can use a symmetric argument at the point b_{i+1} . The worst case is that $b_{i+1} = b_n$ and t_3 is at t_2 . Although A_t and A_b can have an intersection at this point, the Hamiltonian unfolding itself causes no overlap. \square

3.2 General Prismoid

Theorem 5 *The number of Hamiltonian paths in a prismoid of $2n$ vertices is $n^3 + 2n^2$ for even n , and $n^3 + 2n^2 - n$ for odd n .*

Proof. Let $P_t = (a_1, a_2, \dots, a_n)$ and $P_b = (b_1, b_2, \dots, b_n)$ be the top and base polygons of the prismoid, respectively. We assume that a_i and b_i are joined by an edge for each $1 \leq i \leq n$. The key observation is that, once we add $(a_{i-1}, a_i, b_i, b_{i+1})$ or $(a_{i-1}, a_i, b_i, b_{i-1})$ as a subpath of a Hamiltonian path,

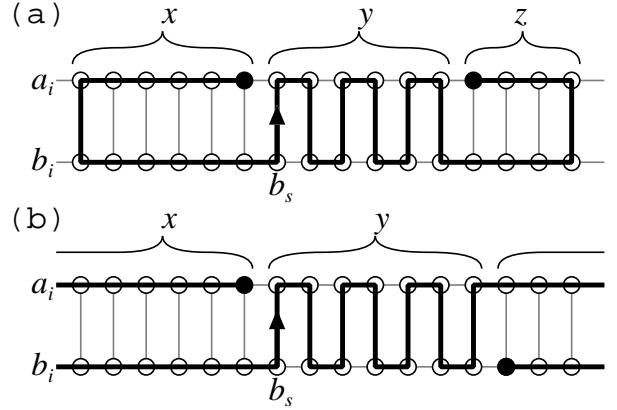


Figure 8: Two possible types of Hamiltonian paths in a prismoid

the graph is separated into two parts at the edge $\{a_i, b_i\}$. Thus we have at most one consecutive zig-zag pattern $(a_{i-1}, a_i, b_i, b_{i+1}, a_{i+1}, a_{i+2}, b_{i+2}, \dots)$ in a Hamiltonian path. The remaining part is filled by two paths in two different ways. The possible patterns are depicted in Figure 8 (the bold arrow indicates the start point of the zig-zag pattern from the vertex b_s). The first one (Figure 8(a)) divides the remaining part into two parts, say, the left and right part. Each of them is filled by a bending path. In the second one (Figure 8(b)), one of two subpaths spans the vertices in P_t , and the other subpath spans the vertices in P_b . (Thus the length of the zig-zag pattern is odd.)

Now we count the number of possible Hamiltonian paths on the prismoid. We first assume that the unique zig-zag pattern starts from $(b_{s-1}, b_s, a_s, a_{s+1})$ as in Figure 8. Then the number of possible combinations of the first case (Figure 8(a)) is the number of partitions of n into three parts of size $x \geq 0$, $y \geq 0$, and $z \geq 0$ with $x + y + z = n$, which is equal to $\binom{n+1}{2}$. On the other hand, the number of possible combinations of the second case (Figure 8(b)) is the number of partitions of n into two parts of size $x \geq 0$ and (odd) $y \geq 0$ with $x + y = n$, which is equal to $\lfloor n/2 \rfloor$. Thus we have $\binom{n+1}{2} + \lfloor n/2 \rfloor$ Hamiltonian paths in the case. We have n ways to choose b_s , and we have the other case that the unique zig-zag pattern starts from $(a_{s-1}, a_s, b_s, b_{s+1})$. Therefore, we have $2n(\binom{n+1}{2} + \lfloor n/2 \rfloor)$ Hamiltonian paths on the prismoid. \square

Corollary 6 *Hamiltonian-unfoldability of a prismoid can be determined in polynomial time. Moreover, all Hamiltonian-unfolding can be enumerated in polynomial time.*

Proof. We can check each cut along a Hamiltonian path in the prismoid to see if it gives us a nonoverlap-

ping unfolding. By Theorem 5, the number of Hamiltonian paths in the prismoid is $O(n^3)$. Thus we can test all possible Hamiltonian unfoldings in polynomial time. \square

4 Conclusion

Some simple families of polyhedra that are edge-unfoldable were presented in [DO07]. Among them, it is easy to see that pyramids and prisms are also Hamiltonian-unfoldable, by so-called “band unfolding”.

As we saw, any nested prismoid is Hamiltonian-unfoldable, and the Hamiltonian unfoldability of a general prismoid can be tested in polynomial time. We conjecture that all prismoids are Hamiltonian-unfoldable. It is worth mentioning that Aloupis showed in his thesis [Alo05] that the band of any prismoid (without top and bottom) can be unfolded. But a naive idea to attach the top and bottom to the unfolded band does not work; there are nested prismoids that cause overlap in any band unfolding [O'R12]. Since Hamiltonian-unfolding is more flexible than band unfolding, we may avoid overlapping for such prismoids.

A generalization of prismoids are *prismatoids*: a prismatoid is the convex hull of any two parallel convex polygons. Theorem 5 cannot be extended to prismatoids because some prismatoids have exponentially many Hamiltonian paths; see Figure 9.

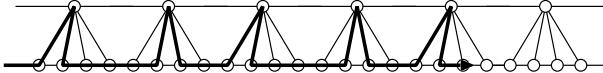


Figure 9: The side profile of a prismatoid that has exponentially many Hamiltonian paths

Acknowledgements

The first and second authors were supported in part by NSF ODISSEI grant EFRI-1240383 and NSF Expedition grant CCF-1138967. This work was initiated when the third author was visiting MIT, and discussed at the 28th Bellairs Winter Workshop on Computational Geometry, co-organized by Erik D. Demaine and Godfried Toussaint, held on February 22–29, 2013, in Holetown, Barbados. We thank the other participants of that workshop for providing a stimulating research environment.

References

- [ADL⁺08] Greg Aloupis, Erik D. Demaine, Stefan Langerman, Pat Morin, Joseph O'Rourke, Ileana Streinu, and Godfried Toussaint. Edge-unfolding

nested polyhedral bands. *Computational Geometry*, 39:30–42, 2008.

- [Alo05] Greg Aloupis. *Reconfigurations of Polygonal Structure*. PhD thesis, School of Computer Science, McGill University, January 2005.
- [DDL⁺10] Erik D. Demaine, Martin L. Demaine, Anna Lubiw, Arlo Shallit, and Jonah L. Shallit. Zipper unfoldings of polyhedral complexes. In *CCCG 2010*, pages 219–222, 2010.
- [DiB90] Julie DiBiase. *Polytope Unfolding*. Undergraduate thesis, Smith College, 1990.
- [DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007.
- [O'R01] Joseph O'Rourke. Unfolding prismoids without overlap. Unpublished manuscript, May 2001.
- [O'R08] Joseph O'Rourke. Unfolding polyhedra. <http://cs.smith.edu/~orourke/Papers/PolyUnf0.pdf>, July 2008.
- [O'R11] Joseph O'Rourke. *How to Fold It: The Mathematics of Linkage, Origami and Polyhedra*. Cambridge University Press, 2011.
- [O'R12] Joseph O'Rourke. Unfolding prismatoids as convex patches: counterexample and positive results. arXiv:1205.2048v1, May 2012.

Map Folding

Rahnuma Islam Nishat*

Sue Whitesides *†

Abstract

A crease pattern is an embedded planar graph on a piece of paper. An $m \times n$ map is a rectangular piece of paper with a crease pattern that partitions the paper into an $m \times n$ regular grid of unit squares. If a map has a configuration such that all the faces of the map are stacked on a unit square and the paper does not self-intersect, then it is flat foldable, and the linear ordering of the faces is called a valid linear ordering. Otherwise, the map is unfoldable. In this paper, we show that given a linear ordering of the faces of an $m \times n$ map, we can decide in linear time whether it is a valid linear ordering, which improves the quadratic time algorithm of Morgan. We also define a class of unfoldable $2 \times n$ mountain-valley patterns for every $n \geq 5$.

1 Introduction

A *piece of paper* is a connected polygon in \mathbb{R}^2 , with or without holes. A paper has a *light side* and a *dark side*. A *crease pattern* is an embedded planar graph on a piece of paper. Each edge of a crease pattern that is not on the boundary of the paper is called a *crease*. The crease pattern divides the surface of the paper into a set of bounded regions called *faces*. Each face is bounded by a set of creases and possibly by part of the boundary of the paper. Each crease is incident to exactly two faces. A *vertex* of a crease pattern is an endpoint of a crease that is not on the boundary of the paper.

If a crease pattern partitions a rectangular piece of paper without holes into an $m \times n$ regular grid of unit squares, then the piece of paper is called an $m \times n$ *grid paper* or an $m \times n$ *map*. A crease pattern on an $m \times n$ map is called an $m \times n$ *crease pattern*. When we fold a piece of paper with a given crease pattern, we are restricted to fold the paper only along the creases. A crease can be folded either as a *mountain* or as a *valley*. A *mountain fold* folds the paper such that the two faces incident to the crease touch each other on the dark side after the fold. Similarly, a *valley fold* folds the paper such that the two faces incident to the crease touch each other on the light side after the fold.

A *mountain-valley assignment* is a many-to-one function from the creases in a crease pattern to a label set $\{M, V\}$. A *mountain-valley pattern* is a crease pattern together with a mountain-valley assignment. Figure 1(a) shows a mountain-valley pattern on a 3×3 map. The valley creases are denoted by triple-dot dashed (red) lines and the mountain creases are denoted by dashed (blue) lines.

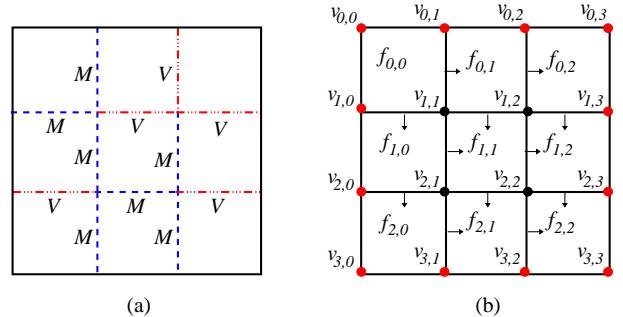


Figure 1: (a) A mountain-valley pattern on a 3×3 map. (b) A 3×3 map with crease pattern C , where the vertices of C are shown as black disks and the dummy vertices are shown as red disks.

Hull [4] gave upper and lower bounds on the number of flat foldable mountain-valley assignments on a single-vertex crease pattern on a disk. Researchers have also been interested in combinatorial problems in origami. Justin [5] enumerated a number of unfoldable mountain-valley patterns on 2×5 , 2×6 and 2×7 maps. Uehara [11] showed that any mountain-valley $1 \times n$ pattern is flat foldable, and gave new upper and lower bounds on the number of flat folded states for that case. Jack Edmonds posed the following open problem in 1997 [3].

Open Problem: *What is the complexity of deciding whether an $m \times n$ map with a given mountain-valley pattern is flat foldable?*

In an attempt to answer the above question, Arkin *et al.* [1] introduced “simple folding” techniques. They showed that any flat foldable 1D mountain-valley pattern is flat foldable using simple folding. Recently, Morgan [8] has given an $O(n^9)$ algorithm for $2 \times n$ mountain-valley patterns and an exponential time algorithm for $m \times n$ mountain-valley patterns. Bern and Hayes [2] proved that both the *flat foldability* and the *assigned flat foldability* problems are NP-complete. The *flat foldability* problem asks whether a paper with a given crease

*Department of Computer Science, University of Victoria, BC, Canada, rnishat@uvic.ca, sue@uvic.ca

†Supported by an NSERC Discovery Grant and the University of Victoria. Results here appear in the first author’s MSc thesis [9].

pattern has a final flat folded state, where the creases are not necessarily labeled and the crease pattern is “locally flat foldable”. In an assigned flat foldability problem, each crease is labeled either mountain or valley.

In this paper, we give an exponential time algorithm to determine whether a given $m \times n$ mountain-valley pattern is flat foldable. We also investigate the combinatorial properties of mountain-valley patterns. The main results of the paper are as follows.

In Section 3, we show that given a linear ordering of the faces of an $m \times n$ mountain-valley pattern, we can decide in linear time whether it is a valid linear ordering or not. In Section 4, we give an exponential time algorithm to decide flat foldability of an $m \times n$ mountain-valley pattern. In Section 5, we show that there is an unfoldable $2 \times n$ mountain-valley pattern for each and every $n \geq 5$ and define a class of unfoldable $2 \times n$ mountain-valley patterns for every $n \geq 5$.

2 Preliminaries

In this section, we define the terminology used throughout the paper. We also mention some previous results that we use.

Let P be a piece of paper. Let \mathbb{C} be a crease pattern on P such that P is flat foldable with respect to \mathbb{C} . Let v be a vertex of \mathbb{C} . Suppose we draw any circle r around v such that no other vertex of \mathbb{C} is on r or inside r . Since P is flat foldable with respect to \mathbb{C} , the disk bounded by r is also flat foldable. The following results are known for a crease pattern on a disk with a single vertex v at the center of the disk.

Lemma 1[7] *The difference between the number of creases with the label mountain and the number of creases with the label valley meeting at v is 2.*

Let P be a map with crease pattern \mathbb{C} . It easily follows from Lemma 1 that each vertex of \mathbb{C} must have either three mountain creases and one valley crease or three valley creases and one mountain crease incident to it when P is flat foldable. If the conditions stated in Lemma 1 is satisfied for all the vertices in \mathbb{C} , we say that the crease pattern \mathbb{C} is *locally flat foldable*.

A *fragment* of P is a subset of faces of \mathbb{C} that form a connected rectangular region (without a hole). A *flat folded state* of P is a stack of disjoint fragments of P that are parallel to each other, connected along the creases of \mathbb{C} , and such that the union of all the fragments is P . Each fragment in the stack is called a *layer*. A *final flat-folded state* of P is a flat folded state where each layer consists of exactly one face of \mathbb{C} . If P has a final flat-folded state, then P is *flat foldable*. A final flat-folded state of P is also called a final flat-folded state of \mathbb{C} . Figure 2 shows an example of a flat folded state of a 6×8 map.

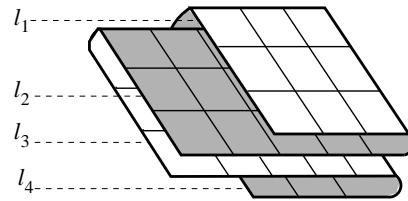


Figure 2: A flat folded state of P with four layers l_1, l_2, l_3 and l_4 .

A *vertex of \mathbb{C}* is an endpoint of a crease of \mathbb{C} that is not on the boundary of the paper. A *vertex of P* is either an endpoint of a crease or a corner of the boundary of P . We call a vertex of P that is not a vertex of \mathbb{C} a *dummy vertex*. Figure 1(b) shows the vertices of a 3×3 map.

We denote by $f_{i,j}$ a face of \mathbb{C} that has the vertices $v_{i,j}, v_{i+1,j}, v_{i,j+1}, v_{i+1,j+1}$ on its boundary as shown in Figure 1(b). For each face $f_{i,j}$ of \mathbb{C} , we associate the creases $(v_{i,j}, v_{i+1,j})$ (left side of the unit square), where $0 < j < n$, and $(v_{i,j}, v_{i,j+1})$ (top of the unit square), where $0 < i < m$, to $f_{i,j}$. The creases associated with each face are shown in Figure 1(b).

A *column c_j* of \mathbb{C} is a set of m faces $f_{0,j}, f_{1,j}, \dots, f_{m-1,j}$, where $0 \leq j \leq n - 1$. A *row r_i* of \mathbb{C} is a set of n faces $f_{i,0}, f_{i,1}, \dots, f_{i,n-1}$, where $0 \leq i \leq m - 1$. The creases associated with a column c_j are the creases associated with the faces in c_j . Similarly, the creases associated with a row r_i are the creases associated with the faces in r_i .

2.1 Checkerboard Pattern

Let us assume that P is flat foldable and let L be the linear ordering of the faces of \mathbb{C} in a final flat folded state S_f of P . Without loss of generality we assume that the face $f_{0,0}$ is facing light side up in S_f and the vertex $v_{0,0}$ is incident to the top-left corner of the unit square on which the faces are stacked. It is easy to observe that the faces that share an edge with $f_{0,0}$ must face dark side up. In a similar way, if a face $f_{i,j}$, $0 \leq i \leq m - 1$ and $0 \leq j \leq n - 1$, is facing light side (respectively, dark side) up, then all the faces that share an edge with $f_{i,j}$ must face dark side (respectively, light side) up. So the faces of \mathbb{C} form a *checkerboard pattern*, where the color of a face f depends on which side of f must face up in any final flat folded state of P (under our assumption), as shown in Figure 4(a).

2.2 Butterflies

A *butterfly* B is a pair of faces f, f' of \mathbb{C} incident to the same crease e . We call f and f' the *wings* of B and the crease e the *hinge* of B . A *pair of butterflies* is a set of two butterflies B_1 and B_2 with no wing in common.

Let S be any flat folded state of P and let B_1, B_2 be a pair of butterflies such that the wings of B_1, B_2 lie above the same unit square u on the XY -plane and the hinges of B_1, B_2 lie above the same edge of u . Let the wings of B_1 and B_2 be f_1, f'_1 and f_2, f'_2 , respectively. Here, f_1 and f_2 denote the lower wings and f'_1 and f'_2 denote the upper wings of their respective butterflies. Then the ordering of the four wings from bottom to top must be one of the following: (f_1, f'_1, f_2, f'_2) , (f_2, f'_2, f_1, f'_1) , (f_2, f_1, f'_1, f'_2) or (f_1, f_2, f'_2, f'_1) , as shown in Figure 3(a)–(d), respectively. Note that the ordering of the wings cannot be (f_1, f_2, f'_1, f'_2) or (f_2, f_1, f'_2, f'_1) as P would self-intersect. If the order of the wings is as in Figure 3(a) or (b), we say that B_1 and B_2 *stack*. Otherwise, we say that B_1 and B_2 *nest*.

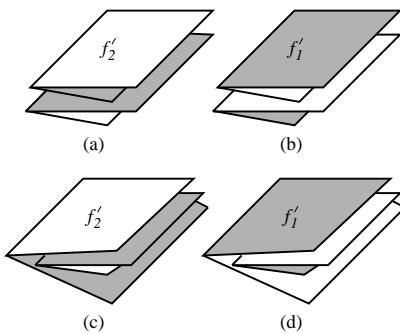


Figure 3: A pair of butterflies B_1, B_2 , where (a) B_2 is stacked on B_1 , (b) B_1 is stacked on B_2 , (c) B_1 nests in B_2 , and (d) B_2 nests in B_1 .

If P is flat foldable, then there exists a final flat folded state of P where all the faces of \mathbb{C} lie above a unit square u on the XY -plane. Since we assume that in any final flat folded state (if one exists) of P , $v_{0,0}$ is incident to the top-left corner of u , the horizontal creases in row r_i , $0 \leq i \leq m-1$, lie above the top edge of u when i is even. We call the butterflies that have these creases as hinges the *north butterflies*. Similarly, the horizontal creases in row r_i , $0 \leq i \leq m-1$, lie above the bottom edge of u when i is odd. We call butterflies with these hinges the *south butterflies*. The vertical edges in column c_j , $0 \leq j \leq n-1$, lie above the left edge of u when j is even and they lie above the right edge of u when j is odd. We call butterflies with those hinges the *west butterflies* and the *east butterflies*, respectively. A pair of butterflies B_1 and B_2 is called a pair of *twin butterflies* if both of them are north or south or east or west butterflies.

2.3 Directed Network

Let B be a butterfly of P with hinge e and wings f, f' . Since f and f' are adjacent faces, exactly one of them has light side up in the checkerboard pattern. Without loss of generality, we assume that f has the light side up. Then the label of e (mountain or valley) determines

the *ordering* of f and f' . If e has the label mountain, then f (the face with the light side up) comes above f' (the face with dark side up). We denote the ordering by $f \prec f'$, where \prec means ‘comes above’. On the other hand, if e has the label valley, then the ordering is $f' \prec f$. The labels of all the creases give a *directed network* (of the faces) of \mathbb{C} . For example, Figure 4(a) shows a 2×2 mountain-valley pattern. The creases

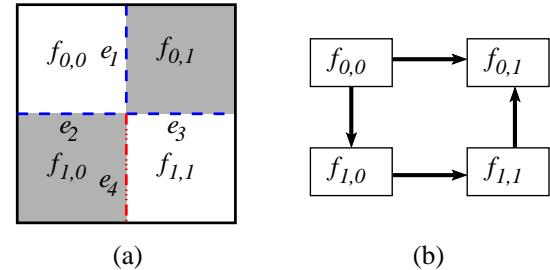


Figure 4: (a) A 2×2 mountain-valley pattern, and (b) the directed network.

e_1, e_2, e_3 and e_4 impose the following directed network. (See Figure 4(b).) $f_{1,1}$ is facing light side up and $f_{1,0}$ is facing dark side up. Since e_4 has the label valley, then $f_{1,0} \prec f_{1,1}$. $f_{0,0}$ is facing light side up and $f_{1,0}$ is facing dark side up. Since e_2 has the label mountain, then $f_{0,0} \prec f_{1,0}$. $f_{0,1}$ is facing dark side up and $f_{0,0}$ is facing light side up. Since e_1 has the label mountain, then $f_{0,0} \prec f_{0,1}$. In a similar way, $f_{1,1} \prec f_{0,1}$.

Since $f_{0,0}$ comes above all other faces, it must be the topmost face. Similarly, the bottommost face must be $f_{0,1}$. In fact, in this particular example, the directed network gives a unique candidate for a valid linear ordering of the faces of \mathbb{C} , which is $L = (f_{0,0}, f_{1,0}, f_{1,1}, f_{0,1})$ from top to bottom. Notice that the directed network in Figure 4(c) is a directed acyclic graph (DAG).

We claim that the directed network of any flat foldable $m \times n$ mountain-valley pattern must be a DAG. Our approach is independent of, but similar to [8].

Lemma 2 *Let \mathbb{C} be an $m \times n$ mountain-valley pattern. If \mathbb{C} is flat foldable, then its directed network \mathcal{N} is a directed acyclic graph.*

3 Recognizing Valid Linear Orderings

In this section, we give an algorithm to decide whether a given linear ordering of the faces of a mountain-valley pattern \mathbb{C} is a valid linear ordering of \mathbb{C} .

Here is an outline of our algorithm, which is essentially the method of [8]. Let L be any linear ordering of the faces of \mathbb{C} . For each pair of twin butterflies B_1, B_2 in \mathbb{C} , we check whether B_1, B_2 nest, stack or intersect in L . If they either stack or nest, then we check whether the ordering of the wings of B_1 and B_2 satisfies the

ordering in the directed network. If each pair of twin butterflies satisfies the ordering in the directed network and does not intersect, then L is a valid linear ordering. Otherwise, it is not a valid linear ordering.

We now prove the correctness the algorithm.

Theorem 3 *Let P be an $m \times n$ map with the mountain-valley pattern \mathbb{C} . Let L be a linear ordering of the faces of \mathbb{C} . Then L is a valid linear ordering if and only if (a)–(b) hold: (a) every pair of twin butterflies either stacks or nests in L (i.e., satisfies the Butterfly Condition), and (b) L satisfies the directed network \mathcal{N} of \mathbb{C} .*

Proof. We first assume that L is a valid linear ordering of \mathbb{C} . Then Conditions (a) and (b) must hold. Therefore, we assume that Conditions (a) and (b) hold. We first decompose P into $m \times n$ distinct unit squares, where each square is a face of \mathbb{C} . Each of these squares has a light side and a dark side. We stack these squares on a unit square u according to the linear ordering L . The checkerboard pattern of \mathbb{C} decides for each face whether it faces dark or light side up. For each north butterfly B in P , we join its two wings (along the hinge of B) such that its hinge lies above the top edge of u . Since any two north butterflies either nest or stack, there will be no intersection of butterflies. We join the wings of the south, east and west butterflies along the bottom, right and left edge of u in a similar way. In this way, we construct a final flat folded state S_f of P and L is the linear ordering of the faces of \mathbb{C} in S_f . Therefore, L is a valid linear ordering. \square

We now calculate the running time of the algorithm.

Theorem 4 *The running time of the algorithm above is $O(m^2n^2)$. With a careful implementation, the running time can be reduced to $O(mn)$ which is linear in the size of the input.*

Proof. Since there are $O(m^2n^2)$ pairs of twin butterflies, and it takes $O(1)$ time to check whether a pair of twin butterflies intersect and whether the order of the wings of each butterfly satisfies the ordering given by the directed network, the total running time is $O(m^2n^2) \times O(1) = O(m^2n^2)$.

We now show a careful implementation to reduce the time complexity. We first check for each pair of north butterflies whether they intersect or not. We take a two dimensional array $M[0 \dots m - 1][0 \dots n - 1]$ and a stack $S[1 \dots mn]$. At first the stack is empty and each of the entries in M is 0. We preprocess M based on the directed network of \mathbb{C} , and M remains unchanged during the processing of the faces. Here are the rules for preprocessing M .

For each $1 \leq i \leq m - 2$, where i is odd, and for each $0 \leq j \leq n - 1$, we do the following:

- If $f_{i,j}$ faces light side up in the checkerboard pattern of \mathbb{C} and the crease between $f_{i,j}, f_{i+1,j}$ is labeled mountain, then set $M[i+1, j] = 1$. This means that the face $f_{i,j}$ must occur in L before the face $f_{i+1,j}$.
- If $f_{i,j}$ faces dark side up in the checkerboard pattern of \mathbb{C} and the crease between $f_{i,j}, f_{i+1,j}$ is labeled mountain, then set $M[i, j] = 1$.
- If $f_{i,j}$ faces light side up in the checkerboard pattern of \mathbb{C} and the crease between $f_{i,j}, f_{i+1,j}$ is labeled valley, then set $M[i, j] = 1$.
- If $f_{i,j}$ faces dark side up in the checkerboard pattern of \mathbb{C} and the crease between $f_{i,j}, f_{i+1,j}$ is labeled valley, then set $M[i + 1, j] = 1$.

We now take the faces in the order given by L and process them as follows. Let the current face be $f_{x,y}$. If $x < 1$, or $x > m - 2$ and m is even, then it is not a wing of a north butterfly. Therefore, we proceed to the next face in L . Otherwise, it is the wing of a north butterfly and we examine $M[x, y]$.

- If $M[x, y] = 0$, then it is the wing of a butterfly that occurs before the other wing. Push the face $f_{x,y}$ to S .
- If $M[x, y] = 1$, the other wing of the corresponding butterfly is already in the stack. In this case, we check the top of the stack. If the topmost face in the stack is $f_{x+1,y}$ (x is odd) or $f_{x-1,y}$ (x is even), then we pop the topmost face and proceed to the next face. Otherwise, there is an intersection, and hence L is not a valid linear ordering.

We stop when either we detect an intersection or we reach the end of L . Therefore, checking for intersection among the north butterflies takes $O(mn)$ time. Similarly we check the south, west and east butterflies. The running time of the algorithm is $4 \times O(mn) = O(mn)$, linear in the size of the input linear ordering. \square

4 Enumerating Valid Linear Orderings

In this section, we sketch the outline of an exponential-time exact algorithm to enumerate all the valid linear orderings (if any exist) of an $m \times n$ mountain-valley pattern \mathbb{C} . Note that the existence of a valid linear ordering of \mathbb{C} proves that \mathbb{C} is flat foldable.

Since \mathbb{C} is a mountain-valley pattern, there is a unique directed network \mathcal{N} of \mathbb{C} . We assume that \mathcal{N} is a directed acyclic graph; otherwise \mathbb{C} is not flat foldable by Lemma 2. We now enumerate all the linear orderings of the faces of \mathbb{C} using the algorithm of [10], which takes constant amortized time; i.e., the total running time of the algorithm is $O(e(\mathcal{N}))$, where $e(\mathcal{N})$ is the number of linear orderings generated by the algorithm from the partial order \mathcal{N} . We can decide whether a linear ordering is a valid linear ordering in

$O(mn)$ time. From Theorem 1.1 of [6], we know that $e(\mathcal{N}) \leq 2^{mn(\log(mn) - H(\mathcal{N}))} \leq 2^{mn\log(mn)} = O(mn^{mn})$, where $H(\mathcal{N}) \leq \log mn$ is the entropy function of \mathcal{N} . Therefore, enumerating all valid linear orderings takes $O(mn) \times O(mn^{mn}) = O(mn^{mn+1})$ time.

5 Unfoldable Maps

In this section, we define a class χ_n of unfoldable $2 \times n$ mountain-valley patterns, $n \geq 5$. Note that any $2 \times n$ mountain-valley pattern is flat foldable when $n \leq 4$. We first show a subclass S_n of unfoldable $2 \times n$ mountain-valley patterns, for every $n \geq 5$. We then observe that any map with an unfoldable pattern (i.e., a pattern in S_n) as a fragment is unfoldable. Using this result, we define the class χ_n , which includes S_n as a subclass.

Let P be a $2 \times n$ map with a mountain-valley pattern \mathbb{C} . By definition, there are n horizontal creases. We call each of these creases a *spinal crease* and collectively we call these creases the *spine*. We call the $n-1$ vertical creases above the spine the *upper ribs* and the remaining creases the *lower ribs*. We denote the upper ribs in \mathbb{C} by u_1, u_2, \dots, u_{n-1} from left to right. Similarly, the lower ribs are denoted by l_1, l_2, \dots, l_{n-1} from left to right, and the spinal creases are denoted by s_1, s_2, \dots, s_n from left to right. A pair of upper and lower ribs $\{u_i, l_i\}$ incident to the same vertex is called a *pre-spine fold* if they both have mountain or valley label.

We now define the subclass S_n of unfoldable $2 \times n$, $n \geq 5$, mountain-valley patterns. Let \mathbb{C} be a pattern for S_n that satisfies the following (a)–(d).

- (a) \mathbb{C} is locally flat foldable.
- (b) There are exactly two pre-spine folds $\{u_2, l_2\}$ and $\{u_{n-2}, l_{n-2}\}$.
- (c) All the upper ribs receive the same label.
- (d) s_3 receives the opposite label of the upper ribs.

The upper ribs of \mathbb{C} can be labeled either mountain or valley. Without loss of generality we assume that the upper ribs of \mathbb{C} receive the label mountain as shown in Figure 5. Consequently, all the lower ribs of \mathbb{C} except l_2 and l_{n-2} must be labeled valley. Since $\{u_2, l_2\}$ and $\{u_{n-2}, l_{n-2}\}$ are the pre-spine folds, l_2 and l_{n-2} receive the same label mountain as u_2 and u_{n-2} . The spinal creases s_4, \dots, s_{n-2} must all be labeled the same as s_3 , which according to requirement (s) must be labeled valley. To preserve local flat foldability, the other spinal creases s_1, s_2, s_{n-1}, s_n must be labeled mountain (opposite to the label of s_3).

The following lemma shows that \mathbb{C} is unfoldable.

Lemma 5 *Let \mathbb{C} be a $2 \times n$ mountain-valley pattern in S_n , $n \geq 5$. Then \mathbb{C} is unfoldable.*

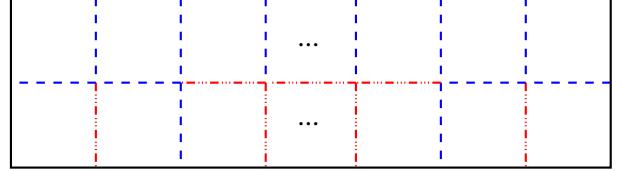


Figure 5: An unfoldable $2 \times n$ mountain-valley pattern.

Proof. (Sketch of proof) We show the case when n is odd. The case when n is even is similar. Let L_i be a candidate for a valid linear ordering of the faces in the columns c_0, \dots, c_i of \mathbb{C} , $0 \leq i \leq n-1$. From the directed network of \mathbb{C} , $L_3 = f_{0,0} \prec f_{1,0} \prec f_{1,1} \prec f_{1,2} \prec f_{0,2} \prec f_{0,3} \prec f_{1,3} \prec f_{0,1}$ is the unique candidate for the case $i = 3$. We show that for each $4 \leq i \leq n-3$, when $n > 5$, the following (a)–(c) hold.

- (a) L_i is the only candidate,
- (b) the order of the faces $f_{0,i}, f_{1,i}, f_{0,i-2}$ and $f_{0,i-1}$ in L_i is $f_{0,i-2} \prec f_{1,i} \prec f_{0,i} \prec f_{0,i-1}$, when i is even, and $f_{0,i-1} \prec f_{0,i} \prec f_{1,i} \prec f_{0,i-2}$, when i is odd, and
- (c) the faces $f_{0,i}, f_{1,i}, f_{0,i-2}$ and $f_{0,i-1}$ are consecutive as a set (i.e., they appear together, with no other faces lying between the extremal faces in this set).

We first construct the unique candidate L_{n-3} (for $n = 5$, L_{n-3} is L_3). We then show that we cannot avoid self-intersection of the paper when constructing L_{n-2} from L_{n-3} .

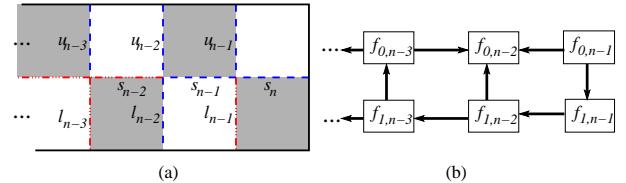


Figure 6: (a) The checkerboard pattern of the last three columns c_{n-3}, \dots, c_{n-1} of \mathbb{C} , when n is odd and (b) the directed network of \mathbb{C} .

When n is odd, $f_{0,n-3} \prec f_{0,n-2}$ from Figure 6(b). By the conditions (a)–(c) above, $f_{0,n-5} \prec f_{1,n-3} \prec f_{0,n-3} \prec f_{0,n-4}$ in L_{n-3} (since $n-3$ is even) and these four faces are consecutive. If $f_{0,n-4} \prec f_{0,n-2}$, then the linear ordering $f_{0,n-5} \prec f_{0,n-3} \prec f_{0,n-4} \prec f_{0,n-2}$ causes intersection between the east butterflies u_{n-4} and u_{n-2} (since n is odd, $n-4$ and $n-2$ are odd and hence the butterflies u_{n-4}, u_{n-2} are east butterflies). Therefore, $f_{0,n-2} \prec f_{0,n-4}$ and the linear ordering of the faces $f_{0,n-3}, f_{0,n-2}, f_{0,n-4}$ in L_{n-2} is $f_{0,n-3} \prec f_{0,n-2} \prec f_{0,n-4}$. Since there is a directed path from $f_{0,n-1}$ to $f_{0,n-3}$ in the directed network, $f_{0,n-1} \prec f_{0,n-3}$. We then have to place $f_{0,n-1}$ somewhere above $f_{0,n-3}$. But any such placement will have

the linear ordering $f_{0,n-1} \prec f_{0,n-3} \prec f_{0,n-2} \prec f_{0,n-4}$, and thus cause intersection between the west butterflies u_{n-3} and u_{n-1} (the indices of the ribs are even since n is odd and hence the butterflies are west butterflies). Therefore, there is no valid linear ordering of \mathbb{C} . \square

We claim that any mountain-valley pattern that has an unfoldable fragment is also unfoldable.

Lemma 6 *Let \mathbb{C} be an $m \times n$ mountain-valley pattern. Let \mathbb{C}' be a fragment of \mathbb{C} . If \mathbb{C}' is not flat foldable, then \mathbb{C} is not flat foldable.*

We now define a class χ_n of unfoldable $2 \times n$ mountain-valley patterns, where $n \geq 5$. We say a pattern belongs to χ_n if and only if \mathbb{C} satisfies the following (a)–(d).

- (a) \mathbb{C} is locally flat foldable.
- (b) There are exactly two pre-spine folds $\{u_i, l_i\}$ and $\{u_j, l_j\}$, where $2 \leq i < j \leq n - 2$.
- (c) All the upper ribs receive the same label and all the lower ribs except u_i, u_j receive the label opposite to the upper ribs.
- (d) s_{i+1} receives the opposite label of the upper ribs.

We now show that every member of χ_n is unfoldable.

Theorem 7 *Let \mathbb{C} be a $2 \times n$ mountain-valley pattern in χ_n , where $n \geq 5$. Then \mathbb{C} is unfoldable. Furthermore, membership in χ_n can be tested in linear time.*

Proof. Let $i = 2$ and $j = n - 2$. Then $\mathbb{C} \in S_n$, and hence the pattern is unfoldable by Lemma 5. Therefore, we assume that $\mathbb{C} \notin S_n$. Let \mathbb{C}' be the fragment of \mathbb{C} with the faces in the columns c_{i-2}, \dots, c_{j+1} . Then $\mathbb{C}' \in S_x$, where $x = j - i + 4$. Therefore, \mathbb{C}' is unfoldable by Lemma 5. Since a fragment of \mathbb{C} is unfoldable, \mathbb{C} is unfoldable by Lemma 6.

We can check in $O(n)$ time whether all the upper ribs receive the same label (i.e., Condition (c) is satisfied) by scanning from left to right. We can check in $O(n)$ time whether \mathbb{C} is locally flat foldable (i.e., Condition (a) is satisfied) by checking the creases incident to each of the $n - 1$ vertices of \mathbb{C} . If Conditions (a) and (c) are satisfied, we check in $O(n)$ time whether there are exactly two pre-spine folds (Condition (b)) and get the index i for the leftmost pre-spine fold $\{u_i, l_i\}$. If Conditions (a)–(c) are satisfied, then it takes $O(1)$ time to check whether the label of s_{i+1} is opposite to the label of the upper ribs (Condition (d)). Therefore, it takes $O(n) + O(n) + O(n) + O(1) = O(n)$ time to check whether \mathbb{C} is a member of χ_n . \square

6 Conclusion

In this paper, we introduced the concepts of *butterflies*, *checkerboard patterns* and *directed networks*. Using these tools, we gave a linear time algorithm to recognize a valid linear ordering and an exponential time

algorithm to decide flat foldability of an $m \times n$ mountain-valley pattern. We also have identified a class of unfoldable $2 \times n$ mountain-valley patterns. It remains open to characterize all the unfoldable $2 \times n$ mountain-valley patterns. It is also open to solve Edmonds' open problem for $m \times n$ maps.

References

- [1] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S. B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Computational Geometry : Theory and Applications*, 29:23–46, September 2004.
- [2] Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proceedings of the 7th annual ACM-SIAM symposium on Discrete algorithms (SODA 1996)*, SODA 1996, pages 175–183. Society for Industrial and Applied Mathematics, 1996.
- [3] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, New York, NY, USA, 2007.
- [4] Thomas Hull. Counting mountain-valley assignments for flat folds. *Ars Combinatoria*, 67, 2003.
- [5] Jacques Justin. Aspects mathématiques du pliage de papier (mathematical aspects of paper fold. In H. Huzita, editor, *1st International Meeting of Origami Science and Scientific Origami*, pages 263–277, 1989.
- [6] Jeff Kahn and Jeong Han Kim. Entropy and sorting. In *Proceedings of the 24th annual ACM symposium on Theory of computing (STOC 1992)*, pages 178–187. ACM, 1992.
- [7] Kunihiko Kasahara and Toshie Takahama. *Origami for the Connoisseur*. Japan Publications Inc., 1987.
- [8] Tom Morgan. Map folding. Master's thesis, Massachusetts Institute of Technology, June 2012.
- [9] Rahnuma Islam Nishat. Map folding. Master's thesis, University of Victoria, BC, Canada, April 2013.
- [10] Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994.
- [11] Ryuhei Uehara. Stamp foldings with a given mountain-valley assignment. In *ORIGAMI 5*, pages 585–597. CRC Press, 2011.

Partial Searchlight Scheduling is Strongly PSPACE-Complete

Giovanni Viglietta*

Abstract

The problem of searching a polygonal region for an unpredictably moving intruder by a set of stationary guards, each carrying an orientable laser, is known as the **SEARCHLIGHT SCHEDULING PROBLEM**. Determining the computational complexity of deciding if the polygon can be searched by a given set of guards is a long-standing open problem.

Here we propose a generalization called the **PARTIAL SEARCHLIGHT SCHEDULING PROBLEM**, in which only a given subregion of the environment has to be searched, as opposed to the entire area. We prove that the corresponding decision problem is strongly **PSPACE**-complete, both in general and restricted to orthogonal polygons where the region to be searched is a rectangle.

Our technique is to reduce from the “edge-to-edge” problem for *nondeterministic constraint logic machines*, after showing that the computational power of such machines does not change if we allow “asynchronous” edge reversals (as opposed to “sequential”).

1 Introduction

Previous work. The **SEARCHLIGHT SCHEDULING PROBLEM** (SSP), first studied in [3], is a pursuit-evasion problem in which a polygon has to be searched for a moving intruder by a set of stationary guards. The intruder moves unpredictably and continuously with unbounded speed, and each guard carries an orientable *searchlight*, emanating a 1-dimensional ray that can be continuously rotated about the guard itself. The polygon’s exterior cannot be traversed by the intruder, nor penetrated by searchlights. The intruder is caught whenever it is hit by a searchlight. Because the intruder’s location is unknown until it is actually caught, each guard has to sway its searchlight according to a predefined *schedule*. If the guards always catch the intruder, regardless of its path, by following their schedules in concert, they are said to have a *search schedule*.

SSP is the problem of deciding if a given set of guards has a search schedule for a given polygon (possibly with holes). The computational complexity of this decision problem has been only marginally addressed in [3], but has later gained more attention, until in [2] the space of

all possible schedules has been shown to be discretizable and reducible to a finite graph, which can be explored systematically to find a search schedule, if one exists. Since the graph may have double exponential size, this technique easily places SSP in **2-EXP**. Whether SSP is **NP-hard** or even in **NP** is left in [2] as an open problem.

More recently, in [5, 7], the author studied the complexity of a 3-dimensional version of SSP, in which the input polygonal environment is replaced by a polyhedron, and the 1-dimensional rays become 2-dimensional half-planes, which rotate about their boundary lines. This variation of SSP is shown to be strongly **NP-hard**.

Our contribution. In the present paper we take a further step along this line of research, by introducing the **PARTIAL SEARCHLIGHT SCHEDULING PROBLEM** (PSSP), in which the guards content themselves with searching a smaller subregion given as input. That is, a search schedule should only guarantee that the given *target region* is eventually cleared, either by catching the intruder or by confining it outside. We prove that PSSP is strongly **PSPACE**-complete, both for general polygons and restricted to orthogonal polygons in which the region to be searched is a rectangle.

To prove that PSSP is a member of **PSPACE**, we do a refined analysis of the discretization technique of [2]. To prove **PSPACE**-hardness, we give a reduction from the “edge-to-edge” problem for nondeterministic constraint logic machines, discussed in [1]. Another contribution of this paper is the observation that the nondeterministic constraint logic model of computation stays essentially the same if we allow “asynchronous” moves, as opposed to “sequential” ones.

An earlier version of this paper has appeared in [4], and most of the material is also contained in the author’s Ph.D. thesis [6].

2 Preliminary observations

An instance of PSSP is a triplet $(\mathcal{P}, G, \mathcal{T})$, where \mathcal{P} is a polygon, possibly with holes, G is a finite set of point guards located in \mathcal{P} or on its boundary, and $\mathcal{T} \subseteq \mathcal{P}$ is a target polygonal region. The question is whether the guards in G can turn their lasers in concert, from a fixed starting position and following a finite schedule, so as to guarantee that in the end any intruder that moves in \mathcal{P} and tries to avoid lasers is necessarily not in \mathcal{T} .

*School of Computer Science, Carleton University, Ottawa ON, Canada, viglietta@gmail.com.

We remark that $\text{SSP} \preceq_{\mathcal{P}} \text{PSSP}$ trivially, in that in SSP we have $\mathcal{T} = \mathcal{P}$ always. One feature of SSP that is not preserved by this generalization is what we call the *time reversal invariance* property. In SSP, a given schedule successfully searches \mathcal{P} if and only if reversing it with respect to time also searches \mathcal{P} . In contrast, this is not the case with PSSP, and Figure 1(a) shows a simple example. The dark target region can be cleared only if the guard turns its searchlight clockwise, as indicated by the arrow. If the searchlight is turned counterclockwise instead, the intruder can first hide in the protected area on the left, then come out and safely reach the target region. Protected areas like this one, that cannot be searched because they are invisible to all guards, provide a constant source of *recontamination*, and will be extensively used in our main **PSPACE**-hardness reduction (see Lemma 4).

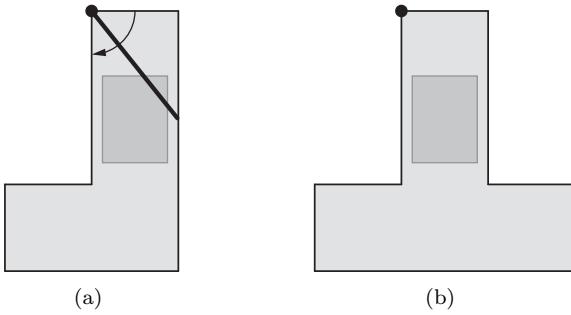


Figure 1: Two instances of PSSP

A search heuristic called *one-way sweep strategy* was described in [3] for SSP restricted to simple polygons, and later extended to polygons with holes in [8]. An interesting consequence of this heuristic is that, if a set of guards lies on the boundary of a simple polygon, and no point in the polygon is invisible to all guards, then there is a schedule that successfully searches the whole polygon. However, such property does not straightforwardly generalize to PSSP, as Figure 1(b) illustrates. Here we have a simple polygon with a guard on the boundary that can see the whole target region. But, because there are protected areas on both sides, the target region is unsearchable, no matter in which direction the guard turns it searchlight.

As it turns out, adding just another guard anywhere on the boundary of the polygon in Figure 1(b) makes the target region searchable. For example, if the second guard is placed on the top-right corner, it can orient its laser downward, thus “closing” the right protected area, and allowing the first guard to search the target region as in Figure 1(a). On the other hand, if the second guard is placed on the boundary of any protected area, then the whole polygon is visible to the guards, and therefore we know that it can be entirely searched.

3 NCL machines and asynchrony

Our **PSPACE**-hardness reduction is based on a model of computation called nondeterministic constraint logic, whose definition and main properties are detailed in [1]. Here we extend the basic model by introducing *asynchrony*, and showing that its computational power stays the same.

Basic NCL machines. Consider an undirected 3-connected 3-regular planar graph, whose vertices can be of two types: *AND vertices* and *OR vertices*. Of the three edges incident to an AND vertex, one is called its *output edge*, and the other two are its *input edges*. Such a graph is (a special case of) a *nondeterministic constraint logic machine (NCL machine)*. A *legal configuration* of an NCL machine is an orientation (direction) of its edges, such that:

- for each AND vertex, either its output edge is directed inward, or both its input edges are directed inward;
- for each OR vertex, at least one of its three incident edges is directed inward.

A *legal move* from a legal configuration to another configuration is the reversal of a single edge, in such a way that the above constraints remain satisfied (i.e., such that the resulting configuration is again legal).

Given an NCL machine with two *distinguished edges* e_a and e_b , and a *target orientation* for each, we consider the problem of deciding if there exist legal configurations A and B such that e_a has its target orientation in A , e_b has its target orientation in B , and there is a sequence of legal moves from A to B . In a sequence of moves, the same edge may be reversed arbitrarily many times. We call this problem **EDGE-TO-EDGE FOR NON-DETERMINISTIC CONSTRAINT LOGIC MACHINES (EE-NCL)**.

A proof that EE-NCL is **PSPACE**-complete is given in [1], by a reduction from **TRUE QUANTIFIED BOOLEAN FORMULA**. By inspecting that reduction, we may further restrict the set of EE-NCL instances on which we will be working. Namely, we may assume that $e_a \neq e_b$, and that in no legal configuration both e_a and e_b have their target orientation.

Asynchrony. For our main reduction, it is more convenient to employ an *asynchronous* version of EE-NCL. Intuitively, instead of “instantaneously” reversing one edge at a time, we allow any edge to start reversing at any given time, and the *reversal phase* of an edge is not “atomic” and instantaneous, but may take any strictly positive amount of time. It is understood that several edges may be in a reversal phase simultaneously. While an edge is reversing, its orientation is undefined,

hence it is not directed toward any vertex. During the whole process, at any time, both the above constraints on AND and OR vertices must be satisfied. We also stipulate that no edge is reversed infinitely many times in a bounded timespan, or else its orientation will not be well-defined in the end. With these extended notions of configuration and move, and with the introduction of “continuous time”, EE-NCL is now called EDGE-TO-EDGE FOR ASYNCHRONOUS NONDETERMINISTIC CONSTRAINT LOGIC MACHINES (EE-ANCL).

Despite its asynchrony, such new model of NCL machine has precisely the same power of its traditional synchronous counterpart.

Theorem 1 EE-NCL = EE-ANCL.

Proof. Obviously $\text{EE-NCL} \subseteq \text{EE-ANCL}$, because any sequence of moves in the synchronous model trivially translates into an equivalent sequence for the asynchronous model.

For the opposite inclusion, we show how to “serialize” a legal sequence of moves for an asynchronous NCL machine going from a legal configuration A to configuration B in a bounded timespan, in order to make it suitable for the synchronous model. An asynchronous sequence is represented by a set

$$S = \{(e_m, s_m, t_m) \mid m \in M\},$$

where M is a set of “edge reversal events”, e_m is an edge with a reversal phase starting at time s_m and terminating at time $t_m > s_m$. For consistency, no two reversal phases of the same edge may overlap.

Because no edge can be reversed infinitely many times, S must be finite. Hence we may assume that $M = \{1, \dots, n\}$, and that the moves are sorted according to the (weakly increasing) values of s_m , i.e., $1 \leq m < m' \leq n \implies s_m \leq s_{m'}$. Then we consider the serialized sequence

$$S' = \{(e_m, m, m) \mid m \in M\},$$

and we claim that it is valid for the synchronous model, and that it is equivalent to S .

Indeed, each move of S' is instantaneous and atomic, no two edges reverse simultaneously, and every edge is reversed as many times as in S , hence the final configuration is again B (provided that the starting configuration is A). We still have to show that every move in S' is legal. Let us do the first m edge reversals in S' , for some $m \in M$, starting from configuration A , and reaching configuration C . To prove that C is also legal, consider the configuration C' reached in the asynchronous model at time s_m , according to S , right when e_m starts its reversal phase (possibly simultaneously with other edges). By construction of S' , every edge whose direction is well-defined in C' (i.e., every edge that is not in

a reversal phase) has the same orientation as in C . It follows that, for each vertex, its inward edges in C are a superset of its inward edges in C' . By assumption on S , C' satisfies all the vertex constraints, then so does C , *a fortiori*. \square

Corollary 2 EE-ANCL is **PSPACE**-complete. \square

4 PSPACE-completeness of PSSP

To prove that PSSP belongs to **PSPACE** we use the discretization technique of [2], and to prove that PSSP is **PSPACE**-hard we give a reduction from EE-ANCL.

Membership. Due to Savitch’s theorem, it suffices to show that PSSP belongs to **NPSPACE**.

Lemma 3 PSSP $\in \text{NPSPACE}$.

Proof. As detailed in [2], a technique known as *exact cell decomposition* allows to reduce the space of all possible schedules to a finite graph G . Each searchlight has a linear number of *critical angles*, which yield an overall partition of the polygon into a polynomial number of *cells*. In the discretized search space, searchlights take turns moving, and can stop or change direction only at critical angles. Thus, a vertex of G encodes the *status* of each cell (either *contaminated* or *clear*) and the critical angle at which each searchlight is oriented.

As a consequence, G can be navigated nondeterministically by just storing one vertex at a time, which requires polynomial space. Notice that deciding if two vertices of G are adjacent can be done in polynomial time: an edge in G represents a move of a single searchlight between two consecutive critical angles, and the updated status of each cell can be easily evaluated. Indeed, cells’ vertices are intersections of lines through input points, hence their coordinates can also be efficiently stored and handled as rational expressions involving the input coordinates.

Now, in order to verify that a path in G is a witness for SSP, one checks if the last vertex encodes a status in which every cell is clear. But the very same cell decomposition works also for PSSP: the analysis in [2] applies even if just a subregion of the polygon has to be searched, and a path in G is a witness for PSSP if and only if its last vertex encodes a status in which every cell that has a non-empty intersection with the target subregion is clear. \square

Hardness. For the **PSPACE**-hardness part, we first give a reduction in which the target region to be cleared is an orthogonal hexagon. Then, in Section 5 we will explain how to modify our construction, should we insist on having a rectangular (hence convex) target region.

Lemma 4 EE-ANCL \preceq_P PSSP restricted to orthogonal polygons.

Proof. We show how to transform a given asynchronous NCL machine G with two distinguished edges e_a and e_b into an instance of PSSP.

A rough sketch of our construction is presented in Figure 2. All the vertices of G are placed in a row (a), and are connected together by a network of thin *corridors* (b), turning at right angles, representing edges of G . (Although G is 3-regular, only a few of its edges are sketched in Figure 2.) Each *subsegment* of a corridor is a thin rectangle, containing a *subsegment guard* in the middle (not shown in Figure 2). Two subsegments from different corridors may indeed cross each other like in (c), but in such a way that the crossing point is far enough from the ends of the two subsegments and from the two subsegment guards (so that no subsegment guard can see all the way through another subsegment). All the vertices of G and all the *joints* between consecutive subsegments (i.e., the turning points of each corridor) are connected via extremely thin *pipes* (d) to the upper area (e), which contains the target region (shaded in Figure 2).

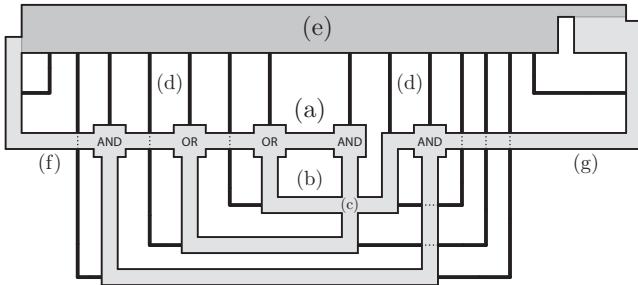


Figure 2: Construction overview

Two corridors (f) and (g) also reach the upper area, and they correspond to the distinguished edges of G , e_a and e_b , respectively. That is, if $e_a = \{u, v\}$, and the target orientation of e_a is toward v , then the corridor corresponding to e_a connects vertex u in our construction to the upper area (e), rather than to v . The same holds for e_b . Indeed, observe that we may assume that e_a and e_b are reversed only once (respectively, on the first and last move) in a sequence of moves that solves EE-ANCL on G . As a consequence, contributions to vertex constraints given by distinguished edges oriented in their target direction may be ignored.

Each pipe turns at most once, and contains one *pipe guard* in the middle, lying on the boundary. Notice that straight pipes never intersect corridors, but some turning pipes do. Figure 3 shows a turning pipe, with its pipe guard (a) and an intersection with a corridor (b) (proportions are inaccurate). The *intersection guards* (c) separate the pipe from the corridor with their

lasers (dotted lines in Figure 3), without “disconnecting” the pipe itself. Although a pipe narrows every time it crosses a corridor, its pipe guard can always see all the way through it, because it is located in the middle. The small *nook* (d) is unclearable because no guard can see its bottom, hence it is a constant source of recontamination for the target region (e), unless the pipe guard is covering it with its laser. (Each straight pipe also has a similar nook.)

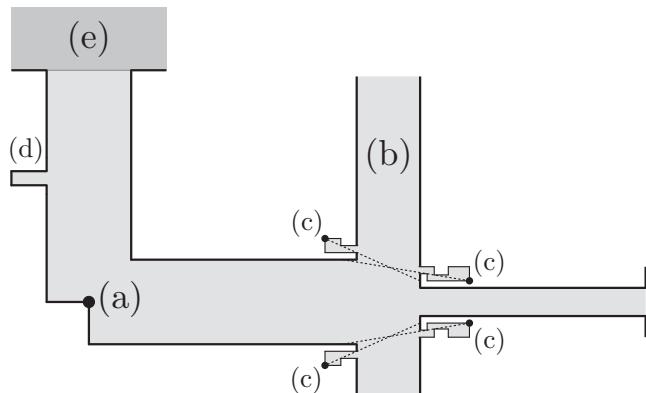


Figure 3: Intersection between a pipe and a corridor

In our construction, corridor guards implement edge orientations in G : whenever all the subsegment guards in a corridor connecting vertices u and v have their lasers oriented in the same “direction” from vertex u to vertex v , it means that the corresponding edge $\{u, v\}$ in G is oriented toward v .

Figure 4 shows an OR vertex. The three subsegment guards from incoming corridors (a) can all “cap” pipe (b) with their lasers, and nook (c) guarantees that the pipe is recontaminated whenever all three guards turn their lasers away.

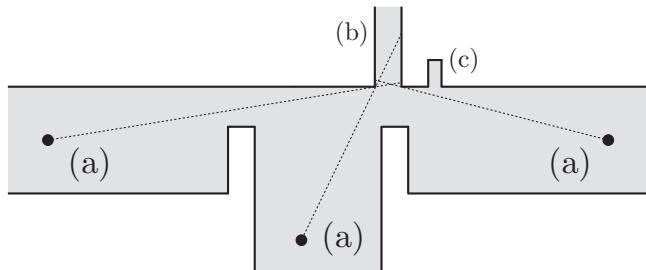


Figure 4: OR vertex

AND vertices are implemented as in Figure 5. The two subsegment guards (a) correspond to input edges, and are able to cap one pipe (e) each, whereas guard (c) can cover them both simultaneously. But that leaves pipe (d) uncovered, unless it is capped by guard (b), which belongs to the corridor corresponding to the output edge. Again, uncovered pipes are recontaminated

by unclearable nooks (f).

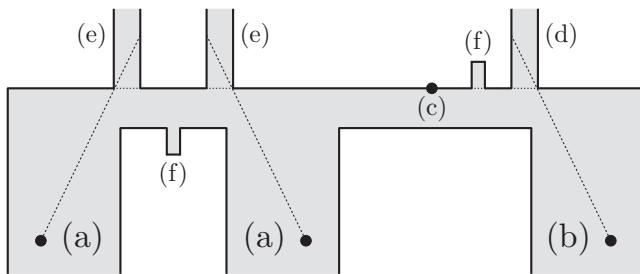


Figure 5: AND vertex

Joints between consecutive subsegments of a corridor may be viewed as OR vertices with two inputs, shaped like in Figure 4, but without the corridor coming from the left.

Finally, Figure 6 shows the upper area of the construction, reached by the distinguished edges e_a and e_b (respectively, (a) and (b)), and by all the pipes (c). The guard in (d) can cap all the pipes, one at a time, and its purpose is to clear the left part of the target region, while the small rectangle (e) on the right will be cleared by the guard in (f). The two pipes (g) implement additional OR vertices with two inputs, and prevent (d) and (f) from acting, unless the respective distinguished edges are in their target orientations. Nook (h) will contaminate part of the target region, unless (d) is aiming down. Nooks (i) prevent area (e) from staying clear whenever guard (f) is not aiming up. The guard in (j) separates the two parts of the target region with its laser, so that they can be cleared in two different moments.

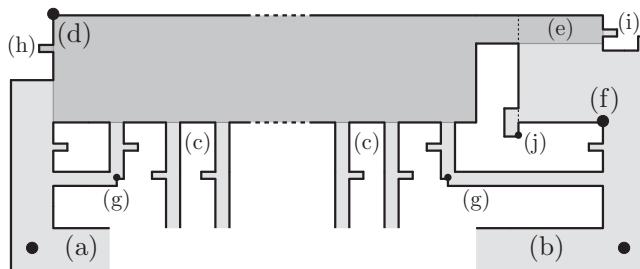


Figure 6: Target region

Suppose G is a solvable instance of EE-ANCL. Then we can “mimic” the transition from configuration A to configuration B (see Section 3) by turning subsegment guards. Specifically, if edge $e = \{u, v\}$ in G changes its orientation from u to v , then all the subsegment guards in the corridor corresponding to e turn their lasers around, one at a time, starting from the guard closest to u . Before this process starts, each pipe has one end capped by some subsegment guard, and in particular pipe (g) on the left of Figure 6 is capped by the

guard in (a). Hence, guard (d) is free to turn and cap all the pipes one by one, stopping for a moment to let each pipe’s internal guard clear the pipe itself (which now has both ends capped) and cover its nook (see Figure 3). As a result, the left part of the target region can be cleared by rotating (d) clockwise, from right to down. Then the subsegment guards start rotating as explained above, until configuration B is reached. If done properly, this keeps all the pipes capped and clear, thus preventing the left part of the target region from being recontaminated. (Note that it makes a difference whether we turn a subsegment guard clockwise or counterclockwise: sometimes, only one direction prevents the recontamination of the pipe that the guard is capping.) When B is reached, guard (f) can turn up to clear (e) and finally solve our PSSP instance.

Conversely, suppose that G is not solvable. Observe that rectangle (e) in Figure 6 has to be cleared by guard (f) as a last thing, because it will be recontaminated by nooks (i) as soon as (f) turns away. On the other hand, whenever a pipe has both ends uncapped by external guards, some portion of the target region necessarily gets recontaminated by some nook, regardless of where the pipe guard is aiming its laser. But guard (d) can cap just one pipe at a time and, while it does so, nook (h) keeps some portion of the target region contaminated. Thus, the entire process must start from a configuration A in which all the pipes’ lower ends are simultaneously capped by subsegment guards, and guard (d) is free to turn (i.e., e_a is in its target orientation). From this point onward, no pipe’s lower end may ever be uncapped (i.e., legality must be preserved), otherwise the target region gets recontaminated, and the process has to restart. Finally, a configuration B must be reached in which guard (f) is free to turn up (i.e., e_b is in its target orientation). By assumption this is impossible, hence our PSSP instance is unsolvable. \square

By putting together Lemma 3 and Lemma 4, we immediately obtain the following:

Theorem 5 *Both PSSP and its restriction to orthogonal polygons are strongly **PSPACE**-complete.* \square

The term “strongly” is implied by the fact that all the vertex coordinates generated in the **PSPACE**-hardness reduction of Lemma 4 are numbers with polynomially many digits (or can be made so through tiny adjustments that do not compromise the validity of the construction).

5 Convexifying the target region

We can further improve our Theorem 5 by making the target region in Lemma 4 rectangular.

Our new target region has the same width as the previous one, and the height of rectangle (e) in Figure 6. In

order for this to work, we have to make sure that some portion of the target region is “affected” by each contaminated pipe that is not capped by guard (d), no matter where *all* the pipe guards are oriented. To achieve this, we make pipes reach the upper area of our construction at increasing heights, from left to right, in a staircase-like fashion.

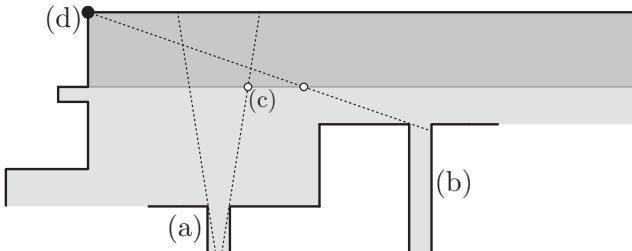


Figure 7: Rectangular target region

Assume we already placed pipe (a) in Figure 7, and we need to find the correct height at which it is safe to connect pipe (b). First we find the rightmost intersection (c) between a laser emanating from the pipe guard of (a) and the lower border of the target region. Then we set the height of pipe (b) so that it is capped by guard (d) when it aims slightly to the right of (c). This is always feasible, provided that pipes are thin enough, which is obviously not an issue.

After we have set all pipes’ heights from left to right, the construction is complete and the proof of Lemma 4 can be repeated verbatim, yielding:

Theorem 6 *Both PSSP and its restriction to orthogonal polygons with rectangular target regions are strongly PSPACE-complete.* \square

6 Further research

There are several promising directions for future research. We suggest a few.

We could simplify PSSP by asking if there exists a neighborhood of a given point, no matter how small, that is clearable. Let this problem be called PSSP*. In contrast with PSSP, here we do not have a polygonal target region, but we are interested just in the surroundings of a point. It is easy to show that PSSP* \leq_P PSSP: clearing a small-enough neighborhood of a point is equivalent to clearing the cells whose topological closure contains the point (cf. the proof of Lemma 3). The author proved in [6] that a 3-dimensional version of PSSP* is PSPACE-hard, even restricted to orthogonal polyhedra. Our question is whether PSSP* is PSPACE-hard (hence PSPACE-complete) for 2-dimensional polygons, as well.

Similarly, we may investigate the complexity of PSSP on other restricted inputs, such as simply connected

polygons, or target regions coinciding with the whole environment. The latter is in fact SSP, whose complexity has been mentioned in Section 1 as an interesting long-standing open problem. Although the author proved that a 3-dimensional version of SSP is NP-hard [5], determining the true complexity of either version still seems a deep problem. Recall that, in our PSPACE-hardness reduction of Lemma 4, we repeatedly used regions that are visible to no guard, and hence can never be cleared. As a matter of fact, this is a remarkably effective way to force the recontamination of other areas whenever certain conditions are met. However, this expedient is of no use in a reduction for SSP (trivially, if the guards cannot see the whole polygon, they cannot search it), and cleverer tools have to be devised for this problem.

Other interesting variations of SSP involve the addition of new environmental elements, such as *mirrors*, which specularly reflect lasers; *transparent walls*, which can be traversed by lasers but not by the intruder; and *curtains*, which can be traversed by the intruder but block lasers. To the best of our knowledge, none of these elements has ever been studied in connection with SSP.

References

- [1] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, vol. 343, pp. 72–96, 2005, special issue “Game Theory Meets Theoretical Computer Science”.
- [2] K. J. Obermeyer, A. Ganguli, and F. Bullo. A complete algorithm for searchlight scheduling. *International Journal of Computational Geometry and Applications*, vol. 21, pp. 101–130, 2011.
- [3] K. Sugihara, I. Suzuki, and M. Yamashita. The searchlight scheduling problem. *SIAM Journal on Computing*, vol. 19, pp. 1024–1040, 1990.
- [4] G. Viglietta. Partial searchlight scheduling is strongly PSPACE-complete. In *Proceedings of the 28th European Workshop on Computational Geometry*, pp. 101–104, Assisi (Italy), 2012.
- [5] G. Viglietta. Searching polyhedra by rotating half-planes. *International Journal of Computational Geometry and Applications*, vol. 22, pp. 243–275, 2012.
- [6] G. Viglietta. *Guarding and searching polyhedra*. Ph.D. Thesis, University of Pisa, 2012.
- [7] G. Viglietta and M. Monge. The 3-dimensional searchlight scheduling problem. In *Proceedings of the 22nd Canadian Conference on Computational Geometry*, pp. 9–12, Winnipeg (Canada), 2010.
- [8] M. Yamashita, I. Suzuki, and T. Kameda. Searching a polygonal region by a group of stationary k -searchers. *Information Processing Letters*, vol. 92, pp. 1–8, 2004.

Set-Difference Range Queries

David Eppstein *

Michael T. Goodrich †

Joseph A. Simons ‡

Abstract

We introduce the problem of performing set-difference range queries, where answers to queries are set-theoretic symmetric differences between sets of items in two geometric ranges. We describe a general framework for answering such queries based on a novel use of data-streaming sketches we call *signed symmetric-difference* sketches. We show that such sketches can be realized using invertible Bloom filters (IBFs), which can be composed, differenced, and searched so as to solve set-difference range queries in a wide range of scenarios.

1 Introduction

Efficiently identifying or quantifying the differences between two sets is a problem that arises frequently in applications, for example, when clients synchronize the calendars on their smart phones with their calendars at work, when databases reconcile their contents after a network partition, or when a backup service queries a file system for any changes that have occurred since the last backup. Such queries can be global, for instance, in a request for the differences across all data values for a pair of databases, or they can be localized, requesting differences for a specific range of values of particular interest. For example, clients might only need to synchronize their calendars for a certain range of dates or a pair of databases may need only to reconcile their contents for a certain range of transactions. We formalize this task by a novel type of range searching problem, which we call *set-difference range queries*.

We assume a collection \mathcal{X} of sets $\{X_1, X_2, \dots, X_N\}$, containing *data items* that are each associated with a geometric point and with a member of universe, \mathcal{U} , of size $U = |\mathcal{U}|$. A set-difference range query is specified by the indices of a pair of data sets, X_i and X_j , and by a pair of *ranges*, R_1 and R_2 , which are each a constant-size description of a set of points such as a hyper-rectangle, simplex, or half-space. The answer to this set-difference range query consists of the elements of X_i and X_j whose associated points belong to the ranges R_1 and R_2 respectively, and whose associated elements in U are contained in one of the two data sets but not both. Thus, we preprocess \mathcal{X} so that given ranges, R_1 and R_2 , and

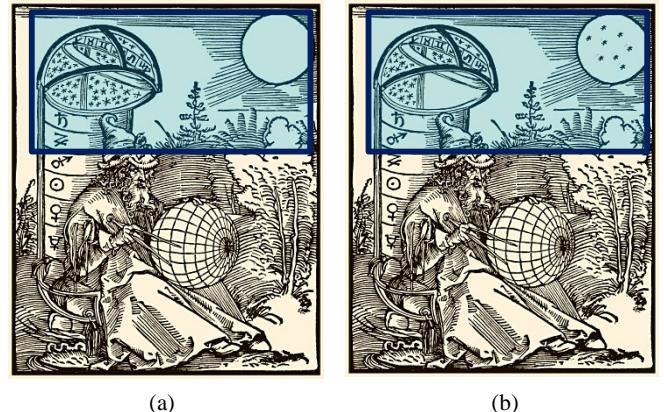


Figure 1: Illustrating the set-difference range query problem. The images in (a) and (b) have four major differences, three of which are inside the common query range. The image (a) is a public-domain engraving of an astronomer by Albrecht Dürer, from the title page of *Messahalah, De scientia motus orbis* (1504).

two sets, X_1 and X_2 , we can quickly report (or count) the universe elements in the set-theoretic symmetric difference $(R_1 \cap X_1) \Delta (R_2 \cap X_2)$. The performance goal in answering such queries is to design data structures having low space and preprocessing requirements that support fast set-difference range queries whose time depends primarily on the size of the difference, not the number of items in the range (see Figure 1). Examples of such scenarios include the following.

- Each set contains readings from a group of sensors in a given time quantum (e.g., see [3]). Researchers may be interested in determining which sensor values have changed between two time quanta in a given region.
- Each set is a catalog of astronomical objects in a digital sky survey. Astronomers are often interested in objects that appear or disappear in a given rectangular region between pairs of nightly observations. (E.g., see [22].)
- Each set is an image taken at a certain time and place. Various applications may be interested in pinpointing changes that occur between pairs of images (e.g., see [18]), which is something that might be done, for instance, via two-dimensional binary search and repeated set-difference range queries.

*Dept. of Comp. Sci., U. of CA, Irvine, eppstein(at)uci.edu

†Dept. of Comp. Sci., U. of CA, Irvine, goodrich(at)uci.edu

‡Dept. of Comp. Sci., U. of CA, Irvine, jsimons(at)uci.edu

Query Type		Query Time	Space
Orthogonal:	Standard [5]	$O(\log^{d-1} n)$	$O(n \log^{d-1} n)$
	SD fixed m	$O(m \cdot \log^d n)$	$O(m \cdot n \log^{d-1} n)$
	SD variable m	$O(m \cdot \log^d n)$	$O(n \log^d n)$
	SD size est.	$O(\log^{d+1} n \log U)$	$O(n \log^d n \log U)$
Simplex:	Standard [17]	$O(n^{1-1/d} (\log n)^{O(1)})$	$O(n)$
	SD fixed m	$O(m \cdot n^{1-1/d} (\log n)^{O(1)})$	$O(m \cdot n)$
	SD variable m	$O(m \cdot n^{1-1/d} (\log n)^{O(1)})$	$O(n \log \log n)$
	SD size est.	$O(n^{1-1/d} (\log n)^{O(1)} \log U)$	$O(n \log n \log U)$
Stabbing:	Standard [4]	$O(\log n)$	$O(n \log n)$
	SD fixed m	$O(m \cdot \log n)$	$O(m \cdot n \log n)$
	SD variable m	$O(m \cdot \log n)$	$O(n \log n)$
	SD size est.	$O(\log^2 n \log U)$	$O(n \log n \log U)$
Partial Sum:	Standard ¹	$O(1)$	$O(n)$
	SD fixed m	$O(m)$	$O(m \cdot n)$
	SD variable m	$O(m)$	$O(n^2)$
	SD size est.	$O(\log n \log U)$	$O(n \log n \log U)$

Table 1: The results labeled “Standard” are previously known results for each data structure. Results labeled “SD” indicate bounds for set-difference range queries. Here d is the dimension of the query, m is the output size, and we assume the approximation factor $(1 \pm \epsilon)$ and failure probability δ are fixed.

1.1 Related Work

We are not aware of prior work on set-difference range queries. However, Suri *et al.* [20] consider approximate range counting in data streams. Shi and JaJa [19] present a data structure for range queries indexed at a specific time for data that is changing over time, achieving polylogarithmic query times. If used for set-difference range queries, however, their scheme would not produce answers in time proportional to the output size. For a survey of general schemes for range searching data structures, see Agarwal [1].

1.2 Our Results

We provide general methods for supporting a wide class of set-difference range queries by combining signed-symmetric difference sketches with any canonical group or semigroup range searching structure. Our methods solve range difference queries where the two sets being compared may be drawn from the same or different data sets, and may be defined by the same or different ranges. In our data structures, sets are combined in the *multiset model*: two data items may be associated with the same element of U , and if they belong to the same query range they are considered to have multiplicity two, while if they belong to the two different query ranges defining the set difference problem then their cardinalities cancel. The result of a set difference query is the set of all elements whose total cardinality defined in this way is nonzero.

Our data structures are probabilistic and return the

correct results with high probability. Our running times depend on the size of the output, but only weakly depend on the size of the original sets. In particular, we derive the results shown in Table 1 for the following range-query problems (see the full version of this article [11] for details):

- Orthogonal: Preprocess a set of points in R^d such that given a query range defined by an axis-parallel hyper rectangle, we can efficiently report or estimate the size of the set of points contained in the hyper-rectangle
- Simplex: Preprocess a set of points in R^d such that given a query range defined by a simplex in R^d , we can efficiently report or estimate the size of set of points contained in the simplex
- Stabbing: Preprocess a set of intervals such that given a query point x , we can efficiently report or estimate the size of the subset which intersects x .
- Partial Sum: Preprocess a grid of total size $O(n)$ (e.g. an image with n pixels, or a d dimensional array with $O(n)$ entries for any constant d) such that given a query range defined by a hyper-rectangle on the grid, we can efficiently report or estimate the sum of the values contained in that hyper-rectangle.

2 The Abstract Range Searching Framework

In order to state our results in their full generality it is necessary to provide some general definitions from the theory of range searching (e.g., see [1]).

A *range space* is a pair (X, R) where X is a universe of objects that may appear as data in a range searching

¹ Using a standard solution to the partial sum problem.

problem (such as the set of all points in the Euclidean plane) and R is a family of subsets of X that may be used as queries (such as the family of all disks in the Euclidean plane). A *range searching problem* is defined by a range space together with an *aggregation function* f that maps subsets of X to the values that should be returned by a query. For instance, in a *range reporting problem*, the aggregation function is the identity; in a *range counting problem*, the aggregation function maps a subset of X to its cardinality. A data structure for the range searching problem must maintain a finite set $Y \subset X$ of objects, and must answer queries that take a range $r \in R$ as an argument and return the value $f(r \cap Y)$. The goal of much research in range searching is to design data structures that are efficient in terms of their space usage, preprocessing time, update time (if changes to Y such as insertions and deletions are allowed), and query time. In particular, it is generally straightforward to answer a query in time $O(|Y|)$, by testing each member of Y for membership in the query range r ; the goal is to answer queries in an amount of time that may depend on the output size but that does not depend so strongly on the size of the data set.

A range searching problem is said to be *decomposable* if there exists a commutative and associative binary operation \oplus such that, for any two disjoint subsets A and B of X , $f(A \cup B) = f(A) \oplus f(B)$. In this case, the operation \oplus defines a *semigroup*.

Many range searching data structures have the following form, which we call a *canonical semigroup range searching data structure*: the data structure stores the values of the aggregation function on some family F of sets of data values, called *canonical sets*, with the property that every data set $r \cap Y$ that could arise in a query can be represented as the disjoint union of a small number of canonical sets r_1, r_2, \dots, r_K . To answer a query, a data structure of this type performs this decomposition of the query range into canonical sets, looks up the values of the aggregation function on these sets, and combines them by the \oplus operation to yield the query result. Note that this is sometimes called a *decomposition scheme*. For instance, the order-statistic tree is a binary search tree over an ordered universe in which each node stores the number of its descendants (including itself) in the tree, and it can be used to quickly answer queries that ask for the number of data values within any interval of the ordered universe. This data structure can be seen as a canonical semigroup range searching data structure in which the aggregation function is the cardinality, the combination operation \oplus is integer addition, and the canonical sets are the sets of elements descending from nodes of the binary search tree. Every intersection of the data with a query interval can be decomposed into $O(\log n)$ canonical sets, and so interval range counting queries can be answered with

this data structure in logarithmic time.

When the combination operation \oplus has additional properties, they may sometimes be used to obtain greater efficiency. In particular, if \oplus has the structure of a group, then we may form a *canonical group range searching data structure*. Again, such a data structure stores the values of the aggregation function on a family of sets of data values, but in this case it represents the query value as an expression $(\pm f(r_1)) \oplus (\pm f(r_2)) \oplus \dots$, where the canonical sets r_i and their signs are chosen with the property that each element of the query range r belongs to exactly one more positive set than negative set. Again, in the interval range searching problem, one may store with each element its *rank*, the number of elements in the range from $-\infty$ to that element, and answer a range counting query by subtracting the rank of the right endpoint from the rank of the left endpoint. In this example, the ranks are not easy to maintain if elements are inserted and deleted, but they allow interval range queries to be answered by combining only two canonical sets instead of logarithmically many.

2.1 Signed Symmetric-Difference Sketches

Suppose that we want to represent an input set S in space sub-linear in the size of S such that we can compute some function $f(S)$ on our compressed representation. This problem often comes up in the streaming literature, and common solutions include dimension reduction (e.g. by a Johnson-Lindenstrauss transform [15]) and computing a *sketch* of S (e.g. Count-Min Sketch [9]).

A *sketch* σ_S of a set S is a randomized compressed representation of S such that we can approximately and probabilistically compute $f(S)$ by evaluating an appropriate function $f'(\sigma_S)$ on the sketch σ_S . This construct also comes up when handling massive data sets, and in this context the compressed representation is sometimes called a *synopsis* [14].

A sketch algorithm σ is called *linear* if it has a group structure. That is, there exist two operators \oplus and \ominus on sketches σ such that given two multi-sets S and T ,

$$\sigma_{S \uplus T} = \sigma_S \oplus \sigma_T \text{ and } \sigma_{S \setminus T} = \sigma_S \ominus \sigma_T$$

where \uplus and \setminus are the multi-set addition and subtraction operators respectively.

For our results, we define two different types of linear sketches. A *Signed Symmetric-Difference Reporting* (SDR) sketch is a linear sketch that supports a function **report**: given a pair of sketches σ_S and σ_T for two sets S and T respectively, probabilistically compute $S \setminus T$ and $T \setminus S$ using only information stored in the sketches σ_S and σ_T in $O(1 + m)$ time, where m is the cardinality of the output. A *Signed Symmetric-Difference Cardinality* (SDC) sketch is a linear sketch that supports

a function `count`: given a pair of sketches σ_S and σ_T for two sets S and T respectively, probabilistically approximate $|S \Delta T|$ using only information stored in the sketches σ_S and σ_T in time linear in the size of the sketches.

3 Main Results

The main idea of our results is to represent each canonical set by an SDR or an SDC sketch. We implement our signed symmetric-difference counting sketches using a linear sketch based on the frequency moment estimation techniques of Thorup and Zhang [21]. We implement our signed symmetric-difference reporting sketches via an *invertible Bloom filter* (IBF), a data structure introduced for straggler detection in data streams [10]. IBFs can be added and subtracted, giving them a group structure and allowing an IBF for a query range to be constructed from the IBFs for its constituent canonical sets. The difference of the IBFs for two query ranges is itself an IBF that allows the difference elements to be reported when the difference is small. To handle set differences of varying sizes we use a hierarchy of IBFs of exponentially growing sizes, together with some special handling for the case that the final set difference size is larger than the size of some individual canonical set.

Further details of the SDR and SDC sketches are given in later sections. In this section, we assume the existence of SDR and SDC sketches as defined above in order to prove the following three theorems which are the crux of our results listed in Table 1

Theorem 1: Suppose that a fixed limit m on the cardinality of the returned set differences is known in advance of constructing the data structure, and our queries must either report the difference if it has cardinality at most m , or otherwise report that it is too large. In this case, we can answer set-difference range queries with probability at least $1 - \epsilon$ for any range space that can be modeled by a canonical group or semigroup range searching data structure. Our solution stores $O(m)$ words of aggregate information per canonical set, uses a combination operation \oplus that takes time $O(m)$ per combination, and allows the result of this combination to be decoded to yield the query results in $O(m)$ time. If the data structure is updated, the aggregate information associated with each changed canonical set can itself be updated in constant time per change.

Proof. See [11] □

Theorem 2: Suppose that we wish to report set differences that may be large or small, without the fixed bound m , in a time bound that depends on the size of the difference but that may depend only weakly on the size of the total data set. In this case, we can

answer range difference queries with probability at least $1 - \epsilon$ for any range space that can be modeled by a canonical group or semigroup range searching data structure. Our solution stores a number of words of aggregate information per canonical set that is $O(1)$ per element of the set, uses a combination operation \oplus that takes time $O(m)$ (where m is the cardinality of the final set-theoretic difference) and allows the result of this combination to be decoded to yield the query results in $O(m)$ time. If the data structure is updated, the aggregate information associated with each changed canonical set can itself be updated in logarithmic time per change.

Proof. See [11]. □

Theorem 3: Suppose that we wish to report the cardinality of the set difference rather than its elements, and further that we allow this cardinality to be reported approximately, within a fixed approximation ratio that may be arbitrarily close to one. In this case, we can answer range difference queries with probability at least $1 - \epsilon$ for any range space that can be modeled by a canonical group or semigroup range searching data structure. Our solution stores a number of words of aggregate information per canonical set that has size $O(\log n \log U)$, uses a combination operation \oplus that takes time $O(\log n \log U)$ and allows the result of this combination to be decoded to yield the query results in $O(\log n \log U)$ time.

Proof. See [11]. □

4 Invertible Bloom Filters

We implement our SDR sketches using the invertible Bloom filter (IBF) [10], a variant of the Bloom filter [6] for maintaining sets of items that extends it in three crucial ways that are central to our application. First, like the counting Bloom filter [7, 13], the IBF allows both insertions and deletions, and it allows the number of inserted elements to far exceed the capacity of the data structure as long as most of the inserted elements are deleted later. Second, unlike the counting Bloom filter, the IBF allows the elements of the set to be listed back out. And third, again unlike the counting Bloom filter, the IBF allows *false deletions*, deletions of elements that were never inserted, and again it allows the elements involved in false deletion operations to be listed back out as long as their number is small. These properties allow us to represent large sets as small IBFs, and to quickly determine the elements in the symmetric difference of the sets, as we now detail.

For the remainder of this paper, we assume without loss of generality that each element x is an integer. An

IBF supports several simple algorithms for item insertion, deletion, and membership queries; for a review of these basic details of the IBF see [11].

In addition, we can take the difference of one IBF, A , with a table T_A , and another one, B , with table T_B , to produce an IBF, C , with table T_C , representing their signed difference, with the items in $A \setminus B$ having positive signs for their cell fields in C and items in $B \setminus A$ having negative signs for their cell fields in C (we assume that C is initially empty). This simple method is also shown in [11].

Finally, given an IBF, which may have been produced either through insertions and deletions or through a subtract operation, we can list out its contents by repeatedly looking for cells with counts of $+1$ or -1 and removing the items for those cells if they pass a test for consistency. This method therefore produces a list of items that had positive signs and a list of items that had negative signs, and is shown in [11]. In the case of an IBF, C , that is the result of a $\text{subtract}(A, B, C)$ operation, the positive-signed elements belong to $A \setminus B$ and the negative-signed elements belong to $B \setminus A$.

4.1 Analysis

In this section, we extend previous analyses [10, 12] to bound the failure probability for the functioning of an invertible Bloom filter to be less than a given parameter, $\epsilon > 0$, which need not be a constant (e.g., its value could be a function of other parameters).

Theorem 4: Suppose X and Y are sets with m elements in their symmetric difference, i.e., $m = |X \Delta Y|$, and let $\epsilon > 0$ be an arbitrary real number. Let A and B be invertible Bloom filters built from X and Y , respectively, such that each IBF has $\lambda \geq k + \lceil \log k \rceil$ bits in its gSum field, i.e., the range of g is $[1, 2^\lambda]$, and each IBF has at least $2km$ cells, where $k > \lceil \log(m/\epsilon) \rceil + 1$ is the number of hash functions used. Then the listItems method for the IBF C resulting from the $\text{subtract}(A, B, C)$ method will list all m elements of $X \Delta Y$ and identify which belong to $X \setminus Y$ and which belong to $Y \setminus X$ with probability at least $1 - \epsilon$.

Proof. [11]. □

To avoid infinite loops, we make a small change to listItems , forcing it to stop decoding after m items have been decoded regardless of whether there remain any decodable cells. This change does not affect the failure probability and with it the running time is always $O(mk)$.

5 Frequency Moment Estimation

We implement our SDC sketches using frequency moment estimation techniques. Let x be a vector of length

U , and suppose we have a data stream of length m , consisting of a sequence of updates to x of the form $(i_1, v_1), \dots, (i_m, v_m) \in [U] \times [-M, M]$ for some $M > 0$. That is, each update is a pair (i, v) which updates the i th coordinate of x such that $x_i \rightarrow x_i + v$.

The frequency moment of a data stream is given by

$$F_p = \sum_{i \in [U]} x_i^p = \|x\|_p^p.$$

Since the seminal paper by Alon *et al.* [2], frequency moment estimation has been an area of significant research interest. Indeed the full literature on the subject is too rich to survey here. Instead, see e.g. the recent work by Kane *et al.* [16] and the references therein. Kane *et al.* [16] gave algorithms for estimating F_p , $p \in (0, 2)$. Their algorithm requires $O(\log^2(1/\delta))$ time per update and $O(\delta^{-2} \log(mM))$ space. However, faster results are known for estimating the second frequency moment F_2 with constant probability. Thorup and Zhang [21] and Charikar *et al.* [8] independently improve upon the original result of Alon *et al.* [2], to achieve an optimal $O(1)$ update time using $O(\delta^{-2} \log(mM))$ space.

Given a sparse vector X of length m with coordinates bounded by $[-M, M]$, we can estimate $\|X\|_2^2$ by treating it as a data stream and running the algorithm of Thorup and Zhang. The algorithm computes a *sketch* S_X of X , of size $O(\delta^{-2} \log mM)$ such that $\|S_X\|_2^2$ is within a factor of $O(1 \pm \delta)$ of $\|X\|_2^2$ with constant probability. We can improve the probability bound to any arbitrary $O(1 - \epsilon)$ by running the algorithm $O(\log(1/\epsilon))$ times independently to produce $O(\log(1/\epsilon))$ independent sketches S_{Xi} , and taking the median of $\|S_{Xi}\|_p^p$. This strategy takes $O(\log(1/\epsilon))$ time to process each non-zero element in X , and the space required is $O(\delta^{-2} \log(1/\epsilon) \log(mM))$.

Furthermore, each sketch is linear, and therefore we can estimate the frequency moment of the difference of two sparse vectors X and Y by subtracting their sketches; $\|S_X - S_Y\|_2^2$ is within a $O(1 \pm \delta)$ factor of $\|X - Y\|_2^2$ with constant probability, and we can maintain $O(\log(1/\epsilon))$ independent sketches to achieve probability $O(1 - \epsilon)$.

Now, suppose we want to estimate the Hamming distance between two sets. We treat each set as a sparse bit-vector and use the fact that the Hamming distance between two bit vectors is equivalent to the squared Euclidean distance, which is just the second frequency moment of the difference of the vectors. Then we can apply the above strategy for sparse vectors to produce $O(\log(1/\epsilon))$ sketches for each set in $O(\log(1/\epsilon))$ time per element, and we subtract all $O(\log 1/\epsilon)$ pairs of sketches in $O(\delta^{-2} \log(1/\epsilon) \log U)$ time. Finally we compute the second frequency moment of each sketch and take the median over all estimations in $O(\log 1/\epsilon)$ time.

We summarize this strategy in the following theorem.

Theorem 5: Let $0 < \epsilon < 1$ and $0 < \delta < 1$ be arbitrary real numbers. Given two sets, X and Y , taken from a universe of size U , we can compute an estimate \hat{m} such that

$$(1 - \delta)|X \Delta Y| \leq \hat{m} \leq (1 + \delta)|X \Delta Y|,$$

with probability $1 - \epsilon$, using a sketch of size $O(\delta^{-2} \log(1/\epsilon) \log U)$. The preprocessing time, including the time required to initialize the sketch is $O(\delta^{-2} \log(1/\epsilon) \log U + (|Y| + |X|) \log(1/\epsilon))$ and the time to compute the estimate \hat{m} is $O(\delta^{-2} \log(1/\epsilon) \log U)$.

References

- [1] P. K. Agarwal. Range Searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 575–598. CRC Press, Inc., 1997.
- [2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
- [3] M. Basseville. Detecting changes in signals and systems—A survey. *Automatica*, 24(3):309–326, 1988.
- [4] J. L. Bentley. Solution to Klee’s Rectangle Problem. Unpublished manuscript, 1977.
- [5] J. L. Bentley and J. B. Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.
- [6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [7] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting Bloom filters. In *Proc. 14th Eur. Symp. on Algorithms*, volume 4168 of *LNCS*, pages 684–695. Springer-Verlag, 2006.
- [8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 693–703. Springer, 2002.
- [9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, April 2005.
- [10] D. Eppstein and M. T. Goodrich. Straggler Identification in Round-Trip Data Streams via Newton’s Identities and Invertible Bloom Filters. *IEEE Trans. on Knowledge and Data Engineering*, 23:297–306, 2011.
- [11] D. Eppstein, M. T. Goodrich, and J. A. Simons. Set-difference range queries. Arxiv report, arXiv:1306.3482 [cs.DS], June 2013.
- [12] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese. What’s the difference? Efficient set reconciliation without prior context. In *Proc. SIGCOMM*, 2011.
- [13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.
- [14] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In R. E. Tarjan and T. Warnow, editors, *SODA*, pages 909–910. ACM/SIAM, 1999.
- [15] D. M. Kane and J. Nelson. Sparser johnson-lindenstrauss transforms. In D. Randall, editor, *SODA*, pages 1195–1206. SIAM, 2012.
- [16] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff. Fast moment estimation in data streams in optimal space. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM Symp. on Theory of Computing (STOC)*, pages 745–754, 2011.
- [17] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8(3):315–334, October 1992.
- [18] R. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: a systematic survey. *IEEE Trans. Image Processing*, 14(3):294–307, March 2005.
- [19] Q. Shi and J. JaJa. A new framework for addressing temporal range queries and some preliminary results. *Theor. Comput. Sci.*, 332(1-3):109–121, February 2005.
- [20] S. Suri, C. D. Tóth, and Y. Zhou. Range counting over multidimensional data streams. *Discrete Comput. Geom.*, 36(4):633–655, 2006.
- [21] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In J. I. Munro, editor, *SODA*, pages 615–624. SIAM, 2004.
- [22] D. G. York, J. Adelman, J. John E. Anderson, S. F. Anderson, et al. The Sloan digital sky survey: technical summary. *The Astronomical Journal*, 120(3):1579, 2000.

The Unified Segment Tree and its Application to the Rectangle Intersection Problem

David P. Wagner*

Abstract

In this paper we introduce a variation on the multidimensional segment tree, formed by unifying different interpretations of the dimensionalities of the levels within the tree. Nodes in the resulting d -dimensional structure can have up to d parents and $2d$ children. In order to better visualize these relationships we introduce a diamond representation of the data structure. We show how the relative positions of the nodes within the diamond determine the possible intersections between their representative regions. The new data structure adds the capability to detect intersections between rectangles in a segment tree. We use this to solve the “Rectangle Intersection Problem” with a more straightforward algorithm than has been used previously.

1 Introduction

The Segment Tree is a classic data structure from computational geometry which was introduced by Bentley in 1977 [1]. It is used to store a set of line segments, and it can be queried at a point so as to efficiently return a list of all line segments which contain the query point.

The data structure has numerous applications. For example, in its early days it was used to list all pairs of intersecting rectangles from a list of rectangles in the plane [2], to report all rectilinear line segments in the plane which intersect a query line segment [9], and to report the perimeter of a set of rectangles [12]. More recently the segment tree has become popular for use in pattern recognition and image processing [7].

Vaishnavi described one of the first higher dimensional segment trees in 1982 [8]. Introducing his two-dimensional segment tree as a self-described “segment tree of segment trees”, he attached an “inner segment tree”, representing one dimension, to every node of an “outer segment tree”, representing another dimension, and used this for the purpose of storing rectangles in the plane. A point query would then return a list of all rectangles containing the point. The recursive nature of this data structure meant that it could be generalized to arbitrary dimensions. This has been the standard model for high dimensional segment trees ever since.

In this paper, we describe a variation on the higher dimensional segment tree. In two dimensions, this new data structure is formed by merging the two segment trees which would be formed from different choices for the dimensions of the inner and outer segment trees. Our purpose in introducing this variation is not to show that it is faster for a particular application, but rather that it supports an additional operation, namely the detection of rectangle intersections, while retaining the structure and functionality of a segment tree.

In the following sections, we will define the data structure, and show a useful way to visualize it. We introduce several new definitions as they apply to this variation of data structure. We further show some relationships between the nodes, and the regions those nodes represent.

Finally, we demonstrate that the data structure can be used to solve the “Rectangle Intersection Problem”. Existing methods to solve this problem have either involved using range trees, storing d -dimensional rectangles as $2d$ -dimensional points [3], or sweep planes, processing a lower dimensional problem across the sweep [4]. We think that our new data structure represents a more natural way to store this data, and it allows a greatly simplified rectangle intersection algorithm.

2 Segment Tree Properties

In later sections, we will refer to several well known properties of a one-dimensional segment tree. We list them here for the convenience of the reader.

Property 1 *A segment tree storing n segments has a height of $O(\log n)$.*

Property 2 *A segment stored in a segment tree is split into a canonical representation of $O(\log n)$ subsegments, each of which is stored at a different node of the tree.*

Property 3 *The ancestors of the nodes of the canonical representation of a segment consist of, at most, $O(\log n)$ nodes.*

Property 4 *$O(\log n)$ time is required to insert a segment or to query a point in a segment tree.*

Property 5 *If two distinct nodes each store a segment in a segment tree, and one node is a descendant of the other, then the segment of the descendant node is completely contained within segment of the ancestor node.*

*Department of Electronics Engineering, Hanyang University,
dwagnwagn@gmail.com

Property 6 *If two distinct nodes each store a segment in a segment tree, and neither node is an ancestor of the other, then the two segments are disjoint.*

The next property is perhaps less well known, but it follows directly from the previous two properties.

Property 7 *Two line segments stored at any two nodes in a segment tree are either disjoint, or one is completely contained in the other.*

3 Two-dimensional Segment Trees

Here we introduce a number of definitions relating to the Unified Segment tree. We begin with a description of the two-dimensional segment tree given by Vaishnavi in 1982 [8]. Although we will focus primarily on two dimensions hereafter, almost every concept we will describe generalizes into arbitrary dimensions.

Construction of the Vaishnavi segment tree begins with a single one-dimensional segment tree, called the “outer segment tree”, which represents divisions of the plane along one of the two axes. Attached to every node of this outer segment tree, is an “inner segment tree”, representing further subdivisions along the other axis.

Note that the choice of axis for the outer segment tree could have been either the x -axis or the y -axis. Although this choice is arbitrary, it has a great effect on the organization of the data structure. Therefore, let us give different names to the different data structures resulting from this choice.

Definition 1 (xy-segment tree) *Let the xy-segment tree be the Vaishnavi two-dimensional segment tree whose outer segment tree divides the plane along the x -axis, and whose inner segment trees further divide these regions along the y -axis.*

Definition 2 (yx-segment tree) *Let the yx-segment tree be the Vaishnavi two-dimensional segment tree whose outer segment tree divides the plane along the y -axis, and whose inner segment trees further divide these regions along the x -axis.*

A rectangle inserted into a two-dimensional segment tree is first divided into subrectangles along one axis, and then further subdivided along the other axis. These subrectangles are then stored in the nodes of the tree, and if necessary any ancestors of these nodes are created. We show here that the order of these two axes does not affect the set of subrectangles.

Theorem 1 *A rectangle inserted into an xy-segment tree and into a yx-segment tree will be stored as an equivalent set of subrectangles in the each tree.*

Proof. Upon insertion into an xy -segment tree, a rectangle is first divided along the x -axis, and then along the y -axis. In the yx -segment tree the order of these axes is reversed. The canonical subdivisions are the same, regardless of the order in which the two axes are chosen. Therefore subrectangles created and stored in each of the trees are the same. \square

Multiple rectangles inserted into a segment tree are stored independently of each other. So this leads to the following corollary.

Corollary 2 *A set of rectangles inserted into an xy -segment tree and into a yx -segment tree will be stored as an equivalent set of subrectangles each tree.*

Now we define the Unified Segment Tree in two dimensions based on these two structures.

Definition 3 (Unified Segment Tree (in 2 dimensions)) *Define the unified segment tree storing a set of rectangles in the plane to be the data structure created by the following procedure:*

1. *Create an xy -segment tree and a yx -segment tree, and insert the same set of rectangles into both.*
2. *Merge the root of every inner segment tree with the node of the outer segment tree to which it is attached, so that they are considered to be one node.*
3. *Merge any two nodes in the xy -segment tree and the yx -segment tree which represent the same region of the plane, so that they become one node. By Corollary 2 they would contain the same data.*
4. *Add all the possible ancestors to any node which is missing any of its possible ancestors. (Ancestors in this data structure are defined later.)*

We note several features of the new data structure which are not normally associated with segment trees. First, a node may have up to two parents, one from the xy -segment tree and one from the yx -segment tree. Second, a node may have up to four children, two nodes each from the xy -segment tree and from the yx -segment tree. Finally, the new data structure is technically no longer a tree, as it may contain cycles.

4 Parents, Children, Ancestors, Descendants

In order to accommodate the features of the new data structure, we must create new definitions of previously well-defined concepts such as parent and child.

Definition 4 (x -child) *Let an x -child of a node be either of the two nodes representing the regions created when the original node’s representative region is divided in half along the x -axis.*

A node can have a left x -child, and a right x -child.

Definition 5 (x -parent) Let the x -parent relationship be the inverse of the x -child relationship.

Definition 6 (x -ancestor) Let an x -ancestor be any node which can be reached by following a series of x -parent relationships.

Definition 7 (x -descendant) Let an x -descendant be any node which can be reached by following a series of x -child relationships.

Analogous definitions exist for y -child, y -parent, y -ancestor, and y -descendant.

In addition to x -ancestors, y -ancestors, x -descendants and y -descendants, we define additional nodes to be simply ancestors and descendants.

Definition 8 (Ancestor) Let an ancestor of a node be any node which can be reached by following a series of x -parent and/or y -parent relationships.

Definition 9 (Descendant) Let a descendant of a node be any node which can be reached by following a series of x -child and/or y -child relationships.

Some additional properties can be seen from these definitions

Property 8 The x -parent of the y -parent of a node is the same node as the y -parent of its x -parent.

Property 9 The x -children of the y -children of a node are the same nodes as the y -children of its x -children.

5 Visualization

We find it useful to visualize the unified segment tree as a diamond, where the root of the data structure is at the top of the diamond. We divide the diamond into units, such that all nodes representing a rectangle of the same shape and size are located in the same unit.

The two x -children of any node appear in the same unit below and to the left of their x -parent. The two y -children of any node appear in the same unit below and to the right of their y -parent. Thus, each horizontal row of the diamond has double the number of nodes per unit, as the row above it. See Figure 1.

It is possible to see the xy -segment tree and the yx -segment tree embedded in the diamond representation of the unified segment tree. See Figure 2.

Using this visualization, the ancestors of a node appear in the diamond shaped region above the node. The descendants appear in the diamond shaped region below the node. See Figure 3. The number of ancestors can be bounded as follows.

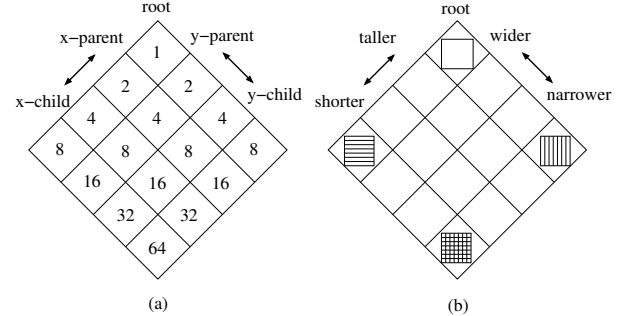


Figure 1: (a) A diamond-shaped visualization of parent child relationships, and the number of nodes in each unit of the diamond. (b) The rectangles which are represented by the nodes in each unit of the diamond.

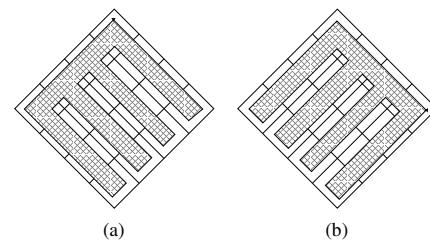


Figure 2: (a) The xy -segment tree embedded into the unified segment tree. (b) The yx -segment tree embedded into the unified segment tree.

Theorem 3 A node has at most one ancestor per unit of the diamond.

Proof. Assume there are two ancestors of a node within the same unit. These two nodes must represent rectangles of the same shape and size, because they are within the same unit. The representative rectangles must contain a common point, since the nodes have a common descendant. The representative rectangles must not partially overlap, by Property 7 of Segment Trees. Therefore, the rectangles, and the nodes representing them, must be the same. \square

We think it is interesting that the nodes taken from the central vertical column of the diamond form a quad tree, while the central column and a neighboring column form a k-D tree. See Figure 4 for a depiction of this.

6 Relationships between Nodes

The relative positions of the nodes within a unified segment tree determine the possible intersections between the rectangular regions they represent. We examine that relationship here.

Theorem 4 A node is a descendant of another node, if and only if it represents a rectangle that falls entirely within the rectangle represented by the ancestor node.

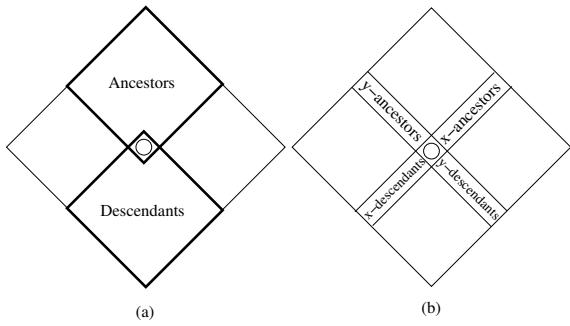


Figure 3: (a) The location of the ancestors and descendants of a node within the diamond. (b) The location of x -ancestors, y -ancestors, x -descendants, and y -descendants within the diamond.

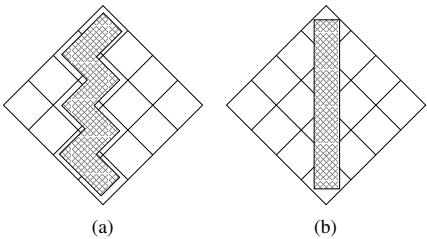


Figure 4: (a) A k-D tree embedded in the unified segment tree. (b) A quad tree embedded in the unified segment tree.

Proof. First, consider two nodes in the segment tree, one of which is a descendant of the other. The rectangle represented by the descendant was formed by successively subdividing the rectangle represented by the ancestor. Therefore, the rectangle represented by a descendant falls entirely within the rectangle represented by any of its ancestors.

Next, consider two rectangles represented by two different nodes of the tree, such that one rectangle falls entirely within the other. Follow the x -parents of the node representing the smaller rectangle, until a node is found which has the same size in the x direction as the larger rectangle. This must have the same x -coordinates as the larger rectangle, otherwise Property 7 would be violated. From there, follow y -parents until a node is found which has the same size in the y direction. This node must have the same y -coordinates as the larger rectangle, for the same reason. Therefore it must be the node which represents the larger rectangle, and the node representing the smaller rectangle must be a descendant of the node representing the larger rectangle. \square

Theorem 5 *Two nodes can have a common ancestor which is an x -ancestor of one node and a y -ancestor of the other, if and only if their representative rectangles completely cross over each other, one in the x direction, and the other in the y direction (see Figure 5).*

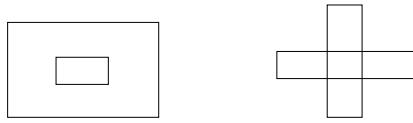


Figure 5: Possible intersections between rectangles represented by nodes in a unified segment tree.

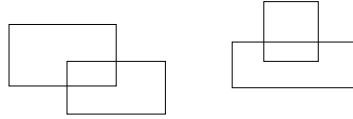


Figure 6: Impossible intersections between rectangles represented by nodes in a unified segment tree.

Proof. Consider any two nodes, having a common ancestor which is an x -ancestor of one and a y -ancestor of the other. The rectangle represented by the common ancestor of the two nodes can be formed by expanding one of the original two rectangles in the x direction, or by expanding the other in the y direction. Therefore, the rectangle of the ancestor must be completely spanned in y direction by the first rectangle, and completely spanned in the x direction by the other. Therefore the two rectangles must completely cross each other.

Next, consider two nodes which represent rectangles that completely cross over each other. The smallest rectangle enclosing both original rectangles can be found by expanding one rectangle in the x -direction or by expanding the other rectangle in the y -direction. Therefore, the enclosing rectangle is represented by an x -ancestor of one of the original nodes, and a y -ancestor of the other node. \square

Theorem 6 *Two rectangles which are represented by nodes in a unified segment tree may only intersect in one of two ways. Either one rectangle can be completely inside of the other, or the two rectangles can completely cross over each other, one in the x direction, and the other in the y direction.*

Proof. By enumerating the possible rectangle intersections, we can see that all other intersections would violate Property 7. See Figures 5 and 6 for a visual depiction of the possible and impossible intersections. \square

7 Analysis

Here we analyze the performance of the unified segment tree, showing bounds on its size, and on the running time of the standard segment tree operations.

Theorem 7 *After the insertion of n rectangles into a unified segment tree, the deepest x -descendant of the root and the deepest y -descendant of the root have a maximum depth of $O(\log n)$.*

Proof. Recall that the unified segment tree was created from an xy -segment tree, and a yx -segment tree. The outer segment trees of these two trees exactly comprise the x -descendants and the y -descendants of the root. By Property 1, these two trees can have a maximum height of $O(\log n)$. \square

Corollary 8 *Any node in a unified segment tree can have a maximum of $O(\log n)$ x -ancestors and a maximum of $O(\log n)$ y -ancestors.*

Theorem 9 *Any node in a two-dimensional unified segment tree can have a maximum of $O(\log^2 n)$ ancestors.*

Proof. A node can have a maximum of $O(\log n)$ x -ancestors, and each of these nodes can have a maximum of $O(\log n)$ y -ancestors. All ancestors can be reached along one of these routes. \square

Theorem 10 *The canonical representation of a rectangle in a two-dimensional segment tree is comprised of a maximum of $O(\log^2 n)$ subrectangles.*

Proof. Any rectangle is decomposed into a maximum of $O(\log n)$ regions in the x direction by Property 2 of Segment Trees. Each of these regions is further subdivided into a maximum of $O(\log n)$ subregions in the y direction. \square

Theorem 11 *All nodes representing the canonical sub-regions of a rectangle have a maximum of $O(\log^2 n)$ ancestors in a two-dimensional unified segment tree.*

Proof. The limit on the number of the ancestors of the canonical representation exists because there can be no more than 16 ancestors of a given shape. Consider that there are 17 ancestors of a particular shape in the tree. By Property 7, these shapes cannot overlap in their x -coordinates, or their y -coordinates, unless the coordinates are the same. Since there are 17 distinct sets of x and y -coordinates, there must be at least 5 distinct pairs of x -coordinates, or 5 distinct pairs of y -coordinates. Assume, without loss of generality that there are 5 distinct pairs of x -coordinates. Consider the node representing the middle of these 5 pairs of coordinates. The x -parent of the node representing this rectangle must be located entirely within the original rectangle. Therefore there is no reason to include the middle rectangle, or any of its descendants in the canonical representation. \square

Corollary 12 *Insertion of a rectangle into a two-dimensional segment tree requires $O(\log^2 n)$ time.*

Corollary 13 *A two-dimensional unified segment tree requires $O(n \log^2 n)$ space to store n rectangles.*

Theorem 14 *A point query of a two-dimensional unified segment tree returns the list of enclosing rectangles in $O(\log^2 n + k)$ time where k is the number of rectangles reported.*

Proof. A point is represented in a unified segment tree at the deepest node. All enclosing rectangles are represented by $O(\log^2 n)$ ancestors of this node. Thus it is sufficient to report all rectangles stored in all ancestors. \square

8 Higher Dimensions

Most every aspect of the unified segment tree can be generalized into arbitrary dimensions in a straightforward way. In d dimensions, each node can have up to d parents, $2d$ children, and $O(\log^d n)$ ancestors. Insertion requires $O(\log^d n)$ time. Query requires $O(\log^d n + k)$ time, where k is the number of results reported. The entire data structure occupies $O(n \log^d n)$ space.

9 The Rectangle Intersection Problem

The rectangle intersection problem is a classic problem dating back to the early days of computational geometry. The problem has been studied by many authors, and numerous variations have been inspired [5, 6].

Although multiple definitions of the “rectangle intersection problem” have appeared in the literature, we use this definition. Store a set of n axis-parallel rectangles so that a rectangle query will efficiently report all rectangles from the set which intersect the query rectangle.

Edelsbrunner and Maurer gave one of the first general purpose algorithms solving this problem in 1981 [4]. Their method involves storing the rectangles in a combination of segment trees and range trees, and these are queried to detect four kinds of intersections between rectangles. In higher dimensions, they sweep across the problem, solving a lower dimensional problems during the sweep, and thereby giving an overall solution that is recursive by dimension.

Edelsbrunner demonstrated two further methods to solve the problem in 1983 [3]. The first of these involves storing the d -dimensional rectangle in a $2d$ -dimensional range tree, and performing an appropriate range query. For the second, a data structure called the “rectangle tree” is developed, and this is queried similarly to the range tree.

Here we show yet another way to solve this problem using an augmented version of our unified segment tree. Our method does not claim a speedup over the previous methods. However, we feel the unified segment tree uses a more natural representation of the data than the range tree or rectangle tree. Additionally, given the machinery that we have already developed in this paper, the algorithm is quite straightforward.

9.1 Augmenting the Unified Segment Tree

For the purpose of supporting a rectangle intersection operation, we augment each node of the unified segment tree with the following additional data:

- A list of rectangles in all descendants of the node.
- A list of rectangles in all x -descendants of the node.
- A list of rectangles in all y -descendants of the node.

Thus a node of a two-dimensional augmented segment tree contains the following information.

```
struct NODE {  
    struct NODE * xparent, yparent;  
    struct NODE * leftxchild, rightxchild;  
    struct NODE * leftychild, rightychild;  
    Segment storedHere[];  
    Segment storedInDescendants[];  
    Segment storedInXDescendants[];  
    Segment storedInYDescendants[];  
}
```

When a segment is inserted, its identifier must be inserted into all canonical nodes, and all ancestors of the canonical nodes, in the appropriate lists. In two dimensions, this does not affect the asymptotic running time of the insert operation. However, in d dimensions, there can exist 2^d separate lists, so an alternate method of storage may be desirable if d is large.

9.2 Rectangle Query Algorithm

A rectangle query operation returns a list of all rectangles which intersect the given query rectangle. Our algorithm for rectangle query first divides the query rectangle into its canonical regions. It then performs a query on each rectangle individually, reporting the union of the rectangles found.

Note that the same rectangle might be found in multiple places, so care must be taken to avoid reporting duplicates, if that is undesirable. If duplicates are reported this may adversely affect the running time by a polylogarithmic factor.

Recall from Theorem 6 that rectangles can only intersect if one is completely inside the other, or if they completely cross over each other, one in each dimension. Therefore, it is sufficient to report the rectangles described in Theorems 4 and 5.

This gives us the following straightforward algorithm, which is performed on each node representing a subrectangle of canonical subdivision of the query rectangle:

1. Report all rectangles stored in ancestors of the node.
2. Report all rectangles stored in the descendant list of the node.

3. Report all rectangles stored in the x -descendant list of a y -ancestor of the node.
4. Report all rectangles stored in the y -descendant list of an x -ancestor of the node.

This information is available in the ancestors of the canonical nodes of the query rectangles. So only $O(\log^2 n)$ nodes need to be accessed. Again care must be taken to avoid reporting duplicates.

Acknowledgments.

The author is grateful to Stefan Langerman and John Iacono for their helpful discussions.

References

- [1] J. L. Bentley. Solutions to Klee's rectangle problems. Technical report, Carnegie-Mellon University, 1977.
- [2] J. L. Bentley and D. Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, C-29(7):571–577, July 1980.
- [3] H. Edelsbrunner. A new approach to rectangle intersections - part I. *International Journal of Computer Mathematics*, 13:209–219, 1983.
- [4] H. Edelsbrunner and H. A. Maurer. On the intersection of orthogonal objects. *Inform. Process. Lett.*, 13:177–181, 1981.
- [5] V. Kapelios, G. Panagopoulou, G. Papamichail, S. Sirmakessis, and A. Tsakalidis. The ‘cross’ rectangle problem. *The Computer Journal*, 38(3):227–235, 1995.
- [6] H. Kaplan, E. Molad, and R. E. Tarjan. Dynamic rectangular intersection with priorities. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 639–648, New York, NY, USA, 2003. ACM.
- [7] G. Racherla, S. Radhakrishnan, and B. J. Oommen. Enhanced layered segment trees: a pragmatic data structure for real-time processing of geometric objects. *Pattern Recognition*, 35(10):2303–2309, 2002.
- [8] V. K. Vaishnavi. Computing point enclosures. *IEEE Transactions on Computers*, C-31:22–29, 1982.
- [9] V. K. Vaishnavi and D. Wood. Rectilinear line segment intersection, layered segment trees, and dynamization. *Journal of Algorithms*, 3:160–176, 1982.
- [10] M. van Kreveld and M. Overmars. Concatenable segment trees. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, pages 493–504, 1989.
- [11] M. van Kreveld and M. Overmars. Union-copy data structures and dynamic segment trees. *Journal of the ACM*, 40:635–652, 1993.
- [12] P. M. B. Vitanyi and D. Wood. Computing the perimeter of a set of rectangles. Technical Report TR 79-CS-23, McMaster University, 1979.

Covering Folded Shapes

Oswin Aichholzer* Greg Aloupis† Erik D. Demaine‡ Martin L. Demaine‡ Sándor P. Fekete§
 Michael Hoffmann¶ Anna Lubiw|| Jack Snoeyink** Andrew Winslow†

Abstract

Can folding a piece of paper flat make it larger? We explore whether a shape S must be scaled to cover a flat-folded copy of itself. We consider both single folds and arbitrary folds (continuous piecewise isometries $S \rightarrow \mathbb{R}^2$). The underlying problem is motivated by computational origami, and is related to other covering and fixturing problems, such as Lebesgue’s universal cover problem and force closure grasps. In addition to considering special shapes (squares, equilateral triangles, polygons and disks), we give upper and lower bounds on scale factors for single folds of convex objects and arbitrary folds of simply connected objects.

1 Introduction

In this paper, we consider covering all possible folded versions of a given shape by a scaled copy of the shape itself, with the objective of keeping the scale factor as small as possible. We explore how folds can make an origami model larger, in the sense that Joseph Wu’s one-fold stegosaurus¹ cannot be covered by a copy of the square from which it is folded.

Problems of covering a family of shapes by one



Completed one fold stegosaurus.
Note alternating plates along the back.

Figure 1: From
Wu’s diagram.

minimum-cost object have a long tradition in geometry. The classical prototype is Lebesgue’s universal cover problem from 1914 [10], which asks for a planar convex set of minimum area that can cover any planar set of diameter at most 1; Brass and Sharifi [3] give the best upper and lower bounds, but a gap remains. A similar question, also with a gap, is Moser’s worm problem [9, 11], which asks for a convex set of minimum area that can cover any planar curve of length 1. As reported in [3] and the book by Brass, Moser, and Pach [2, Chapter 11.4], there is a large family of well-studied, but notoriously difficult problems parameterized by

- the family of sets to be covered,
- the sets allowed as covers,
- the size measure to be minimized, and
- the allowed transformations.

In this paper we consider a given shape S , which is a region of the plane that is a simply connected (no holes) closed 2-manifold with boundary (every interior point has a disk neighborhood and every boundary point a half-disk). A shape S may possess more specific properties: e.g., it may be convex, a (convex or non-convex) polygon, a disk, a square, or an equilateral triangle.

We denote by cS , for $c > 0$, the family of copies of S that have been scaled by c , and then rotated, reflected, and translated. We consider upper and lower bounds on the smallest constant c such that, for any F obtained by folding S , some member of cS contains or *covers* F . Let us be more specific about folding.

A *single fold* of S with line ℓ reflects one or more connected components of the difference $S \setminus \ell$ across ℓ . Let $\mathcal{F}_1(S)$ denote the family of shapes that can be generated by a single fold of S . An *arbitrary fold* of S is a continuous, piecewise isometry from $S \rightarrow \mathbb{R}^2$, which partitions S into a finite number of polygons and maps each rigidly to the plane so that the images of shared boundary points agree. The key property that we will use is that the length of any path in S equals the length of its image in \mathbb{R}^2 . Let $\mathcal{F}(S)$ denote the family of all images of arbitrary folds of S .

The single fold and arbitrary fold are two simple notions of flat folding that avoid concerns of layering and fold order. Note that any upper bound that we prove for arbitrary folds applies to single folds, too. And, although the image of an arbitrary fold need not be the result of single folds, our lower bounds happen to be

*Institute for Software Technology, TU Graz, Inffeldgasse 16b/II, A-8010 Graz, Austria, oaich@ist.tugraz.at. Partially supported by the ESF EUROCORES programme EuroGIGA—CRP ‘ComPoSe’, Austrian Science Fund (FWF): J648-N18.

†Dept. Computer Science, Tufts Univ., 161 College Ave., Medford, MA 02155, USA, aloupis.greg@gmail.com, awinslow@cs.tufts.edu. Partially supported by NSF grant CBET-0941538.

‡Computer Science and Artificial Intelligence Laboratory, MIT, 32 Vassar St., Cambridge, MA 02139, USA, fedemaine,mdemaine}@mit.edu.

§Computer Science, TU Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany, s.fekete@tu-bs.de

¶Institute of Theoretical Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland, hoffmann@inf.ethz.ch. Partially supported by the ESF EUROCORES programme EuroGIGA, CRP GraDR and SNF Project 20GG21-134306.

||David R. Cheriton School of Computer Science, Univ. Waterloo, Waterloo, ONT N2L 3G1, Canada, alubiw@uwaterloo.ca

**Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA, snoeyink@cs.unc.edu. Partially supported by an NSF grant.

¹An origami joke. <http://www.josephwu.com/Files/PDF/stegosaurus.pdf>

limits of finite sequences of single folds. Our results apply to 3-d folded shapes if *covering* is understood to mean covering the orthogonal projection to the plane.

Throughout this paper, we consider *origami covers*:

Definition 1 For a given shape S and $c > 0$, cS is an origami cover of S if any member of $\mathcal{F}(S)$ can be covered by some member of cS . The origami cover factor of S is the smallest such c , which may be ∞ :

$$c^*(S) = \inf\{c \mid cS \text{ is an origami cover of } S\}.$$

Analogously, c_1S is a 1-fold origami cover of S if any member of $\mathcal{F}_1(S)$ can be covered by c_1S ; and

$$c_1^*(S) = \inf\{c \mid cS \text{ is a 1-fold cover of } S\}.$$

Questions of whether folding can increase area or perimeter have been considered before. It is clear that folding a piece of paper introduces overlap, so area can only decrease. On the other hand, the perimeter of a rectangle or square can be greater in a folded than an unfolded state—known as Arnold’s ruble note or the Margulis napkin problem [1, 7]. Folding techniques that increase perimeter, like rumpling and pleat-sinking, make very small but spiky models that are easily covered by the original paper shape, however.

Before we explore single folds in the next section, let us recall some common geometric parameters of a shape S and make one general observation.

For a given shape S , an *incircle*, C_r , is a circle of maximum radius (the *inradius* r) contained in S . Similarly, the *circumcircle*, C_R , is the circle of minimum radius (the *circumradius*) that contains S . For a non-convex shape S , we instead measure geodesic distances within S , i.e., the distance between two points is the length of a shortest path in S between the points. A *geodesic diameter* is a path within S that attains the maximum distance D between two points of S . A *geodesic center* is a point in S that minimizes the maximum distance (the *geodesic radius* R) to all points of S . For convex shapes the geodesic radius R is also the circumradius. Jung’s theorem in the plane says $\sqrt{3}R \leq D \leq 2R$, with the equilateral triangle and circle giving the two extremes [12, ch. 16].

For any folded state of S , these parameters give an upper bound on the origami cover factor.

Lemma 2 Any shape S with inradius r and geodesic radius R has an origami cover factor $c^*(S) \leq R/r$.

Proof. Place any folded state $F \in \mathcal{F}(S)$ in the plane so that the image of a geodesic center is at the origin. Choose a member of $(R/r)S$ with an incircle center at the origin. Because no path in F can be more than R from the origin, the scaled incircle covers F . \square

2 Single Folds

In this section we explore the 1-fold cover factor $c_1^*(S)$, giving general bounds for convex S and for polygons,

and the exact values for equilateral triangles, squares, and a family derived from disks.

2.1 Convex shapes

For a convex set S , there is a lower bound for the 1-fold cover factor $c_1^*(S)$ that is within a constant factor of the upper bound given by Lemma 2.

Theorem 3 Let S be a convex shape with inradius r and circumradius R . Then $\kappa R/r \leq c^*(S) \leq R/r$ for an appropriate constant $\kappa = ((\sqrt{5}-1)/2)^{5/2} \approx 0.300283$.

Proof. The upper bound is from Lemma 2.

For the lower bound, consider the center p^* of the R -circle C_R that contains S . Because R is smallest possible, the set of points where the boundary of C_R touches S , $T := \partial C_R \cap S$, must contain at least two points, and no open halfplane through p^* can contain all of T . If $|T| = 2$, then these two points t_1 and t_2 must lie on a diameter of C_R ; if $|T| > 2$, there must be two points $t_1, t_2 \in T$ that form a central angle $\angle(t_1, p^*, t_2)$ in $[\frac{2}{3}\pi, \pi]$. Thus, for any $\varphi \in [0, \frac{2}{3}\pi]$, we can perform a single fold along a line through p^* that maps t_2 to t'_2 such that the central angle $\angle(t_1, p^*, t'_2)$ is φ .

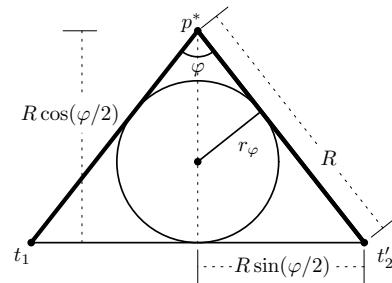


Figure 2: Parameters for calculating the 1-fold cover factor for convex S .

Now, after folding, consider a cover of the three points t_1, p^*, t'_2 by cS for some $c > 0$. As each member of cS is convex, in covering the triangle $\Delta(t_1, p^*, t'_2)$, it also covers the largest circle C_Δ contained in $\Delta(t_1, p^*, t'_2)$; let r_φ be the radius of this circle, see Figure 2. Using elementary geometry we obtain $r_\varphi = \frac{R}{2} \frac{\sin(\varphi)}{1+\sin(\varphi/2)}$, which is maximized at $\varphi = 2 \arctan(((\sqrt{5}-1)/2)^{1/2}) \approx 76.345^\circ$, giving $r_\varphi = \kappa R$ as the radius of C_Δ . Because the largest circle covered by cS has radius cr , and C_Δ is covered by cS , we conclude that $c \geq \kappa R/r$. \square

2.2 Cover factors for specific polygons

In this section we determine $c_1^*(S)$ when S is an equilateral triangle or a square. These two cases illustrate analysis techniques that could in theory be extended to other polygons, except that the number of cases explodes, especially for non-convex shapes.

An important subproblem is to fix the folded shape F and compute, for the given shape S , the smallest c such that cS covers F . With four degrees of freedom for translation, rotation, and scaling, we expect that four first-order contacts between the boundaries of S and F will define the minimum c . In polygons, these will be four pairs consisting of a vertex of F and an edge of S that it lies on. A enclosing triangle, therefore, has some edge touching two vertices of F [5, Lemma 2]; a enclosing square either has some edge touching two vertices or each edge touching one [4].

In the full paper we say more about how these structural characterizations support the use of rotating calipers to compute minimum enclosing shapes. (E.g., an appealing direct construction of the square through four points, which is unique when it exists, is the solution to problem 20 in Kovanova and Radul's list of "Jewish problems" [6]: for points A, B, C, D in ccw order, construct BD' perpendicular and of equal length to AC ; If $D' \neq D$, then two sides of the square must be parallel to DD' .) In what follows we show that the folds that define $c_1^*(S)$ (that maximize the minimum scale factor) are characterized by having multiple equal-sized enclosing shapes.

2.2.1 Equilateral triangle

The example that establishes the maximum 1-fold cover factor of an equilateral triangle is nicely symmetric.

Theorem 4 *The 1-fold cover factor of an equilateral triangle, $c_1^*(\Delta)$, is $4/3$.*

Proof. Let S be the triangle of side length 2 with vertices $(\pm 1, 0)$ and $(0, \sqrt{3})$. We begin by showing that any single fold can be covered by scaling to at most $4/3$.

By symmetry, we may assume that we fold along a line $y = mx + b$ that intersects both edges incident on $(0, \sqrt{3})$; let P be the image of this vertex in the folded state $S' \in \mathcal{F}_1(S)$. Consider three cases for the location of the image P and the resulting minimum enclosing equilateral triangle, depicted in Figure 3.

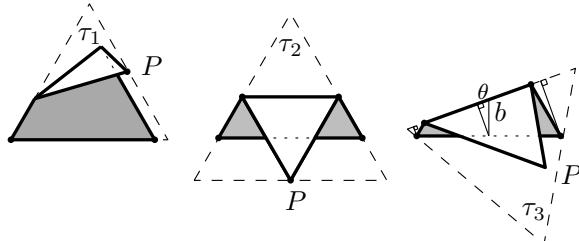


Figure 3: Cases for enclosing triangle depending on P . Point $P \in \tau_3$ should be below $P \in \tau_2$, but then small triangles mentioned in the proof are even harder to see.

First, suppose that P is on or above the x -axis. By symmetry, we may assume that P lies in the wedge

formed by extending both edges of S incident on vertex $(-1, 0)$ to rays from $(-1, 0)$. Because P has distance at most 2 from $(-1, 0)$, scaling S about $(-1, 0)$ by $2/\sqrt{3} < 4/3$ creates an enclosing equilateral triangle τ_1 .

Second, suppose that the image $P = (p_x, p_y)$ has $-\sqrt{3}/3 \leq p_y \leq 0$. Consider the enclosing triangle τ_2 obtained by scaling S about $(0, \sqrt{3})$ until the horizontal edge touches P . The scale factor for this triangle is $\frac{\sqrt{3}-p_y}{\sqrt{3}} = 1 - p_y/\sqrt{3} \leq 4/3$.

Finally, suppose that $P = (p_x, p_y)$ has $p_y \leq -\sqrt{3}/3$. From the previous case, the scale factor for enclosing triangle τ_2 is $1 - p_y/\sqrt{3} \geq 4/3$. So instead consider an enclosing triangle τ_3 with an edge e along the fold line, which we can parameterize by its y -intercept $b \leq \sqrt{3}/3$ and angle from horizontal θ . Draw perpendiculars to e through vertices $(\pm 1, 0)$ to form two small 30-60-90 triangles. Edge e is composed of the short sides of these triangles plus the projection of the base edge of S , so e has length $(2 + 2b/\sqrt{3}) \cos \theta$. Thus, the scale factor of triangle τ_2 is $(1 + b/\sqrt{3}) \cos \theta \leq 4/3 \cos \theta \leq 4/3$.

These cases show that $c_1^*(\Delta) \leq 4/3$, and also reveal necessary conditions for equality: the fold line angle $\theta = 0$ and intercept $b = \sqrt{3}/3$, so $P = (0, -\sqrt{3}/3)$. To show that these are sufficient, we must check one more candidate for enclosing triangle.

Consider τ_4 , with edge incident to $P = (0, -\sqrt{3}/3)$ and $(-1, 0)$. The length of this edge is the sum of sides of two 30-60-90 triangles, marked α and β in Figure 4. The scale factor $(\alpha + \beta)/2 = \sqrt{3}/9 + 2\sqrt{3}/3 = 7\sqrt{3}/9 > 4/3$. Thus, τ_4 is not a minimum enclosing triangle, and $c_1^*(\Delta) = 4/3$, as determined by τ_2 and τ_3 .

This completes the proof. \square

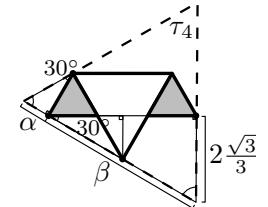


Figure 4: Not a min enclosing triangle.

2.2.2 Square

For squares, the optimal fold is astonishingly complex, and is neither symmetric, nor rational. For the unit square $[0, 1]^2$, the vertex $(0, 1)$ folds to a location whose y coordinate is the root of a degree twelve polynomial: $\Phi(x) = 40x^{12} + 508x^{11} + 1071x^{10} + 930x^9 - 265x^8 - 1464x^7 - 1450x^6 - 524x^5 + 58x^4 + 76x^3 + 3x^2 - 6x - 1$. This polynomial will arise because the optimal fold has three distinct minimum enclosing squares. Let ρ denote the largest (and only positive) real root of $\Phi(x)$, which is approximately 1.105224.

Let $S = \{(x, y) : 0 \leq x, y \leq 1\}$ denote the axis-parallel unit square and consider some $F \in \mathcal{F}_1(S)$ such that $F \neq S$. Note that F is a simple polygon that is uniquely determined (up to symmetry) by a fold line ℓ .

Proposition 5 *The polygon F can be covered by S , unless fold line ℓ intersects S in the relative interior of two opposite sides.*

Proof. If ℓ does not intersect the interior of S then $F \cong S$. Otherwise ℓ intersects ∂S in exactly two points. If these points lie on adjacent sides of S , then folding along ℓ reflects the triangle formed by these sides and ℓ inside the portion of the square S on the opposite side of ℓ . Therefore, F can be covered by S . \square

We are interested in a fold line ℓ that maximizes the smallest enclosing square of F . Using symmetry with Proposition 5, we can assume:

- (1) the line ℓ intersects both horizontal sides of S (else rotate by 90°);
- (2) the slope of ℓ is negative (else reflect vertically);
- (3) ℓ intersects the top side of S left of the midpoint $(1/2, 1)$ (else rotate by 180°).

If we imagine F as the result of folding the part of S to the left of ℓ over to the right, then we can parameterize ℓ by the image $P = (p_x, p_y)$ of the top left corner $(0, 1)$ of S under this fold. Under the above assumptions, a line ℓ that passes (almost) through $(1/2, 1)$ and $(1, 0)$ would maximize p_y . Therefore $0 < p_x < 4/5$ and so $1 < p_y < \sqrt{2p_x - p_x^2 + 1} < 7/5$.

Denote the two points of intersection between ℓ and ∂S by $B = (b_x, 0)$ and $T = (t_x, 1)$ and denote the image of the bottom-left corner $(0, 0)$ of S under the fold across ℓ by $Q = (q_x, q_y)$. If $q_x > 1$, then the convex hull $\mathcal{CH}(F)$ of F is the hexagon $B, (1, 0), Q, (1, 1), P, T$, else Q does not appear on $\partial(\mathcal{CH}(F))$ and it is only a pentagon. Note that in any case the width of F in the y -direction is greater than one, whereas the width in the x -direction is less than one.

For a given $P = (p_x, p_y)$, we have

$$\begin{aligned}\ell : y &= -\frac{p_x}{p_y - 1}x + \frac{p_x^2 + p_y^2 - 1}{2(p_y - 1)}, \\ T &= \left(\frac{p_x^2 + (p_y - 1)^2}{2p_x}, 1 \right), \\ B &= \left(\frac{p_x^2 + p_y^2 - 1}{2p_x}, 0 \right), \text{ and} \\ Q &= \left(\frac{p_x(p_x^2 + p_y^2 - 1)}{p_x^2 + (p_y - 1)^2}, \frac{(p_x^2 + p_y^2 - 1)(p_y - 1)}{p_x^2 + (p_y - 1)^2} \right).\end{aligned}$$

What does a smallest enclosing square σ of F look like? For the upper bound on the cover factor we consider three enclosing squares (Figure 5).

- σ_1 is the smallest axis-parallel enclosing square, which has points B and $(1, 0)$ on the bottom side, P on the top, T on the left, and no point on the right.
- σ_2 has points P and $(1, 1)$ on one side, B on the opposite side, and T on a third side.
- σ_3 has points $B, (1, 0), (1, 1)$, and T appearing in this order, each on a different side of σ_3 .

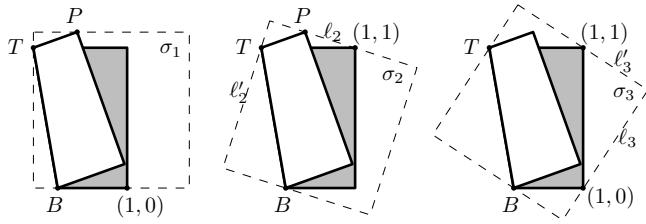


Figure 5: Three minimum enclosing squares for F .

Theorem 6 *The 1-fold cover factor of a square, $c_1^*(\square)$, is ρ , the real root of the degree twelve polynomial Φ .*

Proof. The effort goes into showing that, for each folded shape F , one of the three enclosing squares σ_i , $i \in \{1, 2, 3\}$, as defined above, has side length at most ρ .

Denote the side length of a square σ by $|\sigma|$. For a start it is easy to see that $|\sigma_1| = p_y < 7/5$, which provides a first upper bound.

For σ_2 we have to consider the distance $d(B, \ell_2)$, where ℓ_2 is the line through P and $(1, 1)$ and the distances $d((1, 0), \ell'_2)$ and $d(Q, \ell'_2)$, where ℓ'_2 is the line orthogonal to ℓ_2 through T . Noting that

$$\begin{aligned}d(B, \ell_2) &= \frac{|p_x^2 p_y + p_y^3 + p_x^2 - 2p_x p_y - p_y^2 - p_y + 1|}{2p_x \sqrt{(p_x - 1)^2 + (p_y - 1)^2}} \\ d((1, 0), \ell'_2) &= \frac{|p_y^2 p_x + p_x^3 - p_y^2 - 3p_x^2 + 2p_y + p_x - 1|}{2p_x \sqrt{(p_x - 1)^2 + (p_y - 1)^2}},\end{aligned}$$

it can be checked that the former dominates the latter for $p_y \leq \frac{1}{2}(1 + \sqrt{4p_x - 4p_x^2 + 1})$ and that $d((1, 0), \ell'_2) > p_y$ for $\frac{1}{2}(1 + \sqrt{4p_x - 4p_x^2 + 1}) < p_y < \sqrt{2p_x - p_x^2 + 1}$ (and so $|\sigma_1| \leq |\sigma_2|$ in such a case). Exactly the same holds if $d((1, 0), \ell'_2)$ is replaced by

$$d(Q, \ell'_2) = \frac{|N_1|}{2p_x(1 + (p_x - p_y)^2)\sqrt{(p_x - 1)^2 + (p_y - 1)^2}},$$

where $N_1 = p_x^5 + 2p_x^3 p_y^2 + p_x p_y^4 - p_x^4 - 2p_x^3 p_y - 2p_x p_y^3 + p_y^4 - 4p_x^2 p_y - 4p_y^3 + 4p_x^2 + 2p_x p_y + 6p_y^2 - p_x - 4p_y + 1$. This verifies that σ_2 is enclosing, with side length $|\sigma_2| = d(B, \ell_2)$.

For σ_3 we consider a line $\ell_3 : y = m(x - 1)$ through $(1, 0)$, for some $m > 0$ and the orthogonal line $\ell'_3 : y = (m + 1 - x)/m$ through $(1, 1)$. If σ_3 is a smallest enclosing square, then $d(T, \ell_3) = d(B, \ell'_3)$. For our range of parameters, the only solution is

$$m = \frac{p_x^2 + p_y^2 - 1}{p_x^2 + (p_y - 1)^2},$$

which yields

$$|\sigma_3| = d(T, \ell_3) = \frac{\sqrt{2}|N_2|}{4p_x \sqrt{D_2}},$$

where $N_2 = p_x^4 + 2p_x^2p_y^2 + p_y^4 - 4p_x^3 - 2p_x^2p_y - 4p_xp_y^2 - 2p_y^3 + 4p_xp_y + 2p_y - 1$ and $D_2 = p_x^4 + 2p_x^2p_y^2 + p_y^4 - 2p_x^2p_y - 2p_y^3 + 2p_y^2 - 2p_y + 1$.

Because we choose the smallest square among σ_1 , σ_2 , and σ_3 , the claim certainly holds for $|\sigma_1| = p_y \leq \rho$.

It can be checked that $|\sigma_2| \leq \rho$, for all P with $\rho < p_y < \sqrt{2p_x - p_x^2 + 1}$, except for a small region \mathcal{R} . This region \mathcal{R} is bounded from below by the line $y = \rho$ and from above by the curve $\gamma : |\sigma_2| = \rho$ (the branch of this curve that lies in $\{(x, y) : \rho \leq y < \frac{1}{2}(1 + \sqrt{4x - 4x^2 + 1})\}$). The curve γ intersects the line $y = \rho$ at two points, whose x -coordinates are approximately 0.67969 and 0.77126, respectively. The more interesting of these two is the first point of intersection, which can be described exactly as the smallest positive real root x_ρ of the polynomial $40x^{12} - 116x^{11} - 1045x^{10} + 4756x^9 - 10,244x^8 + 7260x^7 - 8392x^6 - 184x^5 + 620x^4 - 160x^3 + 1088x^2 - 192x + 256$. For the fold defined by $P = (x_\rho, \rho)$ we have $|\sigma_1| = |\sigma_2| = |\sigma_3| = \rho$, while for all other points in \mathcal{R} the corresponding value for $|\sigma_3|$ is strictly less than ρ .

It can also be checked that $|\sigma_3| < \rho$, for any P with $p_y > \rho$ and $\frac{1}{2}(1 + \sqrt{4p_x - 4p_x^2 + 1}) < p_y < \sqrt{2p_x - p_x^2 + 1}$ (above we committed to using σ_2 only if $p_y \leq \frac{1}{2}(1 + \sqrt{4p_x - 4p_x^2 + 1})$).

Altogether it follows that $\min\{|\sigma_i| : i \in \{1, 2, 3\}\} \leq \rho \approx 1.105224446$, as claimed.

Using rotating calipers, one can verify that all other enclosing squares are larger, giving the equality. \square

2.2.3 Polygons and single folds

In the full paper we prove that any polygon (a finite cyclic sequence of vertices and edges with no self-intersections) can be made larger with a single fold. The following lemma is in contrast to observations in Section 3.2 for disks and a family of shapes related to disks.

Lemma 7 *For every plane polygon P , the 1-fold cover factor, $c_1^*(P)$, is greater than 1.*

The idea of the proof is to look for finite sets of structures in P that, if not destroyed by folding, can be covered only by members of that set. For example, the set of diameters in a polygon is finite because the maximum distance D is realized by pairs of vertices, and any diametral pair still at distance D in the folded state F must be covered by a diameter of P , possibly itself.

For a quick example, consider the class of polygons P in which there exist vertices that participate in two or more diametral pairs. (E.g., for odd n , every vertex of a regular n -gon.) Choose as our structure two diametral pairs, pq and qr , that minimize $\theta = \angle pqr$. Fold along a line trisecting θ , reflecting qr to create qr' in the folded shape F . This modified structure has angle $\angle pqr' = \theta/3$

between two diameters; by minimality of θ , it cannot be covered by P .

The proof repeatedly identifies classes of polygons by structures found in the neighborhoods of diameters, until every polygon is in some class. Modifications to these structures show that $c_1^*(P) > 1$ for all polygons.

3 Arbitrary Folds

3.1 Simply connected shapes

In this section we show that, for a simply connected shape S , there is a lower bound for the origami cover factor $c^*(S)$ that is within a constant factor of the upper bound given by Lemma 2.

Theorem 8 *Let S be a simply connected shape with inradius r , geodesic radius R , and geodesic diameter D . Then $\kappa R/r \leq D/(2\pi r) \leq c^*(S) \leq R/r$ for $\kappa = \sqrt{3}/(2\pi) \approx 0.27566$.*

Proof. Again, the upper bound is from Lemma 2. The basic idea for the lower bound is to find a path in S that can be folded into a large circle, which must then be covered by a scaled copy of the incircle of S . Here, for brevity, we use a path of length D , the geodesic diameter.

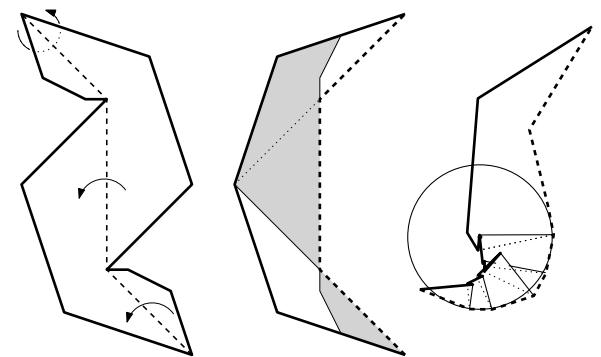


Figure 6: For Theorem 8, folding inflection edges to make a generalized spiral, then crimping to approximate a circle that must be covered by the incircle.

A *generalized spiral* is a simply connected region composed of consistently orientable plane patches having a distinguished shortest path γ that follows the boundary and never turns to the left. A generalized spiral may overlap itself if projected onto a plane, but we can think of it as embedded in a covering space of the plane.

Ordinarily, a diameter path γ will alternate between sequences of left turns and right turns at boundary points; a portion of the path between opposite turns is a line segment that we can call an *inflection edge*. We can simply fold along every inflection edge, gluing doubled layers along these edges, to turn γ into a path that goes only straight or to the right. Folding any non-boundary

edges creates a generalized spiral with path γ . These folds are along lines of the geodesic path, so γ remains a shortest path between its endpoints.

We fold the generalized spiral into a left-turning circle with circumference approaching the length of γ . If we sweep a paired point and normal vector along γ , we can think of painting a portion of the generalized spiral with fibers that each start on γ and grow orthogonal to a local tangent (because γ is a shortest path) and that are disjoint (because the sweep in position and angle is monotonic). We construct a circle whose circumference is arbitrarily close to the length of γ by crimp folds that align successive fibers of γ with the circle center. Figure 6 shows an example. It does not matter how far the fibers extend towards or beyond the circle; in order to cover the boundary of the circle, the inradius r must be scaled to the circle radius, which is $D/(2\pi)$. \square

3.2 Disks with bumps

Because the radius of a disk is simultaneously the inradius and the geodesic radius, Lemma 2 implies that the cover factor of a disk, $c^*(\bigcirc)$, is 1. It is interesting to note that there are other shapes S with $c^*(S) = 1$; here is one simple family.

In a unit disk centered at C with a chord AB , choose a point D between C and the midpoint of AB . Add the disk centered at D of radius $|AD|$. Thus, we have a family of shapes $S_{d,e}$, parameterized by two distances, $d = |CD|$ and $e = \text{distance from } C \text{ to chord } AB$, satisfying $0 < d \leq e < 1$. See Figure 7.

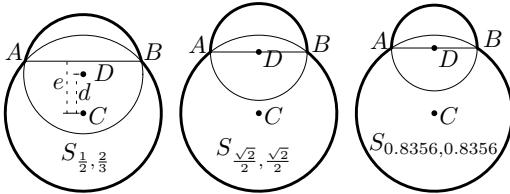


Figure 7: Shapes $S_{d,e}$ with $c^*(S_{d,e}) = 1$.

Lemma 9 *The shape $S_{d,e}$, with $0 < d \leq e < 1$, has origami cover factor $c^*(S_{d,e}) = 1$.*

Proof. Shape $S_{d,e}$ is the union of a unit disk centered at C and a disk centered at D whose radius we denote r . Note that by construction the boundaries of the disks intersect at A and B . This shape also covers all disks of radius r that are centered between C and D .

Now, in an arbitrary folded state $S'_{d,e}$, consider the locations of these centers, C' and D' . Placing a unit disk centered at C' and a radius r disk centered at D' will cover all points of $S'_{d,e}$. Because $|C'D'| \leq |CD|$, this pair of disks will be covered by placing a copy of $S_{d,e}$ with C at C' and D on the ray $C'D'$. \square

Choose any $d \in (0, 1)$ and for all $e \in [d, 1)$ shape $S_{d,d}$ covers $S_{d,e}$, so these extremal members of the family have AB as the diameter of the smaller disk. Just for the sake of curiosity, the example with $d = e = \sqrt{2}/2$ minimizes the ratio of inradius to circumradius, $R/r = (1 + \sin \theta + \cos \theta)/2 \approx 0.8284$, and the example with $d \approx 0.8356$ minimizes the fraction of the circumcircle covered, $(\pi(1 + \sin^2 \theta) + \sin 2\theta - \theta)/(\pi R^2) \approx 0.7819$.

4 Open Problems

The most interesting questions are whether $c^*(\triangle) = c_1^*(\triangle)$ and $c^*(\square) = c_1^*(\square)$, and whether we can completely characterize those shapes with origami or 1-fold cover factor of unity.

Acknowledgments: This work began at the 28th Bellairs Workshop, March 22-29, 2013. We thank all other participants for the productive and positive atmosphere, in particular Godfried Toussaint for co-organizing the event. We thank the CCCG reviewers for detailed and helpful comments.

References

- [1] V. I. Arnold. Problem 1956–1. In *Arnold's Problems*. Springer, Berlin, 2005.
- [2] P. Brass, W. O. J. Moser, and J. Pach. *Research Problems in Discrete Geometry*. Springer, 2005.
- [3] P. Brass and M. Sharifi. A lower bound for Lebesgue's universal cover problem. *Int. J. Comput. Geometry Appl.*, 15(5):537–544, 2005.
- [4] S. Das, P. P. Goswami, and S. C. Nandy. Smallest k -point enclosing rectangle and square of arbitrary orientation. *Inform. Process. Lett.*, 94(6):259–266, 2005.
- [5] N. A. A. DePano and A. Aggarwal. Finding restricted k -envelopes for convex polygons. In *Proc. 22nd Allerton Conf. on Comm., Ctrl, & Comp*, pages 81–90, 1984.
- [6] T. Khovanova and A. Radul. Jewish problems. <http://arxiv.org/abs/1110.1556v2>, Oct. 2011.
- [7] R. J. Lang. *Origami Design Secrets: Mathematical Methods for an Ancient Art*. A. K. Peters, 2003.
- [8] X. Markenscoff, L. Ni, and C. H. Papadimitriou. The geometry of grasping. *Internat. J. Robot. Res.*, 9(1), Feb. 1990.
- [9] R. Norwood, G. Poole, and M. Laidacker. The worm problem of Leo Moser. *Discrete & Computational Geometry*, 7:153–162, 1992.
- [10] J. Pál. Über ein elementares Variationsproblem. *Math.-fys. Medd., Danske Vid. Selsk.*, 3(2):1–35, 1920.
- [11] C. Panraksa, J. E. Wetzel, and W. Wichiramala. Covering n -segment unit arcs is not sufficient. *Discrete & Computational Geometry*, 37(2):297–299, 2007.
- [12] H. Rademacher and O. Toeplitz. *The Enjoyment of Mathematics*. Dover, 1990.

Unfolding Face-Neighborhood Convex Patches: Counterexamples and Positive Results

Joseph O'Rourke*

Abstract

We address unsolved problems of unfolding polyhedra in a new context, focusing on special *convex patches*—disk-like polyhedral subsets of the surface of a convex polyhedron. One long-unsolved problem is edge-unfolding *prismatoids*. We show that several natural strategies for unfolding a prismatoid can fail, but obtain a positive result for “petal unfolding” topless prismatoids, which can be viewed as particular convex patches. We also show that the natural extension of an earlier result on face-neighborhood convex patches fails, but we obtain a positive result for nonobtusely triangulated face-neighborhoods.

1 Introduction

Define a *convex patch* as a connected subset of faces of a convex polyhedron \mathcal{P} , homeomorphic to a disk. A convex patch is convexly curved in 3D, but its boundary need not be convex: it could be quite “jagged.” I propose studying edge-unfolding of convex patches to simple (non-overlapping) polygons in the plane, as presumably easier versions of the many unsolved convex-polyhedron unfolding problems. (Here, *edge-unfolding* cuts only edges of \mathcal{P} ; we leave that understood until the final discussion.) Toward this end, I study here special convex patches, various *face-neighborhoods*, and obtain several positive and negative results.

Face Neighborhoods. Let F be a face of a convex polyhedron \mathcal{P} . There are two natural “face-neighborhoods” of F : the *edge-neighborhood* $N_e(F)$, F together with every face of \mathcal{P} that shares an edge with F , and the *vertex-neighborhood* $N_v(F)$, F together with every face incident to a vertex of F .¹ Clearly, $N_v(F) \supseteq N_e(F)$. A “dome” polyhedron \mathcal{P} is one with a “base face” B such that $N_e(B) = \mathcal{P}$. Domes were earlier proved to unfold without overlap [6, p. 323ff]. Pincu [12] subsequently proved that $N_e(F)$ unfolds without overlap for any F , generalizing the dome result. Both the

dome and the edge-neighborhood unfoldings are what I am now calling “petal unfoldings,” described next in the context of prismatoids.

Prismatoids and Prismoids. A *prismatoid* is the convex hull of two convex polygons A (above) and B (base), that lie in parallel planes. Despite its simple structure, it remains unknown whether or not every prismatoid has a non-overlapping edge-unfolding, a narrow special case of what has become known as Dürer’s Problem: whether every convex polyhedron has a non-overlapping edge-unfolding [6, Prob. 21.1] [11].

If A and B are angularly similar with their edges parallel, then all lateral faces are trapezoids. Such a polyhedron is called a *prismoid*. These special prismatoids are known to edge-unfold without overlap [6, p. 322].

Band and Petal Unfoldings. There are two natural unfoldings of a prismatoid. A *band unfolding* cuts one lateral edge and unfolds all lateral faces connected in *band*, leaving A and B attached each by one uncut edge to opposite sides of the band (see, e.g., [2]). Aloupis showed that the lateral cut-edge can be chosen so that the band alone unfolds [1], but I showed that, nevertheless, there are prismoids such that every band unfolding overlaps [8]. The example, Fig. 1, is repeated here, as it plays a role in the closing discussion (Sec. 4). Note

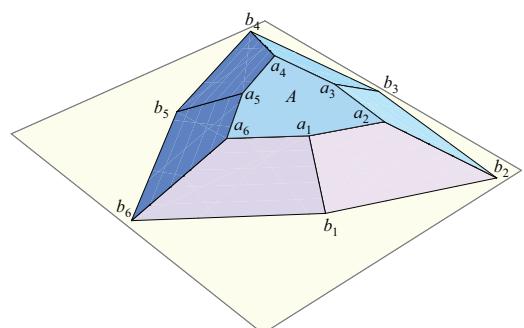


Figure 1: A convex patch with no band unfolding.

that this example also establishes that not every edge-neighborhood patch of a face of \mathcal{P} has a band unfolding: $N_e(A)$ has no band unfolding.

The second natural unfolding of a prismatoid is a

*Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu. This is a revision of [10]. Omitted proofs are in the full version.

¹This is my own terminology. $N_e(F)$ is called the “face-neighborhood” in [7].

*petal unfolding.*² The three positive results mentioned above are all via petal unfoldings: the dome unfolding, the prismoid unfolding, and Pincu’s edge-neighborhood patch unfolding. Thus Fig. 1 without its base, which is a edge-neighborhood patch, can be petal-unfolded: simply cut each lateral edge $a_i b_i$. We henceforth concentrate on petal unfoldings (until the final Sec. 4).

New Results. Given the collection of partial results and unsolved problems reviewed above, it is natural to explore petal unfoldings of vertex-neighborhood patches. Our results are as follows:

1. Define a *topless prismaticoid* as one with A removed; so it is a special (non-jagged) vertex-neighborhood $N_v(B)$. We prove that every topless prismaticoid whose lateral faces are triangles has a petal unfolding without overlap (Thm. 7). This shows that, in some sense, placing the top A is an obstruction to unfolding prismaticoids.
2. Via a counterexample polyhedron \mathcal{P} (Fig. 8), we show that not every vertex-neighborhood patch $N_v(F)$ has a non-overlapping petal unfolding.
3. However, if \mathcal{P} is non-obtusely triangulated, $N_v(F)$ does have a non-overlapping petal unfolding for every face of \mathcal{P} (Thm. 8).
4. This leads to a non-overlapping unfolding of a restricted class of prismaticoids (Cor. 9).

I am hopeful that the main proof technique—obtaining a result for flat patches and then lifting into $z > 0$ —will lead to further results.

We conclude in Section 4 with a conjecture that not every edge-neighborhood has a non-overlapping “zipper unfolding.”

2 Topless Prismaticoid Petal Unfolding

Let \mathcal{P} be a prismaticoid, and assume all lateral faces are triangles, the generic and seemingly most difficult case. Let $A = (a_1, a_2, \dots)$ and $B = (b_1, b_2, \dots)$. Call a lateral face that shares an edge with B a *base* or *B-triangle*, and a lateral face that shares an edge with A a *top* or *A-triangle*. A petal unfolding cuts no edge of B , and unfolds every base triangle by rotating it around its B -edge into the base plane. The collection of *A*-triangles incident to the same b_i vertex—the *A-fan* AF_i —must be partitioned into two groups, one of which rotates clockwise (cw) to join with the unfolded base triangle to its left, and the other group rotating counterclockwise (ccw) to join with the unfolded base triangle to its right. Either group could be empty. Finally, the top A is attached to one *A*-triangle. So a petal unfolding has

choices for how to arrange the *A*-triangles, and which *A*-triangle connects to the top.

It remains possible that every prismaticoid has a petal unfolding: so far I have not been able to find a counterexample. Now we turn to our main result: every topless prismaticoid has a petal unfolding. An example of a petal unfolding of a topless prismaticoid is shown in Fig. 2.

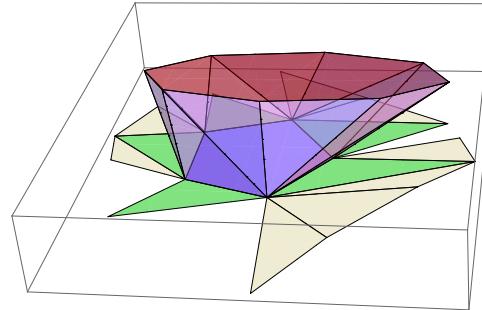


Figure 2: Unfolding of a topless prismaticoid. *A*-fans are lightly shaded.

Even topless prismaticoids present challenges. For example, consider the special case when there is only one *A*-triangle between every two *B*-triangles. Then the only choice for placement of the *A*-triangles is whether to turn each ccw or cw. It is natural to hope that rotating all *A*-triangles consistently ccw or cw suffices to avoid overlap, but this can fail. A more nuanced approach would turn each *A*-triangle so that its (at most one) obtuse angle is not joined to a *B*-triangle, but this can fail also.

The proof that topless prismaticoids have petal unfoldings follows this outline:

1. An “altitudes partition” of the plane exterior to the *base unfolding* (petal unfolding of $N_e(B)$) is defined and proved to be a partition.
2. It is shown that both \mathcal{P} and this partition vary in a consistent manner with respect to the separation z between the *A*- and *B*-planes.
3. An algorithm is detailed for petal unfolding the *A*-triangles for the “flat prismaticoid” $\mathcal{P}(0)$, the limit of $\mathcal{P}(z)$ as $z \rightarrow 0$, such that these *A*-triangles fit inside the regions of the altitude partition.
4. It is proved that nesting within the partition regions remains true for all z .

2.1 Altitude Partition

We use a_i and b_j to represent the vertices of \mathcal{P} , and primes to indicate unfolded images on the base plane.

Let $B_i = \Delta b_i b_{i+1} a'_j$ be the i -th base triangle. Say that $B^\cup = B \cup (\bigcup_i B_i)$ is the *base unfolding*, the petal

²Called a “volcano unfolding” in [6, p. 321].

unfolding of $N_e(B)$ without any A -triangles. The altitude partition partitions the plane exterior to B^\cup .

Let r_i be the *altitude ray* from a'_j along the altitude of B_i . Finally, define R_i to be the region of the plane incident to b_i , including the edges of the B_{i-1} and B_i triangles incident to b_i , and bounded by r_{i-1} and r_i . See Fig. 3.

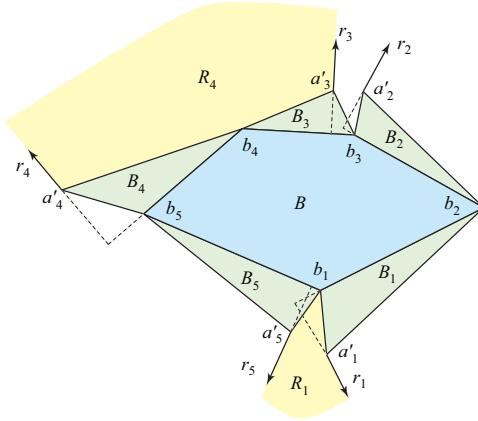


Figure 3: Partition exterior to B^\cup by altitude rays r_i . Here both A and B are pentagons; in general there would not be synchronization between the b_i and a_i indices. The A -triangles are not shown.

Lemma 1 *No pair of altitude rays cross in the base plane, and so they define a partition of that plane exterior to the base unfolding B^\cup .*

Our goal is to show that the A -fan AF_i incident to b_i can be partitioned into two groups, one rotated cw, one ccw, so that both fit inside R_i .

2.2 Behavior of $\mathcal{P}(z)$

We will use “(z)” to indicate that a quantity varies with respect to the height z separating the A - and B -planes.

Lemma 2 *Let $\mathcal{P}(z)$ be a prismatic with height z . Then the combinatorial structure of $\mathcal{P}(z)$ is independent of z , i.e., raising or lowering A above B retains the convex hull structure.*

We will call $\mathcal{P}(0) = \lim_{z \rightarrow 0} \mathcal{P}(z)$ a *flat prismatic*. Each lateral face of $\mathcal{P}(0)$ is either an *up-face* or a *down-face*, and the faces of $\mathcal{P}(z)$ retain this classification in that their outward normals either have a positive or a negative vertical component.

Lemma 3 *Let $\mathcal{P}(z)$ be a prismatic with height z , and $B^\cup(z)$ its base unfolding. Then the apex $a'_j(z)$ of each $B'_i(z)$ triangle $\Delta b_i b_{i+1} a'_j(z)$ in $B^\cup(z)$ lies on the fixed line containing the altitude of $B'_i(z)$.*

Thus the vertices $a'_j(z)$ of the base unfolding “ride out” along the altitude rays r_i as z increases (see ahead to Fig. 6 for an illustration). Therefore the combinatorial structure of the altitude partition is fixed, and R_i only changes geometrically by the lengthening of the edges $b_i a'_j$ and $b_{i+1} a'_j$ and the change in the angle gap $\kappa_{b_i}(z)$ at b_i .

2.3 Structure of A -fans

Henceforth we concentrate on one A -fan, which we always take to be incident to b_2 , and so between $B_1 = \Delta b_1 b_2 a_1$ and $B_2 = \Delta b_2 b_3 a_k$. The *a-chain* is the chain of vertices a_1, \dots, a_k . Note that the plane in \mathbb{R}^3 containing face B_1 of \mathcal{P} supports A at a_1 , and the plane containing B_2 supports A at a_k . Let $\beta = \beta_2$ be the base angle at b_2 : $\beta = \angle b_1 b_2 b_3$. We state here a few facts true of every A -fan.

1. An *a-chain* spans at most “half” of A , i.e., a portion between parallel supporting lines to A (because $\beta > 0$).
2. If an A -fan is unfolded as a unit to the base plane, the *a-chain* consists of convex, reflex, and convex portions, any of which may be empty. So, excluding the first and last vertices, the interior vertices of the chain have convex angles, then reflex, then convex.
3. Correspondingly, an A -fan consists of down-faces followed by up-faces followed by down-faces, where again any (or all) of these three portions could be empty.
4. All four possible combinations of up/down are possible for the B_1 and B_2 triangles.

The second fact above is not so easy to see. The intuition is that there is a limited amount of variation possible in an *a-chain*. It is the third fact that we will use essentially; it will become clear shortly.

2.4 Flat Prismatoid Case Analysis

How the A -fan is proved to sit inside its altitude region R for $\mathcal{P}(0)$ depends primarily on where b_2 sits with respect to A , and secondarily on the three B -vertices (b_1, b_2, b_3). Fig. 4 illustrates one of the easiest cases, when b_2 is in C , the convex region bounded by the *a-chain* and extensions of its extreme edges. Then all the A -faces are down-faces, the *a-chain* is convex, one of the two B -faces is a down-face (B_2 in the illustration), and we simply leave the A -fan attached to that B down-face.

A second case occurs when b_2 is on the reflex side of A . An instance when both B -triangles are down-faces is illustrated in Fig. 5. Now the A -fan consists of down-faces and up-faces, the up-faces incident to the reflex side of the *a-chain*. These up-faces must be flipped in

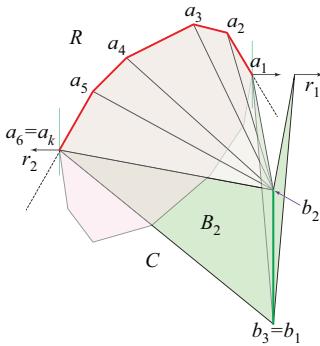


Figure 4: Case 1b. Here we have illustrated $b_1 = b_3$ to allow for the maximum a -chain extent.

the unfolding, reflected across one of the two tangents from b_2 to A . A key point is that not always will both flips be “safe” in the sense that they stay inside the altitude region. Fortunately, one of the two flips is always safe:

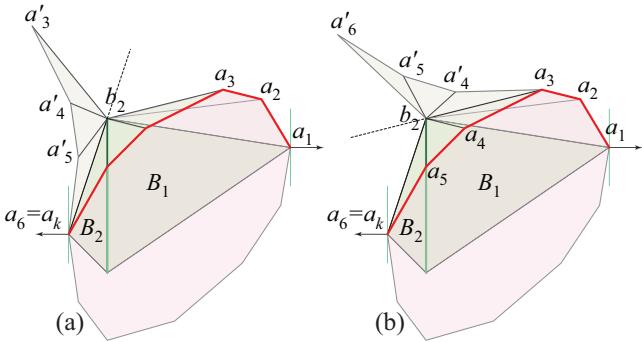


Figure 5: Case 2a. The A -triangles between the tangents b_2 to a_3 and b_2 to a_6 are up-faces. (a) shows the up-faces flipped over the left tangent b_2a_6 , and (b) when flipped over the right tangent b_2a_3 .

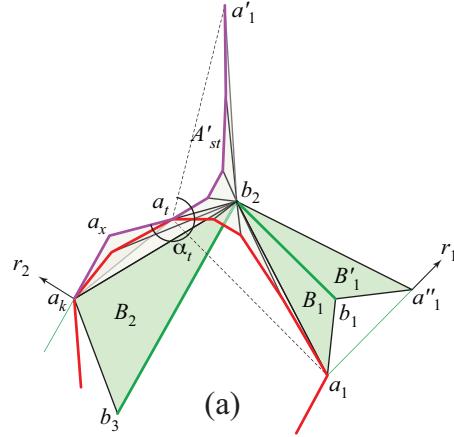
Lemma 4 Let b_2 have tangents touching a_s and a_t of A . Then either reflecting the enclosed up-faces across the left tangent, or across the right tangent, is “safe” in the sense that no points of a flipped triangle falls outside the rays r_1 or r_k .

The remaining cases are minor variations on those illustrated, and will not be further detailed.

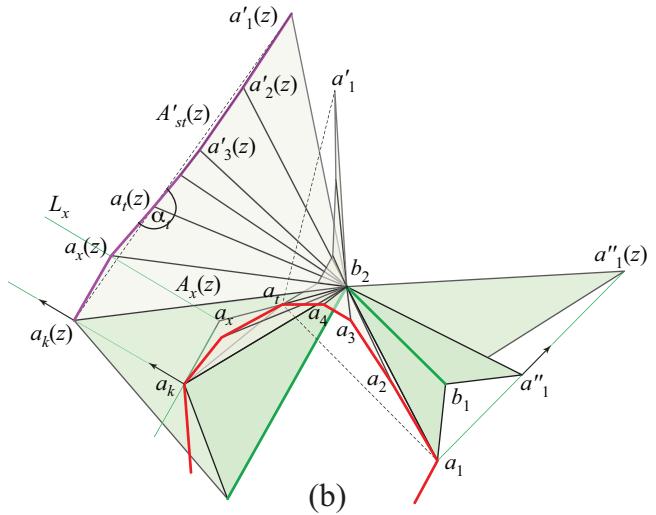
2.5 Nesting in $\mathcal{P}(z)$ regions

The most difficult part of the proof is showing that the nesting established above for $\mathcal{P}(0)$ holds for $\mathcal{P}(z)$. A key technical lemma is this:

Lemma 5 Let $\triangle b, a_1(z), a_2(z)$ be an A -triangle, with angles $\alpha_1(z)$ and $\alpha_2(z)$ at $a_1(z)$ and $a_2(z)$ respectively. Then $\alpha_1(z)$ and $\alpha_2(z)$ are monotonic from their $z = 0$ values toward $\pi/2$ as $z \rightarrow \infty$.



(a)



(b)

Figure 6: (a) $z = 0$. $\triangle a_t a_x a_k$ encloses the convex section, and $\triangle a_1 b_2 a_t$ encloses the reflex section. (b) $z > 0$. Reflex angle $\alpha_t(z)$ decreases as z increases.

I should note that it is not true, as one might hope, that the apex angle at b of that A -triangle, $\angle a_1(z), b, a_2(z)$, shrinks monotonically with increasing z , even though its limit as $z \rightarrow \infty$ is zero. Nor is the angle gap $\kappa_b(z)$ necessarily monotonic. These nonmonotonic angle variations complicate the proof.

Another important observation is that the sorting of ba_i edges by length in $\mathcal{P}(0)$ remains the same for all $\mathcal{P}(z)$, $z > 0$. More precisely, let $|ba_i| > |ba_j|$ for two lateral edges connecting vertex $b \in B$ to vertices $a_i, a_j \in A$ in $\mathcal{P}(0)$. Then $|ba_i(z)| > |ba_j(z)|$ remains true for all $\mathcal{P}(z)$, $z > 0$.

For the nesting proof, I will rely on a high-level description, and one difficult instance. At a high level, each of the convex or reflex sections of the a -chain are enclosed in a triangle, which continues to enclose that portion of the a -chain for any $z > 0$ (by Fact 1, Sec. ??). The reflex enclosure is determined by the tangents from b_2 to A : $\triangle a_s b_2 a_t$. So then the task is to prove these (at most three) triangles remain within $R(z)$. Fig. 6 shows

a case where there is both a convex and a reflex section. Were there an additional convex section, it would remain attached to $B_1(z)$ and would not increase the challenge.

Lemma 6 *If the a -chain consists of a convex and a reflex section, and the safe flip (by Lemma 4) is to a side with a down-face (B_2 in the figure), then $AF'(z) \subset R(z)$: the A -fan unfolds within the altitude region for all z .*

I have been unsuccessful in unifying the cases in the analysis, despite their similarity. Nevertheless, the conclusion is this theorem:

Theorem 7 *Every triangulated topless prismatoid has a petal unfolding.*

It is natural to hope that further analysis will lead to a safe placement of the top A (which, alas, might not fit into any altitude-ray region).

3 Unfolding Vertex-Neighborhoods

We now return to arbitrary face-neighborhoods. As mentioned previously, Pincu proved that the petal unfolding of $N_e(B)$ avoids overlap for any face B of a convex polyhedron. Here we show that the vertex-neighborhood $N_v(B)$ does not always have a non-overlapping petal unfolding, even when all faces in the neighborhood are triangles.

A portion of the a 9-vertex example \mathcal{P} that establishes this negative result is shown in Fig. 7. The b_1b_3 edge of B lies on the horizontal xy -plane. The vertices $\{b_2, a_1, a_2, c_1, c_2\}$ all lie on a parallel plane at height z , with b_2 directly above the origin: $b_2 = (0, 0, z)$.

All of $N_v(B)$ is shown in Fig. 8. The structure in Fig. 7 is surrounded by more faces designed to minimize curvatures at the vertices b_i of B . Finally, \mathcal{P} is the convex hull of the illustrated vertices, which just adds a quadrilateral “back” face (p_1, c_1, c_2, p_3) (not shown).

The design is such that there is so little rotation possible in the cw and ccw options for the triangles incident to vertex b_2 of B , that overlap is forced: see Figs. 9, 10, and 11. The thin $\triangle b_2a_1a_2$ overlaps in the vicinity of a_1 if rotated ccw, and in the vicinity of a_2 is cw (illustrated).

One can identify two features of the polyhedron just described that lead to overlap: low curvature vertices (to restrict freedom) and obtuse face angles (at a_1 and a_2) (to create “overhang”). Both seem necessary ingredients. Here I pursue excluding obtuse angles:

Theorem 8 *If \mathcal{P} is nonobtusely triangulated, then for every face B , $N_v(B)$ has a petal unfolding.*

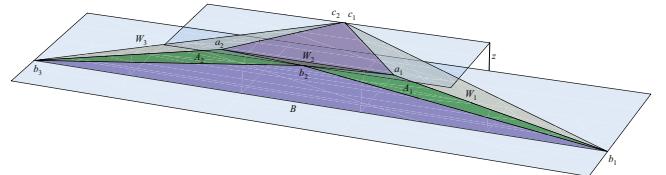


Figure 7: Faces of \mathcal{P} in the immediate vicinity of B .

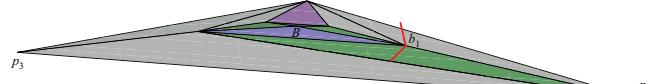


Figure 8: All faces incident to $N_v(B)$, and one more, the purple quadrilateral (a_1, c_1, c_2, a_2) . The red vectors are normal to B and to $\triangle b_1p_1c_1$.

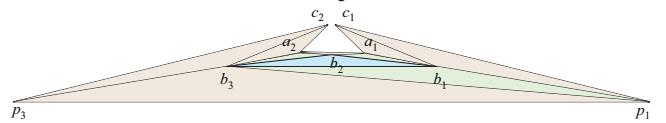


Figure 9: Complete unfolding of all faces incident to B .

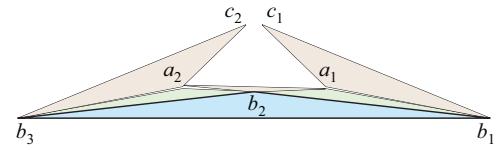


Figure 10: Zoom of Fig. 9.

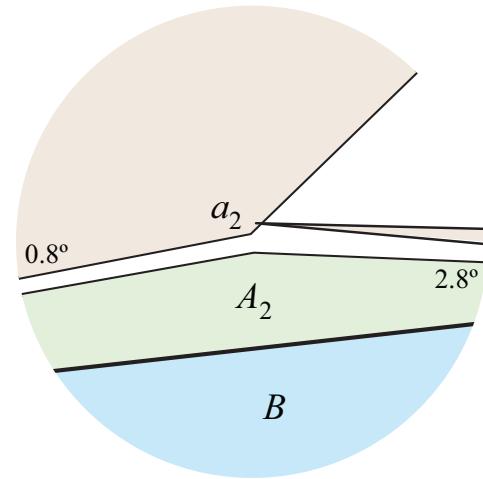


Figure 11: Zoom of Fig. 10 in vicinity of a_2 overlap. The angle gap at b_3 is 0.8° , and the gap at b_2 is 2.8° .

A *nonobtuse triangle* is one whose angles are each $\leq \pi/2$. It is known that any polygon of n vertices has a nonobtuse triangulation by $O(n)$ triangles, which can be found in $O(n \log^2 n)$ time [3]. Open Problem 22.6 [6, p. 332] asked whether every nonobtusely triangulated convex polyhedron has an edge-unfolding. One can view Theorem 8 as a (very small) advance on this problem.³

A little more analysis leads to a petal unfolding of a (very special) class of prismatoids (including their tops):

Corollary 9 *Let \mathcal{P} be a triangular prismatoid all of whose faces, except possibly the base B , are nonobtuse triangles, and the base is a (possibly obtuse) triangle. Then every petal unfolding of \mathcal{P} avoids overlap.*

It seems quite possible that this corollary still holds with B an arbitrary convex polygon, but the proof would need significant extension.

4 Discussion

I believe that unfolding convex patches may be a fruitful line of investigation. For example, notice that the edges cut in a petal unfolding of a vertex-neighborhood of a face form a disconnected spanning forest rather than a single spanning tree. One might ask: Does every convex patch have an edge-unfolding via a single spanning cut tree? The answer is NO, already provided by the banded hexagon example in Fig. 1. For such a tree can only touch the boundary at one vertex (otherwise it would lead to more than one piece), and then it is easy to run through the few possible spanning trees and show they all overlap.

The term *zipper unfolding* was introduced in [5] for a non-overlapping unfolding of a convex polyhedron achieved via Hamiltonian cut path. They studied zipper edge-paths, following edges of the polyhedron, but raised the interesting question of whether or not every convex polyhedron has a zipper path, not constrained to follow edges, that leads to a non-overlapping unfolding. This is a special case of Open Problem 22.3 in [6, p. 321] and still seems difficult to resolve.

Given the focus of this work, it is natural to specialize this question further, to ask if every convex patch has a zipper unfolding, using arbitrary cuts (not restricted to edges). I believe the answer is negative: a version of the banded hexagon shown in Fig. 12, a bottomless prismoid, has no zipper unfolding. My argument for this is long and seems difficult to formalize, so I leave the claim as a conjecture. It would constitute an interesting contrast to the recent result that all “nested” prismoids have a zipper edge-unfolding [4].

³It can also be used to slightly improve Pincu’s “fewest nets” result for this class of polyhedra.

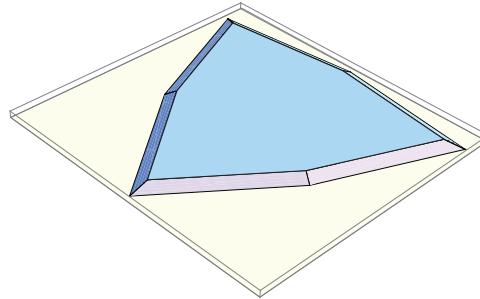


Figure 12: The banded hexagon with a thin band.

References

- [1] G. Aloupis. *Reconfigurations of Polygonal Structures*. PhD thesis, McGill Univ., Sch. Comput. Sci., 2005.
- [2] G. Aloupis, E. D. Demaine, S. Langermann, P. Morin, J. O’Rourke, I. Streinu, and G. Toussaint. Edge-unfolding nested polyhedral bands. *Comput. Geom. Theory Appl.*, 39(1):30–42, 2007.
- [3] M. W. Bern, S. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. *Discrete Comput. Geom.*, 14:411–428, 1995.
- [4] E. Demaine, M. Demaine, and R. Uehara. Zipper unfoldability of domes and prismoids. In *Proc. 25th Canad. Conf. Comput. Geom.*, Aug. 2013.
- [5] E. Demaine, M. Demaine, A. Lubiw, A. Shallit, and J. Shallit. Zipper unfoldings of polyhedral complexes. In *Proc. 22nd Canad. Conf. Comput. Geom.*, pages 219–222, Aug. 2010.
- [6] E. D. Demaine and J. O’Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007. <http://www.gfalop.org>.
- [7] H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack. Shortest path queries between geometric objects on surfaces. In *Comput. Sci. Appl.–ICCSA 2007*, pages 82–95. Springer, 2007.
- [8] J. O’Rourke. Band unfoldings and prismatoids: A counterexample. Technical Report 087, Smith College, Oct. 2007. arXiv:0710.0811v2 [cs.CG]; <http://arxiv.org/abs/0710.0811>.
- [9] J. O’Rourke. Tetrahedron angles sum to π : Bisector plane. MathOverflow: <http://mathoverflow.net/questions/94586/>, Apr. 2012.
- [10] J. O’Rourke. Unfolding prismatoids as convex patches: Counterexamples and positive results. arXiv:1205.2048 [cs.CG]. <http://arxiv.org/abs/1205.2048>, 2012.
- [11] J. O’Rourke. Dürer’s problem. In M. Senechal, editor, *Shaping Space: Exploring Polyhedra in Nature, Art, and the Geometrical Imagination*, pages 77–86. Springer, 2013.
- [12] V. Pincu. On the fewest nets problem for convex polyhedra. In *Proc. 19th Canad. Conf. Comput. Geom.*, pages 21–24, 2007.

Counting Triangulations Approximately

Victor Alvarez*

Karl Bringmann†

Saurabh Ray‡

Raimund Seidel§

Abstract

We consider the problem of counting straight-edge triangulations of a given set P of n points in the plane. Until very recently it was not known whether the *exact* number of triangulations of P can be computed asymptotically faster than by enumerating all triangulations. We now know that the number of triangulations of P can be computed in $O^*(2^n)$ time [2], which is less than the lower bound of $\Omega(2.43^n)$ on the number of triangulations of *any* point set [11]. In this paper we address the question of whether one can approximately count triangulations in sub-exponential time. We present an algorithm with sub-exponential running time and sub-exponential approximation ratio, that is, if we denote by Λ the output of our algorithm, and by c^n the exact number of triangulations of P , for some positive constant c , we prove that $c^n \leq \Lambda \leq c^n \cdot 2^{o(n)}$. This is the first algorithm that in sub-exponential time computes a $(1 + o(1))$ -approximation of the base of the number of triangulations, more precisely, $c \leq \Lambda^{\frac{1}{n}} \leq (1 + o(1))c$. Our algorithm can be adapted to approximately count other crossing-free structures on P , keeping the quality of approximation and running time intact. The algorithm may be useful in guessing, through experiments, the right constants c_1 and c_2 such that the number of triangulations of any set of n points is between c_1^n and c_2^n . Currently there is a large gap between c_1 and c_2 , we know that $c_1 \geq 2.43$ and $c_2 \leq 30$.

1 Introduction

Let P be a set of n points on the plane. A crossing-free structure on P is a straight-line plane graph with vertex set P . Examples of crossing-free structures include triangulations and spanning cycles, also known as polygonizations, among others. Let \mathcal{C} denote a certain class of crossing-free structures, and let $\mathcal{F}_{\mathcal{C}}(P)$ denote the set of all crossing-free structures on P belonging to class \mathcal{C} .

*Fachrichtung Informatik, Universität des Saarlandes, Germany, alvarez@cs.uni-saarland.de.

†Max Planck Institute for Informatics, Karl Bringmann is a recipient of the *Google Europe Fellowship in Randomized Algorithms*, and this research is supported in part by this Google Fellowship. kbringma@mpi-inf.mpg.de.

‡Ben Gurion University of the Negev, Israel saurabh@math.bgu.ac.il.

§Fachrichtung Informatik, Universität des Saarlandes, Germany, rseidel@cs.uni-saarland.de.

Recently, there has been a significant amount of work regarding the question: “Can the cardinality of $\mathcal{F}_{\mathcal{C}}(P)$ be computed faster than by enumerating $\mathcal{F}_{\mathcal{C}}(P)$?”.

In this paper we focus on the particular class \mathcal{C} of all straight-edge triangulations. To the best of our knowledge the first result regarding this question was by O. Aichholzer in '99 [1], where he introduced a geometric structure called “the path of a triangulation”, or T-path for short. Using T-paths he showed a divide-and-conquer algorithm that experimentally indicated that triangulations could be counted in time sub-linear in the number of triangulations, that is, the algorithm was apparently faster than enumeration. However, a formal proof of this - or even a good analysis of its running time - seems hard to obtain, since it is not clear how to bound the number of triangulations containing a single T-path. Later, in '05, S. Ray and R. Seidel [8] presented a new algorithm for counting triangulations that, in practice, appeared to be substantially faster than Aichholzer's algorithm. This algorithm is also very hard to analyze, and thus no good analysis of its running time has been presented so far. It took until '12 for new counting algorithms to come up that could be analyzed properly. The first such algorithm is also based on T-paths but uses the sweep-line paradigm [4]. This algorithm was proven to count triangulations in time $O^*(9^n)$. The second algorithm, based on the onion layers of P and the divide-and-conquer paradigm, was proven to count triangulations in time $O^*(3.1414^n)$ [3]. From the experimental point of view, the second algorithm turned out to be significantly faster, for certain configurations of points, than the one shown in [8]. These experiments can be found in the full version of [3] available on the authors' websites. The third, and so far the latest, algorithm for counting triangulations runs in time $O^*(2^n)$ [2]. This last algorithm finally shows that enumeration algorithms for triangulations can indeed *always* be beaten, as all point sets with n points have at least $\Omega(2.43^n)$ triangulations [11]. From an experimental point of view it was also shown to be significantly faster than all previous algorithms on a variety of inputs [2].

1.1 Our Contribution

The $O^*(2^n)$ algorithm of [2] for counting triangulations *exactly* seems hard to beat at this point. However, it would be very interesting to see whether the number

of triangulations of P can be *approximated*. The result presented in this paper is, to the best of our knowledge, the first result in this new line of research.

Note that since for all sets of n points the number of triangulations is $\Omega(2.43^n)$ [11] and $O(30^n)$ [10], the quantity $\Theta(\sqrt{30 \times 2.43^n})$ approximates the exact number of triangulations within a factor of $O(\sqrt{30/2.43^n})$. Thus, one can trade the exponential time of an exact algorithm for a polynomial time algorithm with exponential approximation ratio. In this paper we bridge the gap between these two solutions by presenting an algorithm with sub-exponential running time and sub-exponential approximation ratio.

Let $\mathcal{F}_T(P)$ denote the set of all triangulations of P . The main result of this paper, whose proof is shown in § 4, is the following:

Theorem 1 *Let P be a set of n points on the plane. Then a number Λ can be computed in time $2^{o(n)}$ such that $|\mathcal{F}_T(P)| \leq \Lambda \leq |\mathcal{F}_T(P)|^{1+o(1)} = 2^{o(n)}|\mathcal{F}_T(P)|$.*

The precise $o(n)$ terms mentioned in Theorem 1 are $O(\sqrt{n} \log(n))$ for the running time and $O\left(n^{\frac{3}{4}} \sqrt{\log(n)}\right)$ for the approximation factor. At the end of § 4 we mention a trade-off between these two, running time and approximation factor.

While the approximation factor of Λ is rather big, the computed value is of the same order of magnitude as the correct value of $|\mathcal{F}_T(P)|$, that is, we compute a $(1+o(1))$ -approximation of the base of the exponentially large value $|\mathcal{F}_T(P)|$. More precisely, for $|\mathcal{F}_T(P)| = c^n$ we have $c \leq \Lambda^{\frac{1}{n}} \leq c^{1+o(1)} \leq (1+o(1))c$. Also, this approximation can be computed in sub-exponential time, which, at least theoretically, is asymptotically faster than the worst-case running times of the algorithms presented in [4, 3, 2]. This is certainly very appealing.

2 Preliminaries

Our algorithm uses simple cycle separators as the main ingredient, originally presented in [7] by G. L. Miller, and improved in [5] by H. N. Djidjev and S. M. Venkatesan. The following theorem accounts for both results:

Theorem 2 (Separator Theorem) *Let T be a triangulation of a set of n points in the plane such that the unbounded face is a triangle. Then there exists a simple cycle C of size at most $\sqrt{4n}$, that separates the set A of vertices of T in its interior from the set B of vertices of T in its exterior, such that the number of elements of each one of A and B is always at most $\frac{2n}{3}$.*

Observe that the Separator Theorem does not imply that *every* triangulation of a set of points contains a *unique* simple cycle separator. One can easily come up

with examples in which a triangulation contains more than one simple cycle separator. The important part here is that *every* triangulation contains *at least* one simple cycle separator.

3 The Algorithm

The idea for an approximate counting algorithm is suggested by the Separator Theorem: We enumerate all possible simple cycle separators C of size at most $\sqrt{4n}$ that we can find in the given set P . We then recursively compute the number of triangulations of each of the parts A and B , specified by the Separator Theorem, that are delimited by C^I . We then multiply the number we obtain for the sub-problem $A \cup C$ by the number we obtain for sub-problem $B \cup C$, and we add these products over all cycle separators C . With this algorithm we clearly over-count the triangulations of P , and it remains to show that we do not over-count by too much. We will later see that in order to keep over-counting small, we have to solve small recursive sub-problems exactly. Note that problems of size smaller than a threshold Δ can be solved exactly in time $O^*(2^\Delta)$ [2].

However, there are some technicalities that we have to overcome first. For starters, the Separator Theorem holds only if the unbounded face of T is also a triangle. Thus, if we add a dummy vertex v_∞ outside $\text{Conv}(P)$, along with the adjacencies between v_∞ and the vertices of $\text{Conv}(P)$, to make the unbounded face a triangle, we can apply the Separator Theorem. Once a simple cycle with the dividing properties of a separator is found, by the deletion of v_∞ we are left with a separator that is either the original cycle that we found, if v_∞ does not appear as a vertex of the separator, or a path otherwise. Thus, when guessing a separator we have to consider that it might be a path instead of a cycle. This brings us to the second technical issue. As we go deeper in the recursion we might create “holes” in P whose boundaries are the separators that we have considered thus far. That is, the recursive problems are polygonal regions, possibly with holes, containing points of P . Therefore, when guessing a separator, cycle or path, we have to arbitrarily triangulate the holes first. This does not modify the size of the sets we guess for a separator in a sub-problem.

We can now prove the first lemma:

Lemma 3 *Let $\mathcal{F}_T(P)$ be the set of triangulations of a set P of n points. Then all separators, simple cycles or paths, among all the elements of $\mathcal{F}_T(P)$ can be enumerated in time $2^{O(\sqrt{n} \log(n))}$.*

Proof. We know by the Separator Theorem and the discussion beneath that *every* element of $\mathcal{F}_T(P)$, a tri-

¹Thus separator C also forms part of the two sub-problems.

angulation, contains at least one separator C , simple cycle or path. Moreover, the size of C is at most $\sqrt{4n}$. Thus, searching by brute-force will do the job. We can enumerate all the subsets of P of size at most $\sqrt{4n}$ along with their permutations. A permutation tells us how to connect the points of the guessed subset, after also guessing whether we have a path or cycle. We can then verify if the constructed simple cycle, or path, fulfills the dividing properties of a separator, as specified in the Separator Theorem.

It is not hard to check that the total number of guessed subsets and their permutations is $2^{O(\sqrt{n} \log(n))}$. Verifying whether a cycle, or a path, is indeed a separator can be done in polynomial time. Thus, the total time spent remains being $2^{O(\sqrt{n} \log(n))}$. \square

We can now proceed with the proof of Theorem 1.

4 Proof of Theorem 1

We will first prove that the approximation ratio is sub-exponential and then prove that the algorithm has sub-exponential running time.

4.1 Quality of approximation

By the proof of Lemma 3 we also obtain that the number of simple cycle separators cannot be larger than $2^{O(\sqrt{n} \log(n))}$. Since at *every* stage of the recursion of the counting algorithm *no* triangulation of P can contain more than the total number of simple cycle separators found at that stage, we can express the *over-counting factor* of the algorithm by the following recurrence:

$$\begin{aligned} S(P, \Delta) &\leq \sum_C S(A \cup C, \Delta) \cdot S(B \cup C, \Delta) \\ &\leq 2^{O(\sqrt{n} \log(n))} \cdot S(A \cup C^*, \Delta) \cdot S(B \cup C^*, \Delta), \end{aligned}$$

where the summation is over all separators C available at the level of recursion, $A \cup C$, $B \cup C$ are the sub-problems as explained before, C^* is the cycle that maximizes $S(A \cup C, \Delta) \cdot S(B \cup C, \Delta)$ over all C , and Δ is a stopping size. Specifically, whenever the current recursive sub-problem contains $\leq \Delta$ points we stop the recursion and compute the number of triangulations of the sub-problem exactly. Hence, we have $S(P', \Delta) = 1$ whenever $|P'| \leq \Delta$. We can now write

$$\begin{aligned} S'(P, \Delta) := \log(S(P, \Delta)) &\leq O(\sqrt{n} \log(n)) + \\ &S'(A \cup C^*, \Delta) + \\ &S'(B \cup C^*, \Delta). \end{aligned}$$

Our goal now is to prove the following lemma:

Lemma 4 *Let P be a set of n points on the plane and assume $\Delta = n^{\Omega(1)}$, $n > \Delta$, and Δ is at least a sufficiently large constant. Then we have*

$$S'(P, \Delta) = O\left(\left(\frac{n}{\sqrt{\Delta/3}} - \sqrt{n}\right) \log \Delta\right).$$

Proof. We use induction over the size of P . Let $P' \subseteq P$ of size $m \leq n$. We have,

$$\begin{aligned} S'(P', \Delta) &\leq O(\sqrt{m} \log(m)) + S'(A \cup C^*, \Delta) + \\ &S'(B \cup C^*, \Delta) \\ &\leq O(\sqrt{m} \log(m)) + \\ &c \left(\frac{m_1}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_2} \right) \log \Delta, \quad (1) \end{aligned}$$

where m_1, m_2 are the sizes of the sub-problems $A \cup C^*$ and $B \cup C^*$ of P' , respectively, and c is some sufficiently large positive constant. By the Separator Theorem, we can express $m_1 \leq \alpha m + \sqrt{4m}$ and $m_2 \leq \beta m + \sqrt{4m}$, such that: (1) α, β are constants that depend on the instance, so $\alpha = \alpha(A \cup C^*)$ and $\beta = \beta(B \cup C^*)$, (2) $0 < \beta \leq \alpha \leq \frac{2}{3}$, and (3) $\alpha + \beta = 1$.

Now let us for the moment focus on the term $\frac{m_1}{\sqrt{\Delta/3}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\Delta/3}} - \sqrt{m_2}$ of equation (1) above:

$$\begin{aligned} &\frac{m_1}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_2} \\ &= \frac{m_1 + m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} - \sqrt{m_2} \\ &\leq \frac{\alpha m + \sqrt{4m} + \beta m + \sqrt{4m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} - \sqrt{m_2} \\ &\leq \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{\alpha m} - \sqrt{\beta m} \\ &= \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m} (\sqrt{\alpha} + \sqrt{\beta}) \\ &\leq \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m} (1 + \varepsilon) \end{aligned}$$

The last inequality is obtained by minimizing $\sqrt{\alpha} + \sqrt{\beta}$. Since we mentioned before that $0 \leq \beta \leq \alpha \leq \frac{2}{3}$ and $\alpha + \beta = 1$, the minimum of $\sqrt{\alpha} + \sqrt{\beta}$ is attained at $(\alpha, \beta) = (\frac{2}{3}, \frac{1}{3})$, and is strictly larger than one, so we can choose $\varepsilon > 0$. Now, since Δ is sufficiently large, we have $\frac{4\sqrt{m}}{\sqrt{\Delta/3}} \ll \varepsilon \sqrt{m}$, so $\frac{4\sqrt{m}}{\sqrt{\Delta/3}} - \varepsilon \sqrt{m} \leq -\varepsilon' \sqrt{m}$, for some positive constant ε' . Thus we can continue as follows:

$$\begin{aligned} \frac{m_1}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_1} + \frac{m_2}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m_2} &\leq \frac{m + 4\sqrt{m}}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m} (1 + \varepsilon) \\ &\leq \frac{m}{\sqrt{\frac{\Delta}{3}}} - (1 + \varepsilon') \sqrt{m}. \quad (2) \end{aligned}$$

Combining equations (1) and (2) we obtain

$$\begin{aligned} S'(P', \Delta) &\leq O(\sqrt{m} \log(m)) + c \left(\frac{m}{\sqrt{\frac{\Delta}{3}}} - (1 + \varepsilon')\sqrt{m} \right) \log \Delta \\ &\leq c \left(\frac{m}{\sqrt{\frac{\Delta}{3}}} - \sqrt{m} \right) \log \Delta \\ &\quad + O(\sqrt{m} \log(m)) - c \cdot \varepsilon' \sqrt{m} \log \Delta \end{aligned}$$

If we choose Δ to be sufficiently large, say $\Delta \geq n^\delta$, for some constant $\delta > 0$, then we have $\Delta \geq n^\delta \geq m^\delta$, and the negative term $-c \cdot \varepsilon' \sqrt{m} \log \Delta$ is larger, for appropriately large c , than the $O(\sqrt{m} \log(m))$ term. Hence, we can conclude that $S'(P', \Delta) \leq O\left(\left(\frac{m}{\sqrt{\Delta/3}} - \sqrt{m}\right) \log \Delta\right)$, which proves the induction step.

It still remains to prove that the inductive claim holds for the boundary condition, so let Q be a recursive sub-problem of size $\leq \Delta$. As Q stems from an application of the Separator Theorem, it is easy to see that $|Q| \geq \frac{\Delta}{3}$. Thus, we have $S'(Q, \Delta) = 0 \leq c \left(\frac{|Q|}{\sqrt{\Delta/3}} - \sqrt{|Q|} \right) \log \Delta$. Lemma 4 follows. \square

Now, let Λ be the number computed by our algorithm. Recall that $|\mathcal{F}_T(P)|$ is the exact number of triangulations of P . By setting $\Delta = \sqrt{n} \log(n)$ we obtain an over-counting factor of the algorithm of:

$$S(P, \Delta) = 2^{S'(P, \Delta)} = 2^{O\left(\frac{n \log \Delta}{\sqrt{\Delta}}\right)} = 2^{O\left(n^{\frac{3}{4}} \sqrt{\log(n)}\right)}$$

Hence $|\mathcal{F}_T(P)| \leq \Lambda \leq |\mathcal{F}_T(P)| \cdot 2^{O\left(n^{\frac{3}{4}} \sqrt{\log(n)}\right)} = |\mathcal{F}_T(P)|^{1+o(1)}$. This completes the qualitative part of Theorem 1. It remains to discuss the running time of our algorithm.

4.2 Running time

The running time of the algorithm can be expressed with the following recurrence:

$$T(n) < 2^{O(\sqrt{n} \log(n))} \cdot T\left(\frac{2n}{3} + \sqrt{4n}\right).$$

Taking again $T'(n) = \log(T(n))$ yields $T'(n) := T'\left(\frac{2n}{3} + \sqrt{4n}\right) + O(\sqrt{n} \log(n))$, which can then be solved using the well-known Akra-Bazzi Theorem for recurrences, see [6]. This yields $T'(n) = O(\sqrt{n} \log(n))$. There is, however, one detail missing, the stopping condition Δ . In order to use the Akra-Bazzi Theorem we need a boundary condition of $T(n) = 1$ for $1 \leq n \leq n_0$ (for some constant n_0), but in the algorithm we stop the recursion whenever a sub-problem Q is of size $\leq \Delta$ (which is dependent on the size n of the original point set). At that point we compute

the exact number of triangulations of Q , which gives $T(|Q|) = O^*(2^{|Q|}) = O^*(2^\Delta)$. Hence the exponent in the running time of the algorithm is upper-bounded by the solution of $T'(n)$, as given by the Akra-Bazzi Theorem, plus $O(\Delta)$, i.e., $T(n) = 2^{O(\sqrt{n} \log(n) + \Delta)}$. If as before we assume that $\Delta = \sqrt{n} \log(n)$ then we end up with $T(n) = 2^{O(\sqrt{n} \log(n))} = 2^{o(n)}$, which concludes the proof of Theorem 1.

As a final remark observe that we could have used other values for Δ , rather than $\sqrt{n} \log(n)$, without violating any argument in the proofs. This yields a trade-off with running time $2^{O(\Delta)}$ and approximation ratio $2^{O(\frac{n \log \Delta}{\sqrt{\Delta}})}$ for any $\sqrt{n} \log(n) \leq \Delta \leq n$. Although the quality of the approximation improves with larger Δ , the running time increases. Since we see no way of not having over-counting with this algorithm, we regard $\Delta = \sqrt{n} \log(n)$ as the most reasonable setting. It remains an open problem to find an algorithm with sub-exponential approximation ratio and running time $2^{o(\sqrt{n} \log(n))}$, e.g., polynomial.

5 Extensions and Conclusions

With the techniques of [3] one can generalize our algorithm for approximately counting triangulations to approximately counting other crossing-free structures. See [9] for other related results. In the following we sketch this for spanning cycles. We embed a spanning cycle in its constrained Delaunay triangulation, annotating the edges by whether they belong to the spanning cycle. To approximate the number of spanning cycles of a given set of points, we enumerate all possible cycle separators *together with all incident triangles* (similar to *sn-paths* versus triangular paths in [3]). Moreover, we enumerate all annotations of this structure by which edges belong to the spanning cycle and the topology of the seen parts of the spanning cycle. This gives an algorithm with the same asymptotic running time and approximation ratio as for triangulations. The full details of this, along with the details of how to approximately count crossing-free matchings and spanning trees, will appear in the full version of this paper.

In summary, in this paper we presented an approximation algorithm for the number of triangulations of a given point set. This algorithm has sub-exponential running time and sub-exponential approximation ratio. Although its approximation ratio is rather large, the algorithm computes a $(1+o(1))$ -approximation of the base c of the number of triangulations c^n , and it does so in sub-exponential time. No algorithm with this property was known before.

6 Acknowledgement

We would like to thank the anonymous referees for their valuable suggestions.

References

- [1] O. Aichholzer, “The path of a triangulation,” in *15th ACM Symposium on Computational Geometry*, pp. 14–23, ACM, 1999.
- [2] V. Alvarez and R. Seidel, “A Simple Aggregative Algorithm for Counting Triangulations of Planar Point Sets and Related Problems,” in *29th ACM Symposium on Computational Geometry* To appear, ACM, 2013.
- [3] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray, “Counting crossing-free structures,” in *28th ACM Symposium on Computational Geometry*, pp. 61–68, ACM, 2012.
- [4] V. Alvarez, K. Bringmann, and S. Ray, “A simple sweep line algorithm for counting triangulations and pseudo-triangulations”, *Submitted*, 2012.
- [5] H. Djidjev and S. M. Venkatesan, “Reduced constants for simple cycle graph separation,” *Acta Inf.*, vol. 34, no. 3, pp. 231–243, 1997.
- [6] T. Leighton, “Notes on better master theorems for divide-and-conquer recurrences,” tech. rep., Massachusetts Institute of Technology, 1996.
- [7] G. L. Miller, “Finding small simple cycle separators for 2-connected planar graphs,” *JCSS*, vol. 32, pp. 265–279, June 1986.
- [8] S. Ray and R. Seidel, “A simple and less slow method for counting triangulations and for related problems,” in *EuroCG*, 2004.
- [9] A. Razen and E. Welzl, “Counting Plane Graphs with Exponential Speed-Up” in *Rainbow of Computer Science*, pp. 36–46, 2011.
- [10] M. Sharir and A. Sheffer, “Counting triangulations of planar point sets,” *Electr. J. Comb.*, vol. 18, no. 1, 2011.
- [11] M. Sharir, A. Sheffer, and E. Welzl, “On degrees in random triangulations of point sets,” *J. Comb. Theory, Ser. A*, vol. 118, no. 7, pp. 1979–1999, 2011.

Aggregate-MAX Nearest Neighbor Searching in the Plane

Haitao Wang*

Abstract

We study the aggregate nearest neighbor searching for the MAX operator in the plane. For a set P of n points and a query set Q of m points, the query asks for a point of P whose maximum distance to the points in Q is minimized. We present data structures for answering such queries for both L_1 and L_2 distance measures. Previously, only heuristic and approximation algorithms were given for both versions. For the L_1 version, we build a data structure of $O(n)$ size in $O(n \log n)$ time, such that each query can be answered in $O(m + \log n)$ time. For the L_2 version, we build a data structure of $O(n \log \log n)$ size in $O(n \log n)$ time, such that each query can be answered in $O(m\sqrt{n} \log^{O(1)} n)$ time, and alternatively, we build a data structure in $O(n^{2+\epsilon})$ time and space for any $\epsilon > 0$, such that each query can be answered in $O(m \log n)$ time.

1 Introduction

Aggregate nearest neighbor (ANN) searching [1, 7, 8, 9, 10, 11, 15, 16, 17, 18, 19], also called group nearest neighbor searching, is a generalization of the fundamental nearest neighbor searching problem [2], where the input of each query is a set of points and the result of the query is based on applying some *aggregate* operator (e.g., MAX and SUM) on all query points. In this paper, we consider the ANN searching on the MAX operator for both L_1 and L_2 metrics in the plane.

For any two points p and q , let $d(p, q)$ denote the distance between p and q . Let P be a set of n points in the plane. Given any query set Q of m points, the ANN query asks for a point p in P such that $g(p, Q)$ is minimized, where $g(p, Q)$ is the *aggregate function* of the distances from p to the points of Q . The aggregate functions commonly considered are MAX, i.e., $g(p, Q) = \max_{q \in Q} d(p, q)$, and SUM, i.e., $g(p, Q) = \sum_{q \in Q} d(p, q)$. If the operator for g is MAX (resp., SUM), we use ANN-MAX (resp., ANN-SUM) to denote the problem.

In this paper, we focus on ANN-MAX in the plane for both L_1 and L_2 versions where the distance $d(p, q)$ is measured by L_1 and L_2 metrics, respectively.

Previously, only heuristic and approximation algorithms were given for both versions. For the L_1 version,

we build a data structure of $O(n)$ size in $O(n \log n)$ time, such that each query can be answered in $O(m + \log n)$ time. For the L_2 version, we build a data structure of $O(n \log \log n)$ size in $O(n \log n)$ time, such that each query can be answered in $O(m\sqrt{n} \log^{O(1)} n)$ time, and alternatively, we build a data structure in $O(n^{2+\epsilon})$ time and space for any $\epsilon > 0$, such that each query can be answered in $O(m \log n)$ time.

1.1 Previous Work

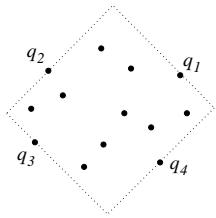
For ANN-MAX, Papadias et al. [16] presented a heuristic Minimum Bounding Method with worst case query time $O(n + m)$ for the L_2 version. Recently, Li et al. [7] gave more results on the L_2 ANN-MAX (the queries were called *group enclosing queries*). By using R -tree [6], Li et al. [7] gave an exact algorithm to answer ANN-MAX queries, and the algorithm is very fast in practice but theoretically the worst case query time is still $O(n+m)$. Li et al. [7] also gave a $\sqrt{2}$ -approximation algorithm with query time $O(m+\log n)$ and the algorithm works for any fixed dimensions, and they further extended the algorithm to obtain a $(1+\epsilon)$ -approximation result. To the best of our knowledge, we are not aware of any previous work that is particularly for the L_1 ANN-MAX. However, Li et al. [9] proposed the *flexible* ANN queries, which extend the classical ANN queries, and they provided an $(1+2\sqrt{2})$ -approximation algorithm that works for any metric space in any fixed dimension.

For L_2 ANN-SUM, a 3-approximation solution is given in [9]. Agarwal et al. [1] studied nearest neighbor searching under uncertainty, and their results can give an $(1+\epsilon)$ -approximation solution for the L_2 ANN-SUM queries. They [1] also gave an exact algorithm that can solve the L_1 ANN-SUM problem and an improvement based on their work has been made in [18].

There are also other heuristic algorithms on ANN queries, e.g., [8, 10, 11, 15, 17, 19].

Comparing with n , the value m is relative small in practice. Ideally we want a solution that has a query time $o(n)$. Our L_1 ANN-MAX solution is the first-known exact solution and is likely to be the best-possible. Comparing with the heuristic result [7, 16] with $O(m+n)$ worst case query time, our L_2 ANN-MAX solution use $o(n)$ query time for small m ; it should be noted that the methods in [7, 16] uses only $O(n)$ space while the space used in our approach is larger.

*Department of Computer Science, Utah State University, Logan, UT 84322, USA. E-mail: haitao.wang@usu.edu.

Figure 1: Illustrating the four extreme points q_1, q_2, q_3, q_4 .

2 The ANN-MAX in the L_1 Metric

In this section, we present our solution for the L_1 version of ANN-MAX queries. Given any query point set Q , our goal is to find the point $p \in P$ such that $g(p, Q) = \max_{q \in Q} d(p, q)$ is minimized for the L_1 distance $d(p, q)$, and we denote by $\psi(Q)$ the above sought point.

For each point p in the plane, denote by p_{\max} the farthest point of Q to p . We show below that p_{\max} must be an extreme point of Q along one of the four *diagonal* directions: northeast, northwest, southwest, southeast.

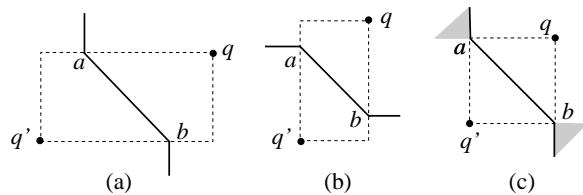
Let ρ_1 be a ray directed to the “northeast”, i.e., the angle between ρ and the x -axis is $\pi/4$. Let q_1 be an extreme point of Q along ρ_1 (e.g., see Fig. 1); if there is more than one such point, we let q_1 be an arbitrary such point. Similarly, let q_2, q_3 , and q_4 be the extreme points along the directions northwest, southwest, and southeast, respectively. Let $Q_{\max} = \{q_1, q_2, q_3, q_4\}$. Note that Q_{\max} may have less than four *distinct* points if two or more points of Q_{\max} refer to the same (physical) point of Q . Lemma 1, whose proof is omitted, shows that $g(p, Q)$ is determined only by the points of Q_{\max} .

Lemma 1 *For any point p in the plane, it holds that $g(p, Q) = \max_{q \in Q_{\max}} d(p, q)$.*

Note that a point may have more than one farthest point in Q . For any point p , if p has only one farthest point in Q , then p_{\max} is in Q_{\max} . Otherwise, p_{\max} may not be in Q_{\max} , and for convenience we re-define p_{\max} to be the farthest point of p in Q_{\max} . For each $1 \leq i \leq 4$, let $P_i = \{p \mid p_{\max} = q_i, p \in P\}$, i.e., P_i consists of the points of P whose farthest points in Q are q_i , and let p_i be the nearest point of q_i in P_i . To find $\psi(Q)$, we have the following lemma, whose proof is omitted.

Lemma 2 *$\psi(Q)$ is the point p_j for some j with $1 \leq j \leq 4$, such that $d(p_j, q_j) \leq d(p_i, q_i)$ holds for any $1 \leq i \leq 4$.*

Based on Lemma 2, to determine $\psi(Q)$, it is sufficient to determine p_i for each $1 \leq i \leq 4$. To this end, we make use of the farthest Voronoi diagram [5] of the four points in Q_{\max} , which is also the farthest Voronoi diagram of Q by Lemma 1. Denote by $FVD(Q)$ the farthest Voronoi diagram of Q_{\max} . Since Q_{\max} has only four points, $FVD(Q)$ can be computed in constant time,

Figure 2: Illustrating the bisector $B(q, q')$ (the solid curve) for q and q' . In (c), since $R(q, q')$ is a square, the two shaded quadrants are entirely in $B(q, q')$, but for simplicity, we only consider the two vertical bounding half-lines as in $B(q, q')$.

e.g., by an incremental approach. Each point $q \in Q_{\max}$ defines a cell $C(q)$ in $FVD(Q)$ such that every point $p \in C(q)$ is farthest to q_i among all points of Q_{\max} . In order to compute the four points p_i with $i = 1, 2, 3, 4$, we first show in the following that each cell $C(q)$ has certain special shapes that allow us to make use of the segment dragging queries [4, 14] to find the four points efficiently. Note that for each $1 \leq i \leq 4$, $P_i = P \cap C(q_i)$ and thus p_i is the nearest point of $P \cap C(q_i)$ to q_i . In fact, the following discussion also gives an incremental algorithm to compute $FVD(Q)$ in constant time.

2.1 The Bisectors

We first briefly discuss the bisectors of the points based on the L_1 metric. In fact, the L_1 bisectors have been well studied (e.g., [14]) and we discuss them here for completeness and some notation introduced here will also be useful later when we describe our algorithm.

For any two points q and q' in the plane, define $r(q, q')$ as the region of the plane that is the locus of the points farther to q than to q' , i.e., $r(q, q') = \{p \mid d(p, q) \geq d(p, q')\}$. The *bisector* of q and q' , denoted by $B(q, q')$, is the locus of the points that are equidistant to q and q' , i.e., $B(q, q') = \{p \mid d(p, q) = d(p, q')\}$. In order to discuss the shapes of the cells of $FVD(Q)$, we need to elaborate on the shape of $B(q, q')$, as follows.

Let $R(q, q')$ be the rectangle that has q and q' as its two vertices on diagonal positions (e.g., see Fig. 2). If the line segment $\overline{qq'}$ is axis-parallel, the rectangle $R(q, q')$ is degenerated into a line segment and $B(q, q')$ is the line through the midpoint of $\overline{qq'}$ and perpendicular to $\overline{qq'}$. Below, we focus on the general case where $\overline{qq'}$ is not axis-parallel. Without loss of generality, we assume q and q' are northeast and southwest vertices of $R(q, q')$, and other cases are similar.

The bisector $B(q, q')$ consists of two half-lines and one line segment in between (e.g., see Fig. 2); the two half-lines are either both horizontal or both vertical. Specifically, let l be the line of slope -1 that contains the midpoint of $\overline{qq'}$. Let $\overline{ab} = l \cap R(q, q')$, and a and b are on the boundary of $R(q, q')$. Note that if $R(q, q')$ is a square, then a and b are the other two vertices of $R(q, q')$ than q and q' ; otherwise, neither a nor b is a vertex.

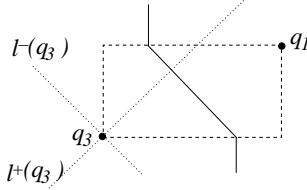


Figure 3: Illustrating an example where q_1 is above $l^-(q_3)$ and below or on $l^+(q_3)$. The bisector $B(q_1, q_3)$ is a v-bisector.

We first discuss the case where $R(q, q')$ is not a square (e.g., see Fig. 2 (a) and (b)). Let $l(a)$ be the line through a and perpendicular to the edge of $R(q, q')$ that contains a . The point a divides $l(a)$ into two half-lines, and we let $l'(a)$ be the one that does not intersect $R(q, q')$ except a . Similarly, we define the half-line $l'(b)$. Note that $l'(a)$ and $l'(b)$ must be parallel. The bisector $B(q, q')$ is the union of $l'(a)$, \overline{ab} , and $l'(b)$.

If $R(q, q')$ is a square, both a and b are vertices of $R(q, q')$ (e.g., see Fig. 2 (c)). In this case, a quadrant of a and a quadrant of b belong to $B(q, q')$, but for simplicity, we consider $B(q, q')$ as the union of \overline{ab} and the two vertical bounding half-lines of the two quadrants.

We call \overline{ab} the *middle segment* of $B(q, q')$ and denote it by $B_M(q, q')$. If $B(q, q')$ contains two vertical half-lines, we call $B(q, q')$ a *v-bisector* and refer to the two half-lines as *upper half-line* and *lower half-line*, respectively, based on their relative positions; similarly, if $B(q, q')$ contains two horizontal half-lines, we call $B(q, q')$ an *h-bisector* and refer to the two half-lines as *left half-line* and *right half-line*, respectively.

For any point p in the plane, we use $l^+(p)$ to denote the line through p with slope 1, $l^-(p)$ the line through p with slope -1 , $l_h(p)$ the horizontal line through p , and $l_v(p)$ the vertical line through p .

2.2 The Shapes of Cells of $FVD(Q)$

A subset Q' of Q is *extreme* if it contains an extreme point along each of the four diagonal directions. Q_{\max} is an extreme subset. A point q of Q_{\max} is *redundant* if $Q_{\max} \setminus \{q\}$ is still an extreme subset. For simplicity of discussion, we remove all redundant points from Q_{\max} . For example, if q_1 and q_2 are both extreme points along the northeast direction (and q_2 is also an extreme point along the northwest direction), then q_1 is redundant and we simply remove q_1 from Q_{\max} (and the new q_1 of Q_{\max} now refers to the same physical point as q_2).

Consider a point $q \in Q_{\max}$. Without loss of generality, we assume $q = q_3$ and the other cases can be analyzed similarly. We will analyze the possible shapes of $C(q_3)$. We assume Q_{\max} has at least two distinct points since otherwise the problem would be trivial. We further assume $q_1 \neq q_3$ since otherwise the analysis is much simpler. According to their definitions, q_1 must

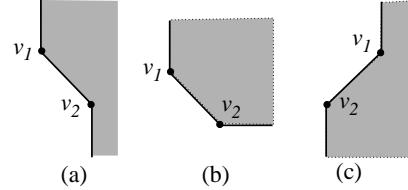


Figure 4: Illustrating three types of regions (shaded).

be above the line $l^-(q_3)$ (e.g., see Fig. 3). However, q_1 can be either above or below the line $l^+(q_3)$. In the following discussion, we assume q_1 is below or on the line $l^+(q_3)$ and the case where q_1 is above $l^+(q_3)$ can be analyzed similarly. In this case $B(q_3, q_1)$ is a v-bisector (i.e., it has two vertical half-lines).

We first introduce three *types* of regions (i.e., A , B , and C), and we will show later that $C(q_3)$ must belong to one of the types. Each type of region is bounded from the left or below by a polygonal curve ∂ consisting of two half-lines and a line segment of slope ± 1 in between.

1. From top to bottom, the polygonal curve ∂ consists of a vertical half-line followed by a line segment of slope -1 and then followed by a vertical half-line extended downwards (e.g., see Fig. 4 (a)). The region on the right of ∂ is defined as a *type-A* region.
2. From top to bottom, ∂ consists of a vertical half-line followed by a line segment of slope -1 and then followed by a horizontal half-line extended rightwards (e.g., see Fig. 4 (b)). The region on the right of and above ∂ is defined as a *type-B* region.
3. From top to bottom, ∂ consists of a vertical half-line followed by a line segment of slope 1 and then followed by a vertical half-line extended downwards (e.g., see Fig. 4 (c)). The region on the right of ∂ is defined as a *type-C* region.

In each type of the regions, the line segment of ∂ is called the *middle segment*. Denote by v_1 the upper endpoint of the middle segment and by v_2 the lower endpoint (e.g., see Fig. 4). Note that the middle segment may be degenerated to a point. By constructing $C(q_3)$ in an incremental manner, Lemma 3 shows that $C(q_3)$ must belong to one of the three types of regions. The proof of Lemma 3 is omitted.

Lemma 3 *The cell $C(q_3)$ must be one of the three types of regions. Further (see Fig. 5), if $C(q_3)$ is a type-A region, then $C(q_3)$ is to the right of $l_v(q_3)$ and v_2 is on $l_h(q_3)$; if $C(q_3)$ is a type-B region, then $C(q_3)$ is to the right of $l_v(q_3)$ and above $l_h(q_3)$; if $C(q_3)$ a type-C region, then $C(q_3)$ is to the right of $l_v(q_3)$ and v_1 is on $l_h(q_3)$.*

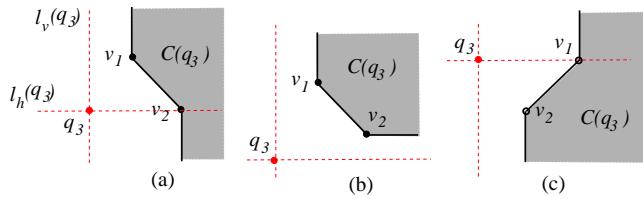


Figure 5: Illustrating the three possible cases for $C(q_3)$: (a) a type-A region; (b) a type-B region; (c) a type-C region.

2.3 Answering the Queries

Recall that our goal is to compute p_3 , which is the nearest point of $P \cap C(q_3)$ to q_3 . Based on Lemma 3, we can compute the point p_3 in $O(\log n)$ time by making use of the segment dragging queries [4, 14]. The details are given in Lemma 4.

Lemma 4 *After $O(n \log n)$ time and $O(n)$ space preprocessing on P , the point p_3 can be found in $O(\log n)$ time.*

Proof. We first briefly introduce the *segment dragging queries* that will be used by our algorithm: *parallel-track queries* and *out-of-corner queries* (e.g., Fig. 6).

Let S be a set of n points in the plane. For each parallel-track query, we are given two parallel vertical or horizontal lines (as “tracks”) and a line segment of slope ± 1 with endpoints on the two tracks, and the goal is to find the first point of S hit by the segment if we drag the segment along the two tracks. For each out-of-corner query, we are given two axis-parallel tracks forming a perpendicular corner, and the goal is to find the first point of S hit by dragging out of the corner a segment of slope ± 1 with endpoints on the two tracks.

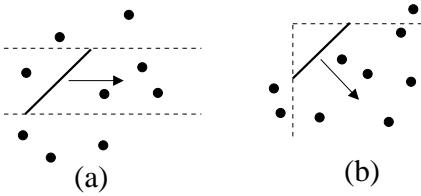


Figure 6: Illustrating the segment dragging queries: (a) a parallel-track query; (b) an out-of-corner query.

As shown by Mitchell [14], after $O(n \log n)$ time and $O(n)$ space preprocessing on S , each of the two types of queries can be answered in $O(\log n)$ [4, 14].

Below, we present our algorithm for the lemma by using the above segment dragging queries. Our goal is to find p_3 . Depending on the type of the $C(q_3)$ as stated in Lemma 3, there are three cases.

Type-A If $C(q_3)$ is a type-A region, we further decompose $C(q_3)$ into three subregions (e.g., see Fig. 7 (a)) by introducing two horizontal half-lines going rightwards

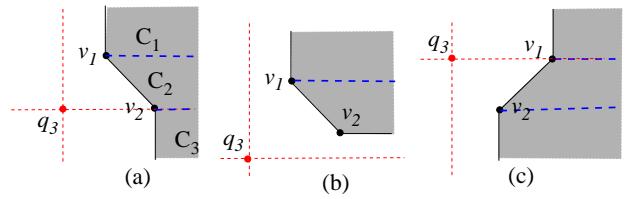


Figure 7: Illustrating the decomposition of $C(q_3)$ for segment-dragging queries.

from v_1 and v_2 (i.e., the endpoints of the middle segment of the boundary of $C(q_3)$), respectively. We call the three subregions the *upper*, *middle*, and *lower* subregions, respectively, according to their heights. To find p_3 , for each subregion C , we compute the closest point of $P \cap C$ to q_3 , and p_3 is the closest point to q_3 among the three points found above.

For the upper subregion, denoted by C_1 , according to Lemma 3, C_1 is in the first quadrant of q_3 . Therefore, q_3 ’s closest point in $P \cap C_1$ is exactly the answer of the out-of-corner query by dragging a segment of slope -1 from the corner of C_1 .

For the middle subregion, denoted by C_2 , according to Lemma 3, C_2 is in the first quadrant of q_3 . Therefore, q_3 ’s closest point in $P \cap C_2$ is exactly the answer of the parallel-track query by dragging the middle segment of the boundary of $C(q_3)$ rightwards.

For the lower subregion, denoted by C_3 , according to Lemma 3, C_3 is in the fourth quadrant of q_3 . Therefore, q_3 ’s closest point in $P \cap C_3$ is exactly the answer of the out-of-corner query by dragging a segment of slope 1 from the corner of C_3 .

Therefore, in this case we can find p_3 in $O(\log n)$ time after $O(n \log n)$ time $O(n)$ space preprocessing on P .

Type-B If $C(q_3)$ is a type-B region, we further decompose $C(q_3)$ into two subregions (e.g., see Fig. 7 (b)) by introducing a horizontal half-line rightwards from v_1 . To find p_3 , again, we find the closest point to q_3 in each of the two sub-regions.

By Lemma 3, both subregions are in the first quadrant of q_3 . By using the same approach as the first case, q_3 ’s closest point in the upper subregion can be found by an out-of-corner query and q_3 ’s closest point in the lower subregion can be found by a parallel-track query.

Type-C If $C(q_3)$ is a type-C region, the case is symmetric to the first case and we can find p_3 by using two out-of-corner queries and a parallel-track query.

As a summary, we can find p_3 in $O(\log n)$ time after $O(n \log n)$ time $O(n)$ space preprocessing on P . \square

By combining Lemmas 2 and 4, we conclude this section with the following theorem.

Theorem 5 Given a set P of n points in the plane, after $O(n \log n)$ time and $O(n)$ space preprocessing, we can answer each L_1 ANN-MAX query in $O(m + \log n)$ time for any set Q of m query points.

Proof. As preprocessing, we build data structures for answering the segment dragging queries on P in $O(n \log n)$ time and $O(n)$ space [4, 14].

Given any query set Q , we first determine Q_{\max} in $O(m)$ time. Then, we compute the farthest Voronoi diagram $FVD(Q)$ in constant time, e.g., by the incremental approach given in this paper. Then, for each $1 \leq i \leq 4$, we compute the point p_i by Lemma 4 in $O(\log n)$ time. Finally, $\psi(Q)$ can be determined by Lemma 2. \square

3 The ANN-MAX in the L_2 Metric

In this section, we present our results for the L_2 version of ANN-MAX queries. Given any query point set Q , our goal is to find the point $p \in P$ such that $g(p, Q) = \max_{q \in Q} d(p, q)$ is minimized for the L_2 distance $d(p, q)$, and we use $\psi(Q)$ to denote the sought point above.

We follow the similar algorithmic scheme as in the L_1 version. Let Q_H be the set of points of Q that are on the convex hull of Q . It is known that for any point p in the plane, its farthest point in Q is in Q_H , and in other words, the farthest Voronoi diagram of Q , denoted by $FVD(Q)$, is determined by the points of Q_H [5, 7]. Note that the size of $FVD(Q)$ is $O(|Q_H|)$ [5].

Consider any point $q \in Q_H$. Denote by $C(q)$ the cell of q in $FVD(Q)$, which is a convex and unbounded polygon [5]. Let $f(q)$ be the closest point of $P \cap C(q)$ to q . Similar to Lemma 2, we have the following lemma.

Lemma 6 If for a point $q' \in Q$, $d(f(q'), q') \leq d(f(q), q)$ holds for any $q \in Q_H$, then $f(q')$ is $\psi(Q)$.

Hence, to find $\psi(Q)$, it is sufficient to determine $f(q)$ for each $q \in Q$, as follows.

Consider any point $q \in Q$. To find $f(q)$, we first triangulate the cell $C(q)$ and let $Tri(q)$ denote the triangulation. For each triangle $\Delta \in Tri(q)$, we will find the closest point to q in $P \cap \Delta$, denoted by $f_\Delta(q)$. Consequently, $f(q)$ is the closest point to q among the points $f_\Delta(q)$ for all $\Delta \in Tri(q)$. Our goal is to determine $\psi(Q)$. To this end, we will need to triangulate each cell of $FVD(Q)$ and compute $f_\Delta(q)$ for each $\Delta \in Tri(q)$ and for each $q \in Q$. Since the size of $FVD(Q)$ is $O(|Q_H|)$, which is $O(m)$, we have the following lemma.

Lemma 7 If the closest point $f_\Delta(q)$ to q in $P \cap \Delta$ can be found in $O(t_\Delta)$ time for any triangle Δ and any point q in the plane, then $\psi(Q)$ can be found in $O(m \cdot t_\Delta)$ time.

In the following, we present our algorithms for computing $f_\Delta(q)$ for any triangle Δ and any point q in the plane. If we know the Voronoi diagram of the points

in $P \cap \Delta$, then $f_\Delta(q)$ can be determined in logarithmic time. Hence, the problem becomes how to maintain the Voronoi diagrams for the points in P such that given any triangle Δ , the Voronoi diagram information of the points in $P \cap \Delta$ can be obtained efficiently. To this end, we choose to augment the $O(n)$ -size simplex range (counting) query data structure in [12], as shown in the following lemma.

Lemma 8 After $O(n \log n)$ time and $O(n \log \log n)$ space preprocessing on P , we can compute the point $f_\Delta(q)$ in $O(\sqrt{n} \log^{O(1)} n)$ time for any triangle Δ and any point q in the plane.

Proof. We first briefly discuss the data structure in [12] and then augment it for our purpose. Note that the data structure in [12] is for any fixed dimension and our discussion below only focuses on the planar case, and thus each simplex below refers to a triangle.

A *simplicial partition* of the point set P is a collection $\Pi = \{(P_1, \Delta_1), \dots, (P_k, \Delta_k)\}$, where the P_i 's are pairwise disjoint subsets (called the *classes* of Π) forming a partition of P , and each Δ_i is a possibly unbounded simplex containing the points of P_i . The *size* of Π is k . The simplex Δ_i may also contain other points in P than those in P_i . A simplicial partition is called *special* if $\max_{1 \leq i \leq k} \{|P_i|\} < 2 \cdot \min_{1 \leq i \leq k} \{|P_i|\}$.

The data structure in [12] is a partition tree, denoted by T , based on constructing special simplicial partitions on P recursively. The leaves of T form a partition of P into constant-sized subsets. Each internal node $v \in T$ is associated with a subset P_v (and its corresponding simplex Δ_v) of P and a special simplicial partition Π_v of size $|P_v|^{1/2}$ of P_v . The root of T is associated with P . The *cardinality* of P_v (i.e., $|P_v|$) is stored at v . Each internal node v has $|P_v|^{1/2}$ children that correspond to the classes of Π_v . Thus, if v is a node lying at a distance i from the root of T , then $|P_v| = O(n^{1/2^i})$, and the depth of T is $O(\log \log n)$. It is shown in [12] that T has $O(n)$ space and can be constructed in $O(n \log n)$ time.

For each query simplex Δ , the goal is to compute the number of points in $P \cap \Delta$. We start from the root of T . For each internal node v , we check its simplicial partition Π_v one by one, and handle directly those contained in Δ or disjoint from Δ ; we proceed with the corresponding child nodes for the other simplices. Each of the latter ones must be intersected by at least one of the lines bounding Δ . If v is a leaf node, for each point p in P_v , we determine directly whether $p \in \Delta$. Each query takes $O(n^{1/2} (\log n)^{O(1)})$ time [12].

For our purpose, we augment the partition tree T . For each node v , we explicitly maintain the Voronoi diagram of P_v , denoted by $VD(P_v)$. Since at each level of T the subsets P_v 's are pairwise disjoint, comparing with the original tree, our augmented tree has $O(n)$ additional space at each level. Since T has $O(\log \log n)$ levels,

the total space of our augmented tree is $O(n \log \log n)$. For the running time, we claim that the total time for building the augmented tree is still $O(n \log n)$ although we have to build Voronoi diagrams for the nodes. Indeed, let $T(n)$ denote the time for building the Voronoi diagrams in the entire algorithm. We have $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + O(n \log n)$, and thus, $T(n) = O(n \log n)$ by solving the above recurrence.

Consider any query triangle Δ and any point q . We start from the root of T . For each internal node v , we check its simplicial partition Π_v , i.e., check the children of v one by one. Consider any child u of v . If Δ_u is disjoint from Δ , we ignore it. If Δ_u is contained in Δ , then we compute in $O(\log n)$ time the closest point of $P \cap \Delta_u$ to q (and its distance to q) by using the Voronoi diagram $VD(P_u)$ stored at the node u . Otherwise, we proceed on u recursively. If v is a leaf node, for each point p in P_v , we compute directly the distance $d(q, p)$ if $p \in \Delta$. Finally, $f_\Delta(q)$ is the closest point to q among all points whose distances to q have been computed above.

Comparing with the original simplex range query on Δ , we have $O(\log n)$ additional time on each node u if Δ_u is contained in Δ , and the number of such nodes is bounded by $O(n^{1/2}(\log n)^{O(1)})$. Hence, the total query time for finding $f_\Delta(q)$ is $O(n^{1/2}(\log n)^{O(1)} \cdot \log n)$, which is $O(n^{1/2}(\log n)^{O(1)})$. The lemma thus follows. \square

Similar augmentation may also be made on the $O(n)$ -size simplex data structure in [13] and the recent randomized result in [3]. If more space are allowed, by using duality and cutting trees [5], we can obtain the following lemma, whose proof is omitted.

Lemma 9 *After $O(n^{2+\epsilon})$ time and space preprocessing on P , we can compute the point $f_\Delta(q)$ in $O(\log n)$ time for any triangle Δ and any point q in the plane.*

Lemmas 7, 8, and 9 lead to the following theorem.

Theorem 10 *Given a set P of n points in the plane, after $O(n \log n)$ time and $O(n \log \log n)$ space preprocessing, we can answer each L_1 ANN-MAX query in $O(m\sqrt{n} \log^{O(1)} n)$ time for any set Q of m query points; alternatively, after $O(n^{2+\epsilon})$ time and space preprocessing for any $\epsilon > 0$, we can answer each L_2 ANN-MAX query in $O(m \log n)$ time.*

References

- [1] P.K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang. Nearest-neighbor searching under uncertainty. In *Proc. of the 31st Symposium on Principles of Database Systems*, pages 225–236, 2012.
- [2] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45:891–923, 1998.
- [3] T.M. Chan. Optimal partition trees. *Discrete and Computational Geometry*, 47:661–690, 2012.
- [4] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3(1–4):205–221, 1988.
- [5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry — Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008.
- [6] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
- [7] F. Li, B. Yao, and P. Kumar. Group enclosing queries. *IEEE Transactions on Knowledge and Data Engineering*, 23:1526–1540, 2011.
- [8] H. Li, H. Lu, B. Huang, and Z. Huang. Two ellipse-based pruning methods for group nearest neighbor queries. In *Proc. of the 13th Annual ACM International Workshop on Geographic Information Systems*, pages 192–199, 2005.
- [9] Y. Li, F. Li, K. Yi, B. Yao, and M. Wang. Flexible aggregate similarity search. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 1009–1020, 2011.
- [10] X. Lian and L. Chen. Probabilistic group nearest neighbor queries in uncertain databases. *IEEE Transactions on Knowledge and Data Engineering*, 20:809–824, 2008.
- [11] Y. Luo, H. Chen, K. Furuse, and N. Ohbo. Efficient methods in finding aggregate nearest neighbor by projection-based filtering. In *Proc. of the 12nd International Conference on Computational Science and its Applications*, pages 821–833, 2007.
- [12] J. Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8(3):315–334, 1992.
- [13] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10(1):157–182, 1993.
- [14] J.S.B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1):55–88, 1992.
- [15] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *Proc. of the 20th International Conference on Data Engineering*, pages 301–312, 2004.
- [16] D. Papadias, Y. Tao, K. Mouratidis, and C.K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems*, 30:529–576, 2005.
- [17] M. Sharifzadeh and C. Shahabi. VoR-Tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries. In *Proc. of the VLDB Endowment*, pages 1231–1242, 2010.
- [18] H. Wang and W. Zhang. The L_1 top- k nearest neighbor searching with uncertain queries. arXiv:1211.5084, 2013.
- [19] M.L. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering*, 17:820–833, 2005.

Data Structures for Incremental Extreme Ray Enumeration Algorithms

Blagoy Genov*

Abstract

Given a halfspace \mathcal{H} and a polyhedral cone \mathcal{P} with a known extreme ray set \mathcal{V} we consider the problem of finding the extreme ray set for the cone $\mathcal{P}' = \mathcal{H} \cap \mathcal{P}$. Regarding the computational time of the above problem, best results have been achieved with data structures based on multidimensional binary search trees. We refined the existing algorithm by developing a specific method for tree creation which brought further computational speedup. Furthermore, we examined alternative data structures based on vantage point trees and identified potential scenarios for their application.

1 Introduction

Polyhedral Cones. A nonempty set \mathcal{P} of vectors in \mathbb{R}^d is called a (*convex*) *polyhedral cone* if there exists a nonzero *representation matrix* $A \in \mathbb{R}^{n \times d}$ such that

$$\mathcal{P} := \{x \in \mathbb{R}^d : Ax \geq 0\}.$$

Those vectors $\mathcal{V}^E \subseteq \mathcal{P}$ which cannot be expressed as a conical combination of other vectors are called *extreme rays* of \mathcal{P} . The *active set* of extreme rays is defined by means of a mapping

$$\psi : \mathcal{V}^E \rightarrow \{0, 1\}^n$$

from extreme rays to binary vectors: $z = \psi(r)$ identifies the row vectors of A which r satisfied with equality, in the sense that

$$z_i = 0 \Rightarrow a_i^T \cdot r > 0 \text{ and } z_i = 1 \Rightarrow a_i^T \cdot r = 0$$

where a_i is the i -th row vector of A . Given the binary vectors $z = (z_1, \dots, z_n)$ and $z' = (z'_1, \dots, z'_n)$ the operations \wedge and \bar{z} (complement) as well as the relations \leq and $<$ are defined as (see [21]):

$$\begin{aligned} z \wedge z' &= (z_1 \wedge z'_1, \dots, z_n \wedge z'_n), \\ \bar{z} &= (\bar{z}_1, \dots, \bar{z}_n), \\ z \leq z' &\Leftrightarrow z_1 \leq z'_1, \dots, z_n \leq z'_n, \\ z < z' &\Leftrightarrow z \leq z' \text{ and } z \neq z'. \end{aligned}$$

The definition of \vee is analogous to that of \wedge . Additionally, we define the *population* of a binary vector

$$\rho : \{0, 1\}^n \rightarrow \mathbb{N}^0$$

as the count of its 1 values.

*Department of Computer Science, University of Bremen, Germany, bgenov@informatik.uni-bremen.de

Extreme Ray Enumeration. For a polyhedral cone \mathcal{P} the *extreme ray enumeration* is defined as the problem to find \mathcal{V}^E out of A . This problem, which is identical to the *vertex enumeration* of polytopes, has a number of algorithmic solutions developed over the years. The first one, called the *double description method*, was introduced by Motzkin et al. [20] and later improved by Fukuda et al. [16]. Further algorithms with historical significance are the *algorithm of Chernikova* [12, 19], the *beneath-and-beyond method* of Seidel [23, 15], the *randomized algorithm* of Clarkson and Shor [13], the *de-randomized algorithm* of Chazelle [11] and the *reverse search method* of Avis and Fukuda [3, 4]. For the moment, there is no general algorithm performing in time polynomial in the size of A and \mathcal{V}^E [1, 2]. The question of whether such an algorithm exists is open as well [18]. Yet, for nondegenerate problems polynomial time solutions are available [24, 3, 10].

In this paper, we focus on the practical implementation of incremental cutting plane algorithms like the double description method and Chernikova's algorithm. Those start with an approximation cone $\mathcal{P}_1 \supseteq \mathcal{P}$ for which the extreme ray set \mathcal{V}_1^E is known and perform a step by step refinement. At each step, the currently best known approximation \mathcal{P}_i is cut with a new halfspace $\mathcal{H}_i = \{a_i^T \cdot x \geq 0\}$ which splits \mathcal{V}_i^E into the subsets

$$\begin{aligned} \mathcal{V}_i^0 &= \{r^0 \in \mathcal{V}_i^E : a_i^T \cdot r^0 = 0\}, \\ \mathcal{V}_i^+ &= \{r^+ \in \mathcal{V}_i^E : a_i^T \cdot r^+ > 0\} \text{ and} \\ \mathcal{V}_i^- &= \{r^- \in \mathcal{V}_i^E : a_i^T \cdot r^- < 0\}. \end{aligned}$$

The set \mathcal{V}_{i+1}^E contains \mathcal{V}_i^0 , \mathcal{V}_i^+ and one new element for each pair of adjacent extreme rays $(r^+, r^-) \in (\mathcal{V}_i^+ \times \mathcal{V}_i^-)$. In practice, enumerating those pairs is the most time consuming part of the algorithm. We propose improvements related to the currently used data structures in order to speed up this process.

Assuming that \mathcal{P} is pointed and thus $\text{rank}[A] = d$, the adjacency test of two extreme rays could be performed in two different ways known as a *combinatorial* (see Lemma 1) and an *algebraic test* (see Lemma 2). For the proof of both lemmas we refer to [16, Proposition 7]. Corollary 3 expresses an incomplete form of the algebraic test delivering either a negative or an indecisive result.

Lemma 1 (Combinatorial Test) *Two extreme rays $r', r'' \in \mathcal{V}^E$ are adjacent if and only if there is no other*

extreme ray $r''' \in \mathcal{V}^E$ such that $\psi(r') \wedge \psi(r'') < \psi(r''')$.

Lemma 2 (Algebraic Test) Two extreme rays $r', r'' \in \mathcal{V}^E$ are adjacent if and only if $\text{rank}[A'] = d - 2$ where $A' \in \mathbb{R}^{k \times d}$ is a submatrix of A containing only those row vectors a_i for which $z_i = 1$ with $z = \psi(r') \wedge \psi(r'')$.

Corollary 3 Two extreme rays $r', r'' \in \mathcal{V}^E$ are nonadjacent if $\rho(z) < d - 2$ with $z = \psi(r') \wedge \psi(r'')$.

Using the above criteria we can outline a simple algorithm to identify all adjacent extreme ray pairs in $(\mathcal{V}_i^+ \times \mathcal{V}_i^-)$. First, we eliminate all pairs satisfying Corollary 3. We call this a *narrowing phase* and all remaining pairs *feasible* ones. Second, we check each feasible pair against Lemma 1 for a definite result. We call this a *verification phase*.

With regard to the narrowing phase, the enumeration of all feasible pairs has a quadratic complexity in the worst case, as each pair may indeed be a feasible one. If, however, the feasible pairs are only a small fraction, the enumeration could be sped up by applying the *divide-and-conquer* approach. First, the set \mathcal{V}_i^+ is partitioned into finitely many subsets, and then for each $r^- \in \mathcal{V}_i^-$ the search for feasible pairs is limited to those subsets which can produce a valid result. A closely related problem in metric spaces is the *fixed-radius near neighbor search* [7, 8]. Note that we are dealing here with a nonmetric space.

The divide-and-conquer approach is also applicable in the verification phase. Each application of Lemma 1 is basically a *partial match query* [22] on \mathcal{V}_i^E where the result is reduced to the existence or nonexistence of a ray r''' matching the given active set constraint. A general analysis on the lower bounds of this problem (referred to as a *no partial match query*) can be found in [9].

Contributions and Related Work. Thus far, the most efficient implementation of the outlined algorithm has been given by Terzer et al. [25, 26]. Terzer et al. introduced the *bit pattern tree* (here *bp-tree*), a data structure based on Bentley’s *k-d tree* [6], on which near neighbor and partial match queries are performed. The overall performance of the implementation, however, depends very much on the structure of the bp-trees. Differently structured trees may require completely different number of operations to process the same set of queries. We made use of the fact that in our case all query inputs are known before the tree creation and developed an optimization called *query bits neutralization*. This method for tree creation considered the query inputs during the creation process. The so generated *bp-qbn-trees* accelerated the overall computation for most of the investigated problems. In some cases, the calculation time was reduced by more than 80%. Furthermore,

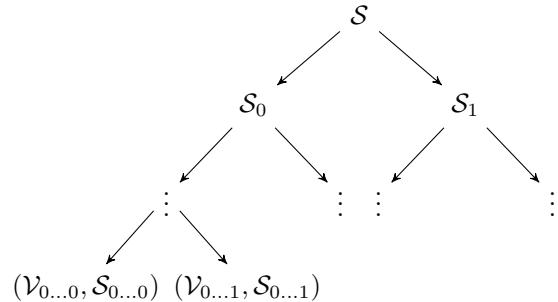


Figure 1: Generic structure of the binary tree.

we examined the performance of *vp-trees* (vantage point trees) [28], also known as *metric trees* [27], as alternative data structures for the algorithm. In this respect, we identified scenarios for which they tend to perform better than bp-trees.

2 Binary Trees for Adjacency Tests

In order to examine the performance of bp- und vp-trees we created a generic framework which supports adjacency tests using different binary tree types. In this section, we briefly introduce three major aspects of its functionality: tree generation, narrowing and verification. Functions whose implementation differs for different tree types are called *generic*. Specific implementations for bp- and vp-trees are presented in Sections 3 and 4.

Generation. The generation of a binary tree for some set of extreme rays \mathcal{V} involves two major steps. First, \mathcal{V} is recursively partitioned into finitely many subsets $\mathcal{V}_{(0|1)^*}$. Each subset corresponds to a single leaf node of the resulting tree. Second, to each tree node, no matter if intermediate or leaf, an auxiliary data $\mathcal{S}_{(0|1)^*}$ is attached. This data is produced during the generation process and encapsulates certain properties of the extreme rays in the subsequent leaf nodes. The basic structure of the resulting tree is shown in Figure 1. Its recursive creation is covered in Function 1 which returns either an intermediate node consisting of two subnodes or a leaf one if no further partitioning is desired. The generic function *partition* splits the set \mathcal{V} into two disjoint subsets according to some criteria.

Narrowing. In the narrowing phase, we construct a binary tree for the set \mathcal{V}_i^+ and perform an operation similar to a fixed-radius near neighbor search for each $r^- \in \mathcal{V}_i^-$. The implementation of the search procedure is given in Function 2. It recursively traverses the tree and enumerates all extreme rays r' building a feasible pair with the given ray r . At each recursion step, the

Function 1 `create(\mathcal{V}, \mathcal{S})`

```

if leaf_condition( $\mathcal{V}, \mathcal{S}$ ) then
    return ( $\mathcal{V}, \mathcal{S}$ )
else
     $((\mathcal{V}_0, \mathcal{S}_0), (\mathcal{V}_1, \mathcal{S}_1)) \leftarrow partition(\mathcal{V})$ 
     $\mathcal{T}_0 \leftarrow create(\mathcal{V}_0, \mathcal{S}_0)$ 
     $\mathcal{T}_1 \leftarrow create(\mathcal{V}_1, \mathcal{S}_1)$ 
    return ( $\mathcal{T}_0, \mathcal{T}_1, \mathcal{S}$ )
end if

```

Function 2 `cand(r, \mathcal{T})`

Require: $\mathcal{T} = (\mathcal{T}_0, \mathcal{T}_1, \mathcal{S})$ if \mathcal{T} is an intermediate node or $\mathcal{T} = (\mathcal{V}, \mathcal{S})$ if \mathcal{T} is a leaf one.

```

if proceed_enum( $r, \mathcal{S}$ ) then
    if leaf( $\mathcal{T}$ ) then
        return  $\{r' \in \mathcal{V} : \rho(\psi(r) \wedge \psi(r')) \geq d - 2\}$ 
    else
        return cand( $r, \mathcal{T}_0 \cup \mathcal{T}_1$ )
    end if
else
    return  $\emptyset$ 
end if

```

auxiliary data \mathcal{S} is used to check whether such rays can be found in the subsequent leaf nodes. If that is not the case, the current branch is abandoned. This decision is made by the generic function `proceed_enum`.

Verification. Here a binary tree is constructed for the set $\mathcal{V}_i^{deg} = \{r \in \mathcal{V}_i^E : \rho(\psi(r)) > d - 1\}$ and then traversed once for each feasible pair (r', r'') produced in the narrowing phase. If no partial match according to Lemma 1 is found then the corresponding rays are adjacent. The query implementation is given in Function 3 where `proceed_ver` is a generic function using the auxiliary data \mathcal{S} in order to check whether a certain branch can produce a match.

3 Bit Pattern Trees

In this section, we give the implementation of all generic functions for bp-trees and present the query bits neutralization method.

Implementation. The partitioning process for bp-trees is given in Function 4. It selects a vector $q \in \{0, 1\}^n$ with exactly one zero bit and groups all extreme rays r for which q is a valid over-approximation of $\psi(r)$ into \mathcal{V}_0 and all others into \mathcal{V}_1 . For each of the resulting subsets \mathcal{V}_k , $k \in \{0, 1\}$, an active set union $u_{\mathcal{V}_k}$ over all $\psi(r), r \in \mathcal{V}_k$, is generated and attached to the corresponding tree node as an auxiliary data. It represents an over-approximation of the active set for each extreme

Function 3 `ver(r', r'', \mathcal{T})`

Require: $\mathcal{T} = (\mathcal{T}_0, \mathcal{T}_1, \mathcal{S})$ if \mathcal{T} is an intermediate node or $\mathcal{T} = (\mathcal{V}, \mathcal{S})$ if \mathcal{T} is a leaf one.

```

 $e \leftarrow \psi(r') \wedge \psi(r'')$ 
if proceed_ver( $e, \mathcal{S}$ ) then
    if leaf( $\mathcal{T}$ ) then
        if  $\exists r''' \in \mathcal{V} \setminus \{r', r''\} : e < \psi(r''')$  then
            return false
        else
            return true
        end if
    else
        return ver( $r', r'', \mathcal{T}_0$ ) and ver( $r', r'', \mathcal{T}_1$ )
    end if
else
    return true
end if

```

Function 4 `partitionbpt(\mathcal{V})`

Let $q \in \{0, 1\}^n$ and $\rho(\bar{q}) = 1$

$$\mathcal{V}_0 \leftarrow \{r \in \mathcal{V} : \psi(r) \leq q\}; \mathcal{V}_1 \leftarrow \mathcal{V} \setminus \mathcal{V}_0$$

$$u_{\mathcal{V}_0} \leftarrow \bigvee_{r \in \mathcal{V}_0} \psi(r); u_{\mathcal{V}_1} \leftarrow \bigvee_{r \in \mathcal{V}_1} \psi(r)$$

```

return  $((\mathcal{V}_0, u_{\mathcal{V}_0}), (\mathcal{V}_1, u_{\mathcal{V}_1}))$ 

```

ray contained in one of the subsequent leaf nodes. Consequently, applying Corollary 3 or Lemma 1 on $u_{\mathcal{V}_k}$ can in some cases indicate the result for all extreme rays stored in the subsequent leaf nodes. The implementation of `proceed_enum` (see Function 5) and `proceed_ver` (see Function 6) for bp-trees rests on the above implication.

Function 5 `proceed_enumbpt(r, \mathcal{S})`

Require: $\mathcal{S} = u_{\mathcal{V}}$

```

if  $\rho(\psi(r) \wedge u_{\mathcal{V}}) \geq d - 2$  then
    return true
else
    return false
end if

```

Query Bits Neutralization. Using the vector q , at each partitioning step we can influence all active set unions in the left branch by defining a specific position at which their value is zero. Consequently, we can use this fact to stimulate the elimination of branches from the search procedures in both phases. In the narrowing phase, for example, we can intendedly plant zeros on positions which are likely to meet nonzero ones in the query input $\psi(r)$. The idea is to neutralize those positions in $\psi(r)$ which are likely to be 1 and thus reduce the value of $\rho(\psi(r) \wedge u_{\mathcal{V}})$ as much as possible. We call the so chosen zero positions *neutralizers*. The determination of the

Function 6 proceed_ver_{bpt}(e, S)

```

Require:  $S = u_{\mathcal{V}}$ 
  if  $e < u_{\mathcal{V}}$  then
    return true
  else
    return false
  end if
```

neutralizers can be done at the beginning of the narrowing phase by analyzing the set of all query inputs. Thus, if the bp-tree is built out of \mathcal{V}_i^+ then each vector q is extracted out of \mathcal{V}_i^- .

Neutralizers are also applicable in the verification phase as each zero position in $u_{\mathcal{V}}$ which is nonzero in e leads to termination of the search in the current branch. In this phase, however, analyzing the input data may cause a substantial overhead due to its generally large size. For those cases, the sets \mathcal{V}_i^+ and \mathcal{V}_i^- could be used to determine the neutralizers instead as $e \in (\mathcal{V}_i^+ \times \mathcal{V}_i^-)$.

There are two major metrics to evaluate the quality of the query bits neutralization. First, the probability of each neutralizer to meet a nonzero bit for some arbitrary query input. We call this a *hit probability*. Second, the number of neutralizers per active set union. It should be pointed out that those two factors may easily build a trade-off. A good neutralizer according to the first metric might also produce a bad partitioning where $|\mathcal{V}_0| << |\mathcal{V}_1|$ (see Function 4). In those cases, the impact of the neutralizer is considerably reduced as it will be planted only in a small fraction of the subsequent active set unions. Consequently, for an effective neutralizer selection both hit probability and potential partitioning should be taken into account.

4 Vantage Point Trees

In this section, we give the implementation of all generic functions for vp-trees.

In the *partition* function (see Function 7) we select an arbitrary extreme ray $v \in \mathcal{V}$, the so called *vantage point*, and measure the distance from v to all other rays in \mathcal{V} using the distance function

$$\delta(v, r) = \rho(\psi(r)) - \rho(\psi(v) \wedge \psi(r)).$$

Let δ_{max} be the maximal measured distance. The set \mathcal{V} is then split by selecting some arbitrary distance $l, 0 < l \leq \delta_{max}$, and grouping all rays r with $\delta(v, r) < l$ into \mathcal{V}_0 and all others into \mathcal{V}_1 . To each of the resulting subsets \mathcal{V}_k , $k \in \{0, 1\}$, we attach as an auxiliary data the active set union $u_{\mathcal{V}_k}$, the vantage point v , the distance range $q_{\mathcal{V}_k}$ and the population range $p_{\mathcal{V}_k}$. The distance range $q_{\mathcal{V}_k}$ is a closed interval bounded by the minimal and maximal distance from v to any ray in \mathcal{V}_k . The population range $p_{\mathcal{V}_k}$ is a closed interval bounded

Function 7 partition_{vpt}(V)

```

Let  $v \in \mathcal{V}$ 
 $\delta_{max} \leftarrow \delta(v, r)$  with  $r \in \mathcal{V} : \forall r' \in \mathcal{V}, \delta(v, r) \geq \delta(v, r')$ 
Let  $l \in (0, \delta_{max}]$ 
 $\mathcal{V}_0 \leftarrow \{r \in \mathcal{V} : \delta(v, r) < l\}; \mathcal{V}_1 \leftarrow \mathcal{V} \setminus \mathcal{V}_0$ 
 $u_{\mathcal{V}_0} \leftarrow \bigvee_{r \in \mathcal{V}_0} \psi(r); u_{\mathcal{V}_1} \leftarrow \bigvee_{r \in \mathcal{V}_1} \psi(r)$ 
 $q_{\mathcal{V}_0} \leftarrow [0, l - 1]; q_{\mathcal{V}_1} \leftarrow [l, \delta_{max}]$ 
 $p_{\mathcal{V}_0} \leftarrow [p_{min_0}, p_{max_0}]$  where
   $\forall r_0 \in \mathcal{V}_0, p_{min_0} \leq \rho(\psi(r_0)) \leq p_{max_0}$ 
 $p_{\mathcal{V}_1} \leftarrow [p_{min_1}, p_{max_1}]$  where
   $\forall r_1 \in \mathcal{V}_1, p_{min_1} \leq \rho(\psi(r_1)) \leq p_{max_1}$ 
 $\mathcal{S}_0 \leftarrow (u_{\mathcal{V}_0}, v, q_{\mathcal{V}_0}, p_{\mathcal{V}_0}); \mathcal{S}_1 \leftarrow (u_{\mathcal{V}_1}, v, q_{\mathcal{V}_1}, p_{\mathcal{V}_1})$ 
return  $((\mathcal{V}_0, \mathcal{S}_0), (\mathcal{V}_1, \mathcal{S}_1))$ 
```

by the minimal and maximal population of the elements in \mathcal{V}_k . With respect to the narrowing phase, the imple-

Function 8 proceed_enum_{vpt}(r, S)

```

Require:  $S = (u_{\mathcal{V}}, v, q_{\mathcal{V}}, p_{\mathcal{V}})$  with  $q_{\mathcal{V}} = [q_{min}, q_{max}]$ 
  and  $p_{\mathcal{V}} = [p_{min}, p_{max}]$ 
 $c_1 \leftarrow \rho(\psi(r)) + \delta(v, r) - q_{min}$ 
 $c_2 \leftarrow p_{max} + q_{max} - \delta(v, r)$ 
if  $c_1 \geq d - 2$  and  $c_2 \geq d - 2$  then
  return true
else
  return false
end if
```

mentation of *proceed_enum* (see Function 8) rests on the implication given in the following Lemma 4.

Lemma 4 *If the extreme rays $r', r'' \in \mathcal{V}^E$ are adjacent then for any $v \in \mathcal{V}^E$*

$$\begin{aligned} \rho(\psi(r')) + \delta(v, r') - \delta(v, r'') &\geq d - 2 \text{ and} \\ \rho(\psi(r'')) + \delta(v, r'') - \delta(v, r') &\geq d - 2. \end{aligned}$$

Let in the context of Lemma 4 r' be the argument r from Function 8. Then for r'' we use the intervals $q_{\mathcal{V}}$ and $p_{\mathcal{V}}$ as an over-approximation for the distance $\delta(v, r'')$ and the population $\rho(\psi(r''))$. As a consequence, the traversal of a certain tree branch needs to be proceeded only if the inequations given in Lemma 4 hold for any distance from $q_{\mathcal{V}}$ and any population from $p_{\mathcal{V}}$. Otherwise the nonexistence of feasible candidates in the subsequent leaf nodes is guaranteed.

For the implementation of *proceed_ver* (see Function 9) we apply the condition defined in Lemma 5. In a similar way, for the distance $\delta(v, r''')$ and the population $\rho(\psi(r'''))$ we use the ranges $q_{\mathcal{V}}$ and $p_{\mathcal{V}}$ from the auxiliary data. In order to maximize the branch elimination the conditional function for bp-trees is invoked as an additional criterion.

The proofs of Lemmas 4 and 5 can be found in the full version of the paper.

Lemma 5 If $r', r'', r''' \in \mathcal{V}^E$ are extreme rays such that $e < \psi(r''')$ for $e = \psi(r') \wedge \psi(r'')$ then for any $v \in \mathcal{V}^E$

$$\begin{aligned}\rho(\psi(v) \wedge e) &\leq \rho(\psi(v)) - \delta(v, r''') \text{ and} \\ \rho(\overline{\psi(v)} \wedge e) &\leq \rho(\psi(r''')) - \rho(\psi(v)) + \delta(v, r''').\end{aligned}$$

Function 9 `proceed_vervpt(e, S)`

Require: $S = (u_V, v, q_V, p_V)$ with $q_V = [q_{min}, q_{max}]$ and $p_V = [p_{min}, p_{max}]$

$c_1 \leftarrow \rho(\psi(v) \wedge e) - \rho(\psi(v)) + q_{min}$
 $c_2 \leftarrow \rho(\psi(v) \wedge e) + \rho(\psi(v)) - p_{max} - q_{max}$
if ($c_1 \leq 0$ **and** $c_2 < 0$) **or** ($c_1 < 0$ **and** $c_2 \leq 0$) **then**
 return `proceed_verbpt(e, u_V)`
else
 return `false`
end if

5 Results

In this section, we present the results from a small study comparing the performance of bp-trees, bt-qbn-trees and vp-trees. We used our own implementation of the double description method¹ in order to apply the vertex and facet enumeration on different polytopes. Those included samples from the work of Avis et al. [1, 2] and a small collection of cut [5], metric [14] and randomly generated 0/1 polytopes by polymake [17]. Table 1 summarizes the results for the cut polytope c_7 , the metric one m_7 , the product of cyclic polytopes $cyc_{4_26_2}$, the product of two simplices and a cube $glue_{54}$, the truncated polytope $trunc_{50}$ and three randomly generated 0/1 polytopes, one of which was joined with a hypercube.

	d	bp-trees	bp-qbn-trees	vp-trees
$cyc_{4_26_2}$	9	118.6	75.2	117.5
c_7	22	110.4	53.0	102.9
m_7	22	3463.8	1028.1	9393.9
rnd_{36}	22	1391.8	206.4	319.2
rnd_{cube}	31	756.3	715.2	137.9
$trunc_{50}$	51	470.0	582.7	421.0
$glue_{54}$	55	22.9	7.3	5.5
rnd_{64}	59	277.6	222.0	168.1

Table 1: Sum of narrowing and verification time (s)

On the basis of the experimental results, we outlined four major tendencies. First, vp-trees scaled best with growing dimensionality. This is visible, for instance, in the calculation times for rnd_{36} and rnd_{64} , which are similar problems in a different dimension. Second, bp-qbn-trees were not suitable for problems where only a

¹ Available at www.informatik.uni-bremen.de/agbs/bgenov

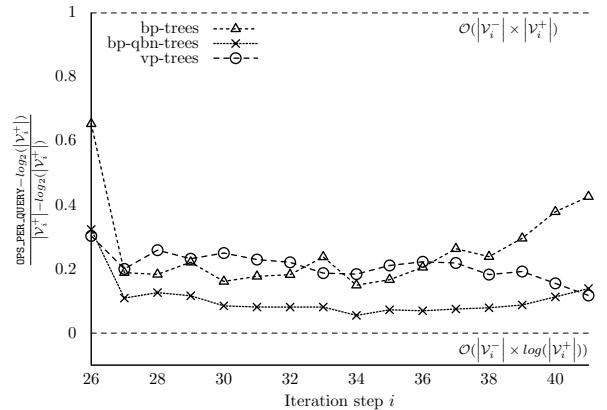


Figure 2: Narrowing phase for c_7 (snapshot).

small amount of data was processed at each step as the overhead for selecting neutralizers could not be compensated (see $trunc_{50}$). Third, the performance of vp-trees was very sensitive to the size of the population range p_V (see Function 8). The wider the range the worse the performance. Figure 2 illustrates the complexity of the nar-

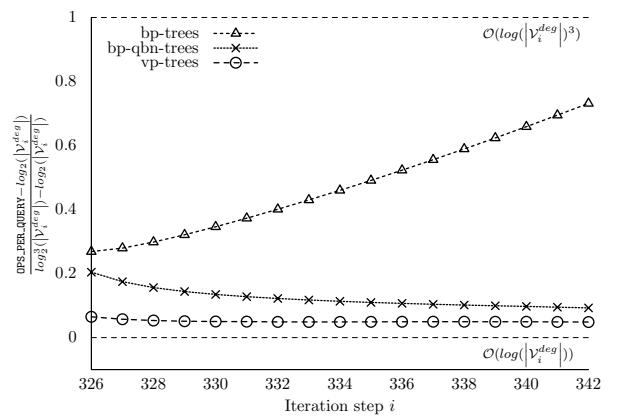


Figure 3: Verification phase for $glue_{54}$ (snapshot).

rowing phase for c_7 . For the majority of steps vp-trees could not reach the efficiency of the bp-qbn-trees due to the size of the population range. In the final steps p_V narrowed down which eventually boosted the vp-trees performance. In comparison, the verification phase of $glue_{54}$ (see Figure 3) illustrates a scenario with minimal population ranges. Finally, vp-trees showed considerably better results for rnd_{cube} which is a problem with an extremely high rate of negative tests in the verification phase. In the most time consuming steps, more than 99.99% of the tests were negative. In comparison, for m_7 this rate remained between 95 and 99%.

It is worth mentioning that for the generation of the complexity charts the invocation of `proceed_enumvpt|bpt` and `proceed_verbpt` counted as one operation. The execution of `proceed_vervpt` might have produced

up to two operations due to the potential call to `proceed_verbpt`.

6 Conclusion

In this paper, we revisited the application of bp-trees within incremental extreme ray enumeration algorithms and proposed a dynamic optimization of the trees with regard to the particular input problem. For most of the investigated problems a reduction in the overall calculation time was achieved. Furthermore, we examined the general suitability of vp-trees as an alternative data structure and presented problems for which vp-trees outperformed bp-trees. Still, further improvements are necessary so that vp-trees become competitive in the general case.

References

- [1] D. Avis and D. Bremner. How good are convex hull algorithms? In *Proceedings of the eleventh annual symposium on Computational geometry*, SCG '95, pages 20–28, New York, USA, 1995. ACM.
- [2] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Computational Geometry: Theory and Applications*, 7:265–301, 1997.
- [3] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. In *Proceedings of the seventh annual symposium on Computational geometry*, SCG '91, pages 98–104, New York, USA, 1991. ACM.
- [4] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(13):21–46, 1996.
- [5] F. Barahona and A. R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, 1986.
- [6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [7] J. L. Bentley. A survey of techniques for fixed radius near neighbor searching. Technical report, Stanford, CA, USA, 1975.
- [8] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.
- [9] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 312–321, New York, USA, 1999. ACM.
- [10] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete & Computational Geometry*, 20:333–357, 1998.
- [11] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.
- [12] N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5:228–233, 1965.
- [13] K. L. Clarkson and P. W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *Proceedings of the fourth annual symposium on Computational geometry*, SCG '88, pages 12–17, New York, 1988. ACM.
- [14] A. Deza, K. Fukuda, D. Pasechnik, and M. Sato. On the skeleton of the metric polytope. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 125–136. Springer, 2001.
- [15] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer-Verlag, New York, 1987.
- [16] K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and Computer Science*. Springer-Verlag, Berlin/Heidelberg, 1996.
- [17] E. Gawrilow and M. Joswig. polymake: a framework for analyzing convex polytopes. In *Polytopes — Combinatorics and Computation*. Birkhäuser, 2000.
- [18] L. Khachiyan, E. Boros, K. Borys, K. M. Elbassioni, and V. Gurvich. Generating all vertices of a polyhedron is hard. *Discrete & Computational Geometry*, 39(1-3):174–190, 2008.
- [19] H. Le Verge. A note on Chernikova's Algorithm. Technical Report 635, IRISA, Rennes, France, 1992.
- [20] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. *The double description method*, in *Contributions to the Theory of Games*, volume II, pages 51–73. Princeton University Press, 1953.
- [21] C. Posthoff and B. Steinbach. *Logic functions and equations: Binary models for computer science*. Springer, Dordrecht, The Netherlands, 2004.
- [22] R. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976.
- [23] R. Seidel. A convex hull algorithm optimal for point sets in even dimensions. Technical report, Vancouver, Canada, 1981.
- [24] R. Seidel. *Output-size sensitive algorithms for constructive problems in computational geometry*. PhD thesis, Ithaca, USA, 1987.
- [25] M. Terzer and J. Stelling. Accelerating the computation of elementary modes using pattern trees. In *Proceedings of the 6th international conference on Algorithms in Bioinformatics*, WABI '06, pages 333–343, Berlin/Heidelberg, 2006. Springer-Verlag.
- [26] M. Terzer and J. Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24:2229–2235, 2008.
- [27] J. K. Uhlmann. Metric trees. *Applied Mathematics Letters*, 4(5):61–62, 1991.
- [28] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. SIAM.

Fault Tolerant Clustering Revisited

Nirman Kumar*

Benjamin Raichel †

Abstract

In discrete k -center and k -median clustering, we are given a set of points P in a metric space M , and the task is to output a set $C \subseteq P$, $|C| = k$, such that the cost of clustering P using C is as small as possible. For k -center, the cost is the furthest a point has to travel to its nearest center, whereas for k -median, the cost is the sum of all point to nearest center distances. In the fault-tolerant versions of these problems, we are given an additional parameter $1 \leq \ell \leq k$, such that when computing the cost of clustering, points are assigned to their ℓ th nearest-neighbor in C , instead of their nearest neighbor. We provide constant factor approximation algorithms for these problems that are both conceptually simple and highly practical from an implementation stand-point.

1 Introduction

Two of the most common clustering problems are k -center and k -median clustering. In both these problems, the goal is to find the minimum cost partition of a given point set P into k clusters. Each cluster is defined by a point in the set of cluster *centers*, $C \subseteq P$, where $|C| = k$. In k -center clustering, the cost is the maximum distance of a point to its assigned cluster center, and in k -median clustering, the cost is the sum of distances of points to their assigned cluster center. In both cases, given a set of cluster centers C , a point is assigned to its closest center in C . Both these problems are NP-hard for most metric spaces. Hochbaum and Shmoys showed that k -center clustering has a 2-approximation algorithm, but for every $\varepsilon > 0$ it cannot be approximated to better than $(2 - \varepsilon)$ unless $P = NP$ [8]. A 2-approximation was also provided by Gonzalez [5], and by Feder and Greene [4]. For k -median, the best known approximation factor is $1 + \sqrt{3} + \varepsilon$. This is a recent result of Li and Svensson [13], but the approximation version of the k -median problem has a long history, and before the result of Li and Svensson, the best known result was by Arya *et al.* [2], that achieved an approximation factor of $(3 + \varepsilon)$ for any $\varepsilon > 0$, using local search. In general metric spaces, k -median is

also APX hard. Jain *et al.* showed that k -median is hard to approximate within a factor of $1 + 2/e \approx 1.736$ [9]. In Euclidean spaces, the k -center problem remains APX-hard [4], while k -median admits a PTAS [1, 11, 7].

Fault-Tolerance. As mentioned earlier, in both the k -center and k -median problems, each point is assigned to its closest center. Consider a realistic scenario where k -center clustering is used to decide in which k of n cities, certain facilities (say Sprawlmarks or hospitals) are opened, so that for clients in the n cities, their maximum distance to a facility is minimized. Once the k cities are decided upon, clearly each client goes to its nearest such facility when it requires service. Due to facility downtimes however, sometimes clients may need to go to their second closest, or third closest facility. Thus, in the fault-tolerant version of the k -center problem, we say that the cost of a client is the distance to its ℓ th nearest facility for some fixed $1 \leq \ell \leq k$. The problem now is to find a set of k centers so that the worst case cost is minimized, where in the worst case each client actually goes to its ℓ th nearest facility, and the cost of clustering is the maximum distance traveled by any client.

The fault-tolerant k -center problem was first studied by Krumke [12], who gave a 4-approximation algorithm for this problem. Chaudhuri *et al.* provided a 2-approximation algorithm for this problem [3], which is the best possible under standard complexity theoretic assumptions. In both these papers, the version considered, differs slightly from ours in that one only considers points which are not centers when computing the point that has the furthest distance to its ℓ th closest center. Khuller *et al.* [10] later considered both versions of the k -center problem. Their first version is the same as ours, i.e. the cost is the maximum distance of any point (including centers) to its ℓ th nearest center. They gave a 2-approximation when $\ell < 4$ and a 3-approximation otherwise. Their second version is the same as that of Krumke [12]. For this version, they provided a 2-approximation algorithm matching the result of Chaudhuri *et al.* [3].

For k -median clustering, a fault-tolerant version has been considered by Swamy and Shmoys [14]. However, our version is different from theirs.

*University of Illinois; nkumar5@illinois.edu; <http://www.cs.uiuc.edu/~nkumar5/>.

†University of Illinois; raichel2@illinois.edu; <http://www.cs.uiuc.edu/~raichel2/>.

Our Contribution. Our main contribution is in providing and proving the correctness of a natural technique for fault-tolerant clustering. In particular, letting $m = \lfloor k/\ell \rfloor$, we show that given a set of centers which is a constant factor approximation to the optimal m -center (resp. m -median) clustering, one can easily compute a set of k centers whose cost is a constant factor approximation to the optimal fault-tolerant k -center (resp. k -median) clustering. Specifically, in order to turn the non-fault-tolerant solution into a fault-tolerant one, simply add for each point of the m center set, its ℓ nearest neighbors in \mathcal{P} . In other words, our main contribution is in proving a relationship between the fault-tolerant and non-fault-tolerant cases, specifically that the non-fault-tolerant solution for m centers is already a near optimal fault-tolerant solution in that, up to a constant factor, it is enough to “reinforce” the current center locations rather than looking for new ones.

For fault-tolerant k -center we prove that if one applies this post-processing technique to any c -approximate solution to the non-fault-tolerant problem with m centers, then one is guaranteed a $(1 + 2c)$ -approximation to the optimal fault-tolerant clustering. Similarly, for fault-tolerant k -median we show this post processing technique leads to a $(1 + 4c)$ -approximation.

Our second main result is that using the algorithm of Gonzalez [5] for the initial m -center solution, gives a tighter approximation ratio guarantee. Specifically, we get a 3-approximation when $\ell|k$, and a 4-approximation otherwise, for fault-tolerant k -center. Additionally, on the median side, to the best of our knowledge, we are the first to consider this particular variant of fault-tolerant k -median clustering.

The approximation ratios of our algorithms are reasonable but not optimal. However, the authors feel that the algorithms more than make up for this in their conceptual simplicity and practicality from an implementation stand-point. Notably, if one has an existing implementation of an m -center or an m -median clustering approximation algorithm, one can immediately turn it into a fault-tolerant clustering algorithm for k centers with this technique.

Organization. In Section 2 we set up notation and formally define our variants for the fault-tolerant k -center and k -median problems. In Section 3 we review the algorithm of Gonzalez [5], and present our algorithms for the fault-tolerant k -center and k -median problems. In Section 4 we analyze the approximation ratios of our algorithm. We conclude in Section 5.

2 Preliminaries

We are given a set of n points $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ in a metric space M . Let $d(\mathbf{p}, \mathbf{p}')$ denote the distance be-

tween the points \mathbf{p} and \mathbf{p}' in M . For a point $\mathbf{p} \in M$, and a number $x \geq 0$, let $\text{ball}(\mathbf{p}, x)$ denote the closed ball of radius x with center \mathbf{p} . For a point $\mathbf{p} \in M$, a subset $S \subseteq \mathcal{P}$, and an integer $1 \leq i \leq |S|$, let $d_i(\mathbf{p}, S)$ denote the radius of the smallest (closed) ball with center \mathbf{p} that contains at least i points in the set S . Let $\text{nn}_i(\mathbf{p}, S)$ denote the i th nearest neighbor of \mathbf{p} in S , i.e. the point in S such that $d(\mathbf{p}, \text{nn}_i(\mathbf{p}, S)) = d_i(\mathbf{p}, S)$.¹ Let $\text{NN}_i(\mathbf{p}, S) = \cup_{j=1}^i \{\text{nn}_j(\mathbf{p}, S)\}$ be the set of i nearest neighbors of \mathbf{p} in S . By definition, for $1 \leq i \leq |S|$, $|\text{NN}_i(\mathbf{p}, S)| = i$. The following is an easy observation.

Observation 2.1 *For any fixed $Q \subseteq \mathcal{P}$ and integer $1 \leq i \leq |Q|$, the function $d_i(\cdot, Q)$ is a 1-Lipschitz function of its argument, i.e., for any $\mathbf{p}, \mathbf{q} \in M$, $d_i(\mathbf{p}, Q) \leq d_i(\mathbf{q}, Q) + d(\mathbf{p}, \mathbf{q})$.*

2.1 Problem Definitions

Problem 2.2 (Fault-tolerant k -center) *Let \mathcal{P} be a set of n points in M , and let k and ℓ be two given integer parameters such that $1 \leq \ell \leq k \leq n$. For a subset $C \subseteq \mathcal{P}$, we define the cost function $\mu(\mathcal{P}, C)$ as,*

$$\mu(\mathcal{P}, C) = \max_{\mathbf{p} \in \mathcal{P}} d_\ell(\mathbf{p}, C).$$

The fault-tolerant k -center problem, denoted $\text{FTC}(\mathcal{P}, k, \ell)$, is to compute a set C^ with $|C^*| = k$ such that,*

$$\mu(\mathcal{P}, C^*) = \min_{C \subseteq \mathcal{P}, |C|=k} \mu(\mathcal{P}, C).$$

For a given instance of $\text{FTC}(\mathcal{P}, k, \ell)$, we call C^* the optimum solution and we let r_{opt} denote its cost, i.e. $r_{\text{opt}} = \mu(\mathcal{P}, C^*)$. The classical **k -center** clustering problem on a point set \mathcal{P} is $\text{FTC}(\mathcal{P}, k, 1)$, and is referred to as the **non-fault-tolerant** k -center problem.

Problem 2.3 (Fault-tolerant k -median) *Let \mathcal{P} be set of n points in M , and let k and ℓ be two given integer parameters such that $1 \leq \ell \leq k \leq n$. For a subset $C \subseteq \mathcal{P}$, we define the cost function $\mu(\mathcal{P}, C)$ as,*

$$\mu(\mathcal{P}, C) = \sum_{\mathbf{p} \in \mathcal{P}} d_\ell(\mathbf{p}, C).$$

The fault-tolerant k -median problem, denoted $\text{FTM}(\mathcal{P}, k, \ell)$, is to compute a set C^ with $|C^*| = k$ such that,*

$$\mu(\mathcal{P}, C^*) = \min_{C \subseteq \mathcal{P}, |C|=k} \mu(\mathcal{P}, C).$$

For a given instance of $\text{FTM}(\mathcal{P}, k, \ell)$, we call C^* the optimum solution and we let σ_{opt} denote its cost, i.e. $\sigma_{\text{opt}} = \mu(\mathcal{P}, C^*)$. The classical **k -median** clustering problem on a point set \mathcal{P} is $\text{FTM}(\mathcal{P}, k, 1)$, and is referred to as the **non-fault-tolerant** k -median problem.

¹In case of non unique distances, we use the standard technique of lexicographic ordering of the pairs $(d(\mathbf{p}, \mathbf{p}_j), j)$ to ensure that the 1st, 2nd, ..., $|S|$ th, nearest-neighbors of \mathbf{p} are all unique.

3 Algorithms

Our algorithms for both problems, $\text{FTC}(\mathcal{P}, k, \ell)$ and $\text{FTM}(\mathcal{P}, k, \ell)$, have the same structure. In the first step they run an approximation algorithm for the non-fault-tolerant version of the respective problem, for $m = \lfloor k/\ell \rfloor$ centers, and in the second step, the solution output by the first step is added to in a straightforward manner described below. Notice that for either fault-tolerant problem, any approximation algorithm for the non-fault-tolerant version can be used in the first step. In particular, we prove that if the chosen algorithm for this first step is a c -approximation algorithm for the non-fault-tolerant problem for m centers, then the set we output at the end of step two will be a $(1 + 2c)$ -approximation (resp. $(1 + 4c)$ -approximation) for the fault-tolerant k -center (resp. k -median) problem with k centers.

Natural choices to use for our non-fault-tolerant m -median algorithm include the local search algorithm of Arya *et al.* [2], which is favored for its combinatorial nature, and simplicity of implementation, or the recent algorithm by Li and Svensson [13], which facilitates a slight improvement in the approximation factor. For the algorithms of Arya *et al.* and that of Li and Svensson we refer the reader to the respective papers, as knowledge of these algorithms is not required for understanding our algorithm. We let $\mathcal{A}_m(\mathcal{P}, m)$ denote the chosen approximation algorithm for m -median.

Similarly, we let $\mathcal{A}_c(\mathcal{P}, m)$ denote the chosen approximation algorithm for non-fault-tolerant m -center. Perhaps the most natural choice for our m -center algorithm is the 2-approximation algorithm by Gonzalez [5]. In fact, in Section 4.2.1 we show that this particular choice leads to a simpler analysis than the general case, and produces a much tighter approximation ratio guarantee. Since knowledge of the algorithm of Gonzalez is needed for this analysis, we briefly review this algorithm below in Section 3.2.

3.1 Fault-tolerant algorithms

We now describe the algorithms for fault-tolerant k -center and fault-tolerant k -median, that is $\text{FTC}(\mathcal{P}, k, \ell)$ and $\text{FTM}(\mathcal{P}, k, \ell)$.

For the problem $\text{FTC}(\mathcal{P}, k, \ell)$ (resp. $\text{FTM}(\mathcal{P}, k, \ell)$) first run the algorithm $\mathcal{A}_c(\mathcal{P}, m)$ (resp. $\mathcal{A}_m(\mathcal{P}, m)$). Let $Q \subseteq \mathcal{P}$ denote the set of m centers output, and let $Q = \{\mathbf{q}_1, \dots, \mathbf{q}_m\}$. Then the set of centers we output for our fault-tolerant solution is, $C = \bigcup_{i=1}^m \text{NN}_\ell(\mathbf{q}_i, \mathcal{P})$. That is, we take the ℓ nearest neighbors of each point \mathbf{q}_i in \mathcal{P} , for $i = 1, \dots, m$. We only use this set C in the analysis. If however C has less than k points, we can throw in $k - |C|$ additional points chosen arbitrarily from $\mathcal{P} \setminus C$, since adding additional centers can only decrease the cost of our solution.

Let $\mathcal{A}_{fc}(\mathcal{P}, k, \ell)$ and $\mathcal{A}_{fm}(\mathcal{P}, k, \ell)$ denote these algorithms for $\text{FTC}(\mathcal{P}, k, \ell)$ and $\text{FTM}(\mathcal{P}, k, \ell)$, respectively.

3.2 The algorithm of Gonzalez

We now describe the 2-approximation algorithm for the m -center problem, due to Gonzalez [5]. Gonzalez's algorithm builds a solution set C iteratively. To kick-start the iteration, we let $C = \{\mathbf{p}\}$ where $\mathbf{p} \in \mathcal{P}$ is an arbitrary point. Until m points have been accumulated, the algorithm repeatedly looks for the furthest point in \mathcal{P} to the current set C , and adds the found point to C . More formally, at each step we compute $\arg \max_{\mathbf{q} \in \mathcal{P}} d(\mathbf{q}, C)$, and add it to C .

This algorithm is not only simple from a conceptual stand-point, but also in regards to implementation and running time. Indeed, by just maintaining for each point in \mathcal{P} , its current nearest center among C , the above algorithm can be implemented in $O(n)$ time per iteration, for a total time of $O(nm)$. As mentioned earlier, the result of Hochbaum and Shmoys [8] implies that the approximation factor for this algorithm for general metric spaces, is the best possible.

4 Results and Analysis

We now present our results and their proofs. Our first result, is that using a factor c -approximation algorithm for $\mathcal{A}_m(\mathcal{P}, m)$ in the algorithm $\mathcal{A}_{fm}(\mathcal{P}, k, \ell)$ gives a $(1 + 4c)$ -approximation algorithm for the problem $\text{FTM}(\mathcal{P}, k, \ell)$. The structure of the k -center problem allows us to use a nearly identical analysis except with one simplification, yielding an improved $(1 + 2c)$ -approximation algorithm for the problem $\text{FTC}(\mathcal{P}, k, \ell)$. Our second result, shows that if one uses the algorithm of Gonzalez [5] for the subroutine $\mathcal{A}_c(\mathcal{P}, m)$, then one can guarantee a tighter approximation ratio of 4 (or 3 if $l|k$), as opposed to the 5 guaranteed by our first result.

4.1 Analysis for fault-tolerant k -median

Theorem 4.1 *For a given point set \mathcal{P} in a metric space M with $|\mathcal{P}| = n$, the algorithm $\mathcal{A}_{fm}(\mathcal{P}, k, \ell)$ achieves a $(1 + 4c)$ -approximation to the optimal solution of $\text{FTM}(\mathcal{P}, k, \ell)$, where c is the approximation guarantee of the subroutine $\mathcal{A}_m(\mathcal{P}, m)$, where $m = \lfloor k/\ell \rfloor$.*

As a corollary we have,

Corollary 4.2 *There is a 12-approximation algorithm for the problem $\text{FTM}(\mathcal{P}, k, \ell)$.*

Proof. We use the $(1 + \sqrt{3} + \varepsilon)$ -approximation algorithm of Li and Svensson [13] with a small enough ε , for the subroutine $\mathcal{A}_m(\mathcal{P}, m)$. The result follows by appealing to Theorem 4.1. \square

Proof of Theorem 4.1

We refer the reader to Section 2 for notation already introduced. We need some more notation. For a given instance of $\text{FTM}(\mathcal{P}, k, \ell)$, let $C^* = \{w_1, w_2, \dots, w_k\}$ be an optimal set of centers, and let σ_{opt} be its cost, i.e., $\sigma_{\text{opt}} = \sum_{p \in \mathcal{P}} d_\ell(p, C^*)$. Let $C = \{c_1, \dots, c_k\}$ be the set of centers returned by $\mathcal{A}_{\text{fm}}(\mathcal{P}, k, \ell)$, and σ_{alg} its cost.

Let $m = \lfloor k/\ell \rfloor$, and let σ_{med} denote the cost of the optimum m -median clustering on \mathcal{P} , i.e., the optimum for the problem $\text{FTM}(\mathcal{P}, m, 1)$. When $\mathcal{A}_{\text{fm}}(\mathcal{P}, k, \ell)$ is run, it makes a subroutine call to $\mathcal{A}_m(\mathcal{P}, m)$. Let $Q = \{q_1, \dots, q_m\}$ be the set of centers returned by this subroutine call. We know that Q is a c -approximation to the optimal solution to $\text{FTC}(\mathcal{P}, m, 1)$.

Notice that, C includes $\bigcup_{i=1}^m \text{NN}_\ell(q_i, \mathcal{P})$. We assume that the set C has exactly k points. As mentioned earlier, we only require that C includes $\bigcup_{i=1}^m \text{NN}_\ell(q_i, \mathcal{P})$ in our analysis, and if $|\bigcup_{i=1}^m \text{NN}_\ell(q_i, \mathcal{P})| < k$, we can always add additional points. This can only decrease the cost of clustering.

Proving the following two claims will immediately imply $\sigma_{\text{alg}} \leq (1 + 4c)\sigma_{\text{opt}}$.

Claim 4.3 *We have that, $\sigma_{\text{alg}} \leq \sigma_{\text{opt}} + 2c\sigma_{\text{med}}$.*

Claim 4.4 *We have that, $\sigma_{\text{med}} \leq 2\sigma_{\text{opt}}$.*

Proof of Claim 4.3: Let $p \in \mathcal{P}$, and let $q = \text{nn}_1(p, Q)$. By Observation 2.1, $d_\ell(p, C) \leq d(p, q) + d_\ell(q, C)$. As $\text{NN}_\ell(q, \mathcal{P}) \subseteq C$, we have that $d_\ell(q, \mathcal{P}) = d_\ell(q, C)$. Again by Observation 2.1, $d_\ell(q, \mathcal{P}) \leq d(q, p) + d_\ell(p, \mathcal{P})$. Combining the two inequalities gives, $d_\ell(p, C) \leq 2d(p, q) + d_\ell(p, \mathcal{P}) = 2d_1(p, Q) + d_\ell(p, \mathcal{P})$. Thus,

$$\begin{aligned} \sigma_{\text{alg}} &= \sum_{p \in \mathcal{P}} d_\ell(p, C) \leq \sum_{p \in \mathcal{P}} (2d_1(p, Q) + d_\ell(p, \mathcal{P})) \\ &\leq 2c\sigma_{\text{med}} + \sigma_{\text{opt}}, \end{aligned} \quad (1)$$

as Q is a c -approximate m -median solution, $d_\ell(p, \mathcal{P}) \leq d_\ell(p, C^*)$, and $\sigma_{\text{opt}} = \sum_{p \in \mathcal{P}} d_\ell(p, C^*)$. ■

The following is required to prove Claim 4.4, but is interesting in its own right.

Lemma 4.5 *Let M be any metric space. Let $X \subseteq M$ with $|X| = t$. Then for any integer $1 \leq h \leq t$, and any finite set $Y \subseteq M$, there exists a subset $S \subseteq Y$, such that (A) $|S| \leq t/h$, and, (B) $\forall y \in Y, d_1(y, S) \leq 2d_h(y, X)$.*

Proof. We give an algorithm to construct such a subset $S \subseteq Y$. This subset is constructed by iteratively scooping out the points of the minimum radius ball containing h points from X , adding the center to S , and repeating. Formally, let $W_0 = \emptyset$, and for $i = 1, \dots, \lfloor t/h \rfloor$, define iteratively, $X_i = X \setminus \left(\bigcup_{j=0}^{i-1} W_j \right)$, $y_i = \arg \min_{v \in Y} d_h(v, X_i)$, and, $W_i = \text{NN}_h(y_i, X_i)$. We prove that $S = \bigcup_{i=1}^{\lfloor t/h \rfloor} \{y_i\}$, is the desired subset of points.

First, clearly $|S| \leq t/h$. Let $y \in Y$, and let $b = \text{ball}(y, x)$, where $x = d_h(y, X)$. Let W_i be the first subset, i.e. the one with smallest index i , such that there exists some point $w \in b \cap W_i$. Such a point must exist, since fewer than h points are in $X \setminus \left(\bigcup_{j=1}^{\lfloor t/h \rfloor} W_j \right)$, while $|b \cap X| \geq h$. Clearly $b \cap X \subseteq X_i$, as i is the minimum index such that $b \cap W_i \neq \emptyset$. As such we have, $d_h(y, X) = d_h(y, X_i)$. Let $r_i = d_h(y_i, X_i)$, be the radius of the ball that scooped out W_i . Clearly $r_i \leq x$, as

$$x = d_h(y, X) = d_h(y, X_i) \geq r_i = \arg \min_{v \in Y} d_h(v, X_i).$$

Now, since $w \in b \cap W_i$, $d(y_i, w) \leq r_i = d_h(y_i, X_i)$. By the triangle inequality,

$$\begin{aligned} d_1(y, S) &\leq d(y, y_i) \leq d(y, w) + d(w, y_i) \leq x + r_i \leq 2x \\ &= 2d_h(y, X). \end{aligned}$$

□

Proof of Claim 4.4: We use Lemma 4.5 with $Y = \mathcal{P}$, $X = C^*$, $t = |C^*| = k$ and $h = \ell$. Let S be the subset of \mathcal{P} guaranteed by Lemma 4.5. Now $|S| \leq k/\ell$, and as such $|S| \leq m$. We have,

$$\sigma_{\text{med}} \leq \sum_{p \in \mathcal{P}} d_1(p, S) \leq \sum_{p \in \mathcal{P}} 2d_\ell(p, C^*) = 2\sigma_{\text{opt}}. \quad (2)$$

The first inequality follows since σ_{med} is the cost of the optimum m -median clustering of \mathcal{P} , while $\sum_{p \in \mathcal{P}} d_1(p, S)$ is the cost of a $|S|$ -median clustering of \mathcal{P} by the set of centers $S \subseteq \mathcal{P}$ with $|S| \leq m$. The second inequality follows from Lemma 4.5. ■

This concludes the proof of Theorem 4.1.

4.2 Analysis for fault-tolerant k -center

We now present the analogues result to Theorem 4.1 for fault-tolerant k -center. By following the proof nearly verbatim from the previous section one sees that similar to $\mathcal{A}_{\text{fm}}(\mathcal{P}, k, \ell)$, $\mathcal{A}_{\text{fc}}(\mathcal{P}, k, \ell)$ also provides a $(1 + 4c)$ -approximation. However, in this case we will actually get a $(1 + 2c)$ -approximation, since now an improved and simpler version of Claim 4.3 holds.

As a quick note on notation, here r_{alg} , r_{opt} , and r_{cen} will play the analogues role for center as σ_{alg} , σ_{opt} , and σ_{med} played for median.

Claim 4.6 *We have that, $r_{\text{alg}} \leq r_{\text{opt}} + 2cr_{\text{cen}}$.*

Proof of Claim 4.6: Let $p \in \mathcal{P}$, and let $q = \text{nn}_1(p, Q)$. By Observation 2.1, $d_\ell(p, C) \leq d(p, q) + d_\ell(q, C) = d(p, q) + d_\ell(q, \mathcal{P})$, where the equality follows since $\text{NN}_\ell(q, \mathcal{P}) \subseteq C$. Thus,

$$\begin{aligned} r_{\text{alg}} &= \max_{p \in \mathcal{P}} d_\ell(p, C) \leq \max_{p \in \mathcal{P}} (d_1(p, Q) + d_\ell(q, \mathcal{P})) \\ &\leq 2cr_{\text{cen}} + r_{\text{opt}}, \end{aligned} \quad (3)$$

as Q is a c -approximate m -center solution, $d_\ell(q, P) \leq d_\ell(q, C^*)$, and $r_{\text{opt}} = \max_{p \in P} d_\ell(p, C^*)$. ■

Theorem 4.7 *For a given point set P in a metric space M with $|P| = n$, the algorithm $\mathcal{A}_{\text{fc}}(P, k, \ell)$ achieves a $(1 + 2c)$ -approximation to the optimal solution of $\text{FTC}(P, k, \ell)$, where c is the approximation guarantee of the subroutine $\mathcal{A}_c(P, m)$, where $m = \lfloor k/\ell \rfloor$.*

Proof. As stated above, the proof of this theorem is very similar to the proof of Theorem 4.1. In fact, we can repeat the proof of Theorem 4.1 almost word for word, except that we need to replace the sum function \sum by the max function, $\max_{p \in P}$. More specifically, this needs to be done for Eq. (2) in the proof of Claim 4.4, and to replace Eq. (1) from Claim 4.3 we instead use the improved Eq. (3) from Claim 4.6. As the proof can be reconstructed step-by-step from the detailed proof of Theorem 4.1 by making these modifications, we omit it for the sake of brevity. □

4.2.1 A tighter analysis when using Gonzalez's algorithm as a subroutine

If we use a 2-approximation algorithm for the subroutine $\mathcal{A}_c(P, m)$, Theorem 4.7 implies that $\mathcal{A}_{\text{fc}}(P, k, \ell)$ is a 9-approximation algorithm. Here we present a tighter analysis for the case when we use the 2-approximation algorithm of Gonzalez [5] (see also Section 3.2) for the subroutine $\mathcal{A}_c(P, m)$.

See Section 2 for definitions and notation introduced previously. Some more notation is needed. Let $C^* = \{w_1, w_2, \dots, w_k\}$ be an optimal set of centers. Its cost, r_{opt} , is $\max_{p \in P} d_\ell(p, C^*)$. Let $C = \{c_1, \dots, c_k\}$ be the set of centers returned by $\mathcal{A}_{\text{fc}}(P, k, \ell)$, and let r_{alg} be its cost.

Let $m = \lfloor k/\ell \rfloor$, where for now we assume $\ell|k$, i.e., $m = k/\ell$. As we show later, this assumption can be removed. When $\mathcal{A}_{\text{fc}}(P, k, \ell)$ is run, it makes a subroutine call to $\mathcal{A}_c(P, m)$. As mentioned, in this section we require this subroutine to be the algorithm of Gonzalez [5]. Let $Q = \{q_1, \dots, q_m\}$ be the set of centers returned by this subroutine call. Additionally, let $r_i = d(q_i, Q_{i-1})$ for $2 \leq i \leq m$, where $Q_{i-1} = \{q_1, \dots, q_{i-1}\}$. We assume $m > 1$, as the $m = 1$ case is easier.

The following is easy to see, and is used in the correctness proof for the algorithm of Gonzalez. See [6] for an exposition.

Lemma 4.8 *For $i \neq j$, $d(q_i, q_j) \geq r_m$.*

Lemma 4.9 *For any q_i , $\text{NN}_\ell(q_i, C^*) \subseteq \text{ball}(q_i, r_{\text{opt}})$ and $\text{NN}_\ell(q_i, P) \subseteq \text{ball}(q_i, r_{\text{opt}})$.*

Proof. The first claim follows since $q_i \in P$ and so $d_\ell(q_i, C^*) \leq r_{\text{opt}}$. As $C^* \subseteq P$, the second claim follows. □

Lemma 4.10 *We have that, $r_{\text{alg}} \leq r_m + r_{\text{opt}}$.*

Proof. As in Gonzalez's algorithm, we have $r_m = \max_{p \in P} d(p, Q_{m-1})$, and so $d(p, Q) \leq r_m$ for any $p \in P$. Consider any point $p \in P$, and let $q = \text{nn}_1(p, Q)$. By how $\mathcal{A}_{\text{fc}}(P, k, \ell)$ is defined, $\text{NN}_\ell(q, P) \subseteq C$, and so $d_\ell(q, C) = d_\ell(q, P) \leq d_\ell(q, C^*) \leq r_{\text{opt}}$. By Observation 2.1 we have, $d_\ell(p, C) \leq d(p, q) + d_\ell(q, C) \leq r_m + r_{\text{opt}}$. □

Lemma 4.11 *If $r_{\text{alg}} > 3r_{\text{opt}}$, then for any $1 \leq i \neq j \leq m$, $\text{ball}(q_i, r_{\text{opt}})$ and $\text{ball}(q_j, r_{\text{opt}})$ are disjoint and each contains at least ℓ centers from C^* .*

Proof. Let q_i and q_j be any two distinct centers in Q . By Lemma 4.8 and Lemma 4.10, $d(q_i, q_j) \geq r_m \geq r_{\text{alg}} - r_{\text{opt}} > 2r_{\text{opt}}$, which implies that, $\text{ball}(q_i, r_{\text{opt}}) \cap \text{ball}(q_j, r_{\text{opt}}) = \emptyset$. Each ball contains ℓ centers from C^* by Lemma 4.9. □

Lemma 4.12 *We have that, $r_{\text{alg}} \leq 3r_{\text{opt}}$.*

Proof. Suppose otherwise that $r_{\text{alg}} > 3r_{\text{opt}}$. By Lemma 4.11, for $i = 1, \dots, m$, $|\text{ball}(q_i, r_{\text{opt}}) \cap C^*| \geq \ell$, and for $1 \leq i < j \leq m$, $\text{ball}(q_i, r_{\text{opt}}) \cap \text{ball}(q_j, r_{\text{opt}}) = \emptyset$. Assign all points in $C^* \cap \text{ball}(q_i, r_{\text{opt}})$ to q_i . Notice, q_i is the unique point from Q within distance r_{opt} for any point assigned to it. Now $|Q| = m = k/\ell$, and each point in Q gets at least ℓ points of C^* assigned to it uniquely. As such, there are at least $m\ell = k$ points of C^* assigned to some point of Q . Since $|C^*| = k$, it follows that each center in C^* gets assigned to a center in Q within distance r_{opt} . For $p \in P$, let v be its closest center in C^* . Let q be v 's center from Q in distance $\leq r_{\text{opt}}$. We have $d(p, q) \leq d(p, v) + d(v, q) \leq r_{\text{opt}} + r_{\text{opt}} = 2r_{\text{opt}}$, by the triangle inequality. As $\text{NN}_\ell(q, P) \subseteq C$, we have that $d_\ell(q, C) = d_\ell(q, P) \leq d_\ell(q, C^*) \leq r_{\text{opt}}$. By Observation 2.1, we have that, $d_\ell(p, C) \leq d_\ell(q, C) + d(p, q) \leq r_{\text{opt}} + 2r_{\text{opt}} = 3r_{\text{opt}}$. This implies $r_{\text{alg}} \leq 3r_{\text{opt}}$, a contradiction. □

Theorem 4.13 *For a given instance of $\text{FTC}(P, k, \ell)$, when using the algorithm of Gonzalez [5] for the subroutine $\mathcal{A}_c(P, m)$, the algorithm $\mathcal{A}_{\text{fc}}(P, k, \ell)$ achieves a 4-approximation to the optimal solution to $\text{FTC}(P, k, \ell)$, and a 3-approximation when $\ell|k$.*

Proof. The $\ell|k$ case follows from Lemma 4.12. If ℓ does not divide k , the proof of Lemma 4.12 needs to be changed as follows. Suppose, $k = \ell * m + r$ for some integer $0 < r < \ell$. Let $k' = \ell * m$. As in the proof of Lemma 4.12, it follows from Lemma 4.11, that if $r_{\text{alg}} > 3r_{\text{opt}}$, then at least k' centers from C^* will be within distance at most r_{opt} to a center in Q . Therefore, there are at most $k - k' = r$ centers from C^* , that are not within r_{opt} to some point in Q . However, each such center needs $\ell > r$ centers from C^* , to be within

distance r_{opt} , and so each such center must be within distance r_{opt} from one of the centers of C^* that is near a center in Q , i.e. within distance r_{opt} to some center in Q . Hence, by the triangle inequality, each center in C^* , has a center of Q within distance at most $2r_{\text{opt}}$. Repeating the argument of Lemma 4.12, with this different upper bound, we get that $r_{\text{alg}} \leq 4r_{\text{opt}}$. \square

5 Conclusions

In this paper we investigated fault-tolerant variants of the k -center and k -median clustering problems. Our algorithm achieves a $(1 + 2c)$ -approximation (resp. $(1 + 4c)$ -approximation) factor, where c is the approximation factor for the non-fault-tolerant m -center (resp. m -median) algorithm that we use as a subroutine. Using a better analysis for the case of fault-tolerant k -center, when we use Gonzalez’s algorithm as a subroutine, we showed that our algorithm has a tighter approximation ratio of 4. For fault-tolerant k -median, we get a $(5 + 4\sqrt{3} + \varepsilon) \approx 12$ -approximation algorithm, by using the recent algorithm of Li and Svensson as a subroutine [13]. We can see several questions for future research.

- The best known approximation factor for the fault-tolerant k -center problem is 2 by Chaudhuri *et al.* [3] and Khuller *et al.* [10]. Their techniques are based on the work of Hochbaum and Shmoys [8] and Krumke [12]. Our algorithm, which leads to a 4-approximation for fault-tolerant k -center is based on the 2-approximation to k -center by Gonzalez [5]. Can the algorithm or its analysis be improved to get a factor 2-approximation? Also, can we deal with the second variant of fault-tolerant k -center in the work of Khuller *et al.*— which also happens to be the version considered by Krumke and Chaudhuri *et al.*?
- The fault-tolerant k -median variant that we investigate, is very different from the work of Swamy and Shmoys [14], but their techniques are more technically involved. As we show, we reduce the fault-tolerant version to the non-fault-tolerant version for a smaller number of centers. An important question that arises is the following: Can the version considered by Swamy and Shmoys be reduced to the non-fault-tolerant version, or some variant thereof, i.e., can we use some simpler problem as an oracle to get a fault-tolerant k -median algorithm, for the version of Swamy and Shmoys?

Acknowledgements

We would like to thank Sariel Har-Peled for useful discussions, and in particular for the discussion that led us to think about this problem.

References

- [1] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k -median and related problems. In *Proc. 30th Annu. ACM Symp. Theory Comput.*, pages 106–113, 1998.
- [2] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k -median and facility location problems. In *Proc. 33rd Annu. ACM Symp. Theory Comput.*, pages 21–29, 2001.
- [3] S. Chaudhuri, N. Garg, and R. Ravi. The p -neighbor k -center problem. *Inf. Proc. Lett.*, 65(3):131–134, 1998.
- [4] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Symp. Theory Comput.*, pages 434–444, 1988.
- [5] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.
- [6] S. Har-Peled. *Geometric Approximation Algorithms*. Amer. Math. Soc., 2011.
- [7] S. Har-Peled and S. Mazumdar. Coresets for k -means and k -median clustering and their applications. In *Proc. 36th Annu. ACM Symp. Theory Comput.*, pages 291–300, 2004.
- [8] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k -center problem. *Math. of Oper. res.*, 10(2):180–184, 1985.
- [9] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proc. 34th Annu. ACM Symp. Theory Comput.*, pages 731–740, 2002.
- [10] S. Khuller, R. Pless, and Y. J. Sussmann. Fault tolerant k -center problems. *Theor. Comput. Sci.*, 242(1-2):237–245, 2000.
- [11] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean κ -median problem. In *Proc. 7th Annu. European Symp. Algorithms*, pages 378–389, 1999.
- [12] S. O. Krumke. On a generalization of the p -center problem. *Inf. Proc. Lett.*, 56:67–71, 1995.
- [13] S. Li and O. Svensson. Approximating k -median via pseudo-approximation. In *Proc. 45th Annu. ACM Symp. Theory Comput.*, page to appear, 2013.
- [14] C. Swamy and D. B. Shmoys. Fault-tolerant facility location. In *Proc. 14th ACM-SIAM Symp. Discrete Algorithms*, pages 735–736, 2003.

Open Problems from CCCG 2012

Joseph S. B. Mitchell*

On Wednesday afternoon, August 8, 2012, we held an open problem session at the *24th Canadian Conference on Computational Geometry*, in Charlottetown, Prince Edward Island, Canada. The following is a description of the problems presented and discussed, as scribed by Joe Mitchell, with follow-up comments and details written by the problem posers.

Zippered Volume

Anna Lubiw

University of Waterloo

alubiw@math.uwaterloo.ca

Given a zipper of length 1, find a simply connected 2D shape, S , with perimeter 2 such that the 3D body obtained by “zipping up” the boundary of S has maximum volume. The problem is also interesting when S is restricted to be a polygon. See Figure 1. For more information, see [16].

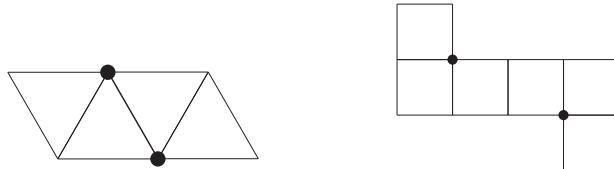


Figure 1: These unfoldings show that with a zipper of length 1 we can make a tetrahedron of side length $\frac{1}{3}$ and volume 44×10^{-4} , or a cube with side length $\frac{1}{7}$ and volume 29×10^{-4} . The start and end of the zipper are marked with dots, and the two sides of the zipper travel in opposite directions around the perimeter from the start to the end.

Many people worked on this problem during the conference, including (but not limited to) Sarah Cannon, Jean-Lou De Carufel, Thomas Hackl, Stefan Huber, Denis Khromov, Matias Korman, Joe Mitchell, Vinayak Pathak, Diane Souvaine, Selim Tawfik, Ryuhei Uehara, Hamideh Vosoughpour.

Denis Khromov suggested focusing on the final shape of the zipper in 3D, which leads to the problem of finding a curve C of length 1 that maximizes the volume of the convex hull of C . It turns out

that this problem has a long history. For a closed curve in 2D, it is the isoperimetric problem (see the wikipedia page) and the solution is a circle. For an open curve in 2D, the solution is a semicircle. For an open curve in 3D this is problem A28 in Croft, Falconer, and Guy [5]. From the exposition there and in the paper of Tilli [23] it seems that the problem is solved for curves that do not cross any plane more than 3 times. In this case, the optimum solution is a circular helix, $x = \sin(t)$, $y = \cos(t)$, $z = t/\sqrt{2}$, as t goes from 0 to 2π , which gives volume 102×10^{-4} . See Figure 2.

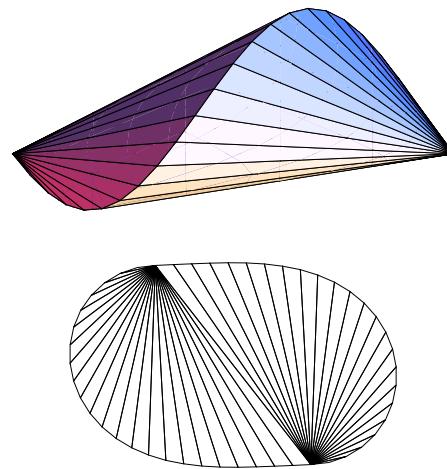


Figure 2: Among a large class of curves, the maximum volume of the convex hull, 102×10^{-4} , is achieved by a helix, with convex hull and unfolding as shown.

Some people at CCCG found (suboptimal) solutions based on cones. Sarah Cannon, Diane Souvaine and I found one with a volume of 84×10^{-4} , where the curve consists of two semicircles lying in orthogonal planes. The 3D body consists of two half-cones where each half-cone has a semicircular base and an apex above one endpoint of the semicircle. See Figure 3.

Characterize Output of Poisson-Disk Process

Scott Mitchell

Sandia National Lab

samitch@sandia.gov

(This might be considered a problem in spatial

*Stony Brook University, Stony Brook, NY 11794-3600, USA,
Joseph.Mitchell@stonybrook.edu. Partially supported by grants
 from the National Science Foundation (CCF-1018388) and the
 Binational Science Foundation (BSF 2010074).

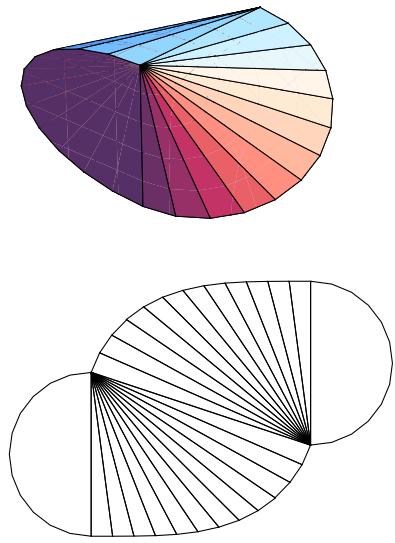


Figure 3: A volume of 84×10^{-4} is achieved by conjoined half cones.

statistics, but there are ties to Delaunay refinement and sphere packings.) Maximal Poisson-disk sampling (MPS) is a particular statistical process for generating a point cloud. The location for the next point is chosen uniformly by area at random. A point has an empty disk of radius r around it; if a new point falls into a prior point's disk, it is rejected and not added to the sample. The process continues until the sampling is maximal: the entire domain is covered by samples' disks and there is no room for another sample. Let the domain be a two-dimensional square with periodic (toroidal) boundary conditions, so there are no domain boundary issues to consider. A math definition appears in Ebeida et al. [9].

I am aware of no analytic description of what the correct output of MPS is supposed to be. I haven't even seen an experimental characterization! As such, currently for an algorithm to be correct, it must be step-by-step equivalent to the statistical process. For an example algorithm like this, see again [9]. A characterization of the output is important because it would enable the design of more efficient algorithms. A metaphor is that bubble-sort is a process, but the characterization of its output as "sorted order" allows the discovery of e.g. quicksort to generate "sorted order" more efficiently.

The computer graphics community typically measures the output of MPS by generating Fourier transform pictures of the output. See "Point Set Analysis" [17], for software and paper references for a standard way of generating these pictures.

My understanding of PSA follows. The vectors of distances between all pairs of points are calculated. The Fourier transform of the distance vectors are taken and displayed, and a picture with oscillating dark and light rings is expected. Integrating this transform over concentric circles produces a one-dimensional graph by increasing radius. (A nuance is how to bin distances to generate smooth pictures.) Figure 4 top shows the kinds of pictures the Graphics community expects to see for MPS.

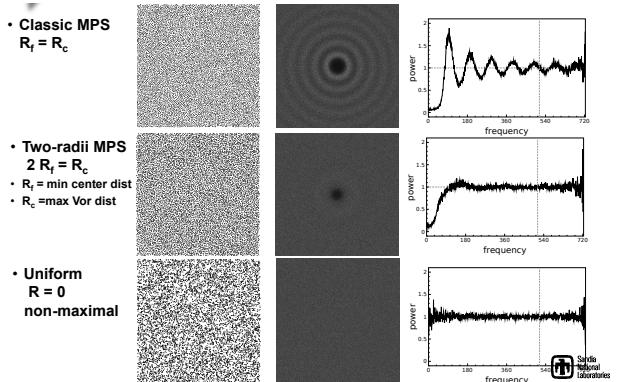


Figure 4: Point clouds visualized using PSA. Top is standard MPS, and middle two-radii MPS from CCCG 2012. The bottom is a uniform random point cloud without inhibition disks, using about the same number of points.

Subproblem A: Can you characterize the PSA pictures for MPS, especially Figure 4 top right? What is the mean location and height of the peaks? What is their standard deviation? Is the distribution around the mean normal? (Recall MPS is a random process.) Perhaps an experimental characterization is an easier place to start than an analytic characterization.

Subproblem B: Is some variant of MPS better than standard MPS for texture synthesis graphics applications? At CCCG 2012 I presented a paper "Variable Radii MPS." The two-radii MPS variant generates a spectrum with less oscillations; see Figure 4 middle. We suspect, but don't know for sure, if this is better for applications.

MPS produces a sphere packing, halve the disk radii r then the disks do not overlap. This is a well-spaced point set. Delaunay refinement also produces a well-spaced point set. Sometimes the PSA pictures of the output of Delaunay refinement look similar to MPS, sometimes not, depending on

the target edge length, angle threshold, the use of off-centers, etc.; see Figure 5.

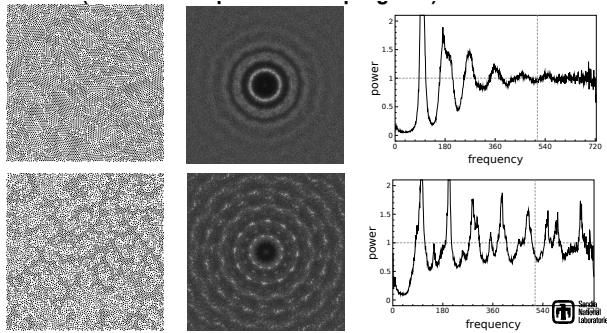


Figure 5: Delaunay refinement (Triangle) point clouds from particular choices of target edge lengths and angles, visualized using the PSA tool. Top left we see patches of hexagonal packings, and bottom left we see circular patterns of jumps in the point spacing. In the Fourier transform, middle column, in the top the rings are more pronounced than form MPS; in the bottom we see bright spots which indicate preferential directions, meaning nearby points are more dense in certain directions than others. In the radial average, right, on both the top and bottom we see accentuated spikes.

Subproblem C: characterize the PSA pictures (Fourier spectrum) of the output of Delaunay refinement and its variants.

In computational geometry we often measure point sets by the angles and edge length histograms in a Delaunay triangulation of the points. These histograms are different for MPS point clouds than for Delaunay refinement output; see Figure 6.

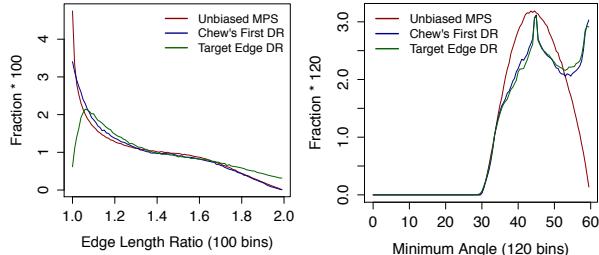


Figure 6: Computational Geometry measures of point clouds. Edge length and angle histograms of DR and MPS output. The minimum angle is the smallest angle of each triangle. The edge length ratio is the ratio of the length of Delaunay edges to the disc radius (MPS) or maximum Delaunay circumradius (DR). In both MPS and DR, the theories guarantee $r < |e| < 2r$.

Subproblem D: Characterize MPS output using computational geometry measures of Delaunay triangulation edge lengths and angle distributions.

Bonus subproblem E: do these problems for dimensions other than 2. Three to five dimensions have some graphics applications.

Bonus subproblem F: characterize the effect of the domain boundary, for non-periodic domains.

Partners: Alexander Rand, Mohamed Ebeida, John Owens, Anjul Patney, Andrew Davidson, Chandrajit Bajaj.

Hiding a Cycle

Paz Carmi

Ben Gurion University

carmip@cs.bgu.ac.il

Pat Morin had posed at CCCG'2007 the following question: Is it possible that a polygonal cycle C in 3D, with each edge axis-parallel, can project orthogonally onto each of the three coordinate planes in such a way that each projection does not have a cycle?

A solution to the problem appears on the cover of the book [24]. However, the three projections in this figure are (rectilinear) trees. Is it possible that all three projections of C are paths?

Divide and Conquer

Jérémie Barbay

Universidad de Chile

jbarbay@dcc.uchile.cl

Adaptive Divide and Conquer: For several problems, divide and conquer algorithms of complexity $O(n \log n)$ and $O(nk)$, for input size n and some additional parameter k , yield algorithms of complexity $O(n(1 + \lg k))$, or even better, $O(n(1 + \mathcal{H}(n_1, \dots, n_k)))$:

- **Selecting k elements** of respective ranks (r_1, \dots, r_k) in an unsorted array of n elements can obviously be done by sorting the n elements in $O(n \lg n)$ comparisons. It can also be performed in $O(nk)$ comparisons by using k times the median of median quick select algorithm. Yet in 1981 Dobkin et al. [7] showed that those k ranks can be computed in $O(n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1}))) \subset O(n(1 + \lg k))$ comparisons and overall time, and in 2006 Kaligosi et al. [13] showed that those k ranks can be computed in $n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1})) + o(n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1}))) + O(n)$ comparisons and $O(n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1})))$ overall time. This result is *input order oblivious instance optimal*, in the sense that no algorithm can perform better in the worst (and average) input order, for any set of values.

- In 1972, Graham [11] showed a reduction of the computational complexity of the **convex hull** to sorting, yielding a complexity of $O(n \log n)$, which is optimal in the worst case over instances composed of n points. Yet in 1973, Jarvis [12] showed that this was not optimal on instances where the number h of points in the convex hull is smaller than $\log n$, via an algorithm running in time within $O(hn) \subset O(n^2)$. It is not until 1986 that Kirkpatrick and Seidel [15] solved the paradox through an algorithm running in time within $O(n(1 + \log h))$, which analysis Afshani et al. [1] later improved to $O(n(1 + \mathcal{H}(n_1, \dots, n_h)))$, where $\mathcal{H}(n_1, \dots, n_h) \equiv \sum_{i=1}^h \frac{n_i}{n} \log \frac{n}{n_i} \leq \log_2 h$ is the minimal entropy of a certificate of the instance.
- In 1983 Reif [19] described an algorithm computing the **minimal cut between two vertices** s and t of a planar graph over n vertices in time $O(n \log^2 n)$, using the fact that the minimal cut corresponds to a cycle of minimal total weight in the dual of the planar graph, which is intersected at most once by the minimal path between s^* and t^* . Another algorithm was known to perform in linear time when s and t share a face. In 2011, Kaplan et al. [14] generalized this result to yield an algorithm computing the minimal cut in $O(n(1 + \log p))$ operations, where p is the minimum number of edges crossed by a curve joining s to t , or equivalently the minimum number of edges from the face s^* to the face t^* in the dual of the planar graph.
- In 1952, Huffman showed that a **prefix free code of minimal redundancy** for n weighted symbols can be computed in $O(n \lg n)$ algebraic operations (In 1976, van Leeuwen showed that an equivalent code can be obtained in linear time when the weights are sorted.). In 2006 Belal et al. [3] claimed an algorithm computing a code of equivalent redundancy in $O(nk)$ algebraic operations, where k is the number of distinct code-lengths in a code of minimal redundancy.
- In an undirected planar graph of n vertices, the **maximum flow** between two vertices s and t can be computed in time $O(n \lg n)$ via $O(n)$ augmenting paths, which can be improved to $O(n(1 + \lg k))$ time when k faces separate s and t . This result is optimal in the worst case over instances for fixed values of n and k .

We ask the following questions:

1. For which other problems are there at once an algorithm working in time $O(n \lg n)$ and an algorithm working in $O(nk)$, for some parameter k , but no known algorithm running in $O(n(1 + \lg k))$?
2. For which other problems is there an algorithm running in $O(n(1 + \lg k))$, and a potential for an algorithm working in time $O(n(1 + \mathcal{H}(n_1, \dots, n_k)))$, where (n_1, \dots, n_k) is a vector describing the difficulty of the instance in a finer way than k and $\mathcal{H}(n_1, \dots, n_k)$ is its entropy?
3. For the problems where there is an algorithm running in time $O(n(1 + \mathcal{H}(n_1, \dots, n_k)))$, can this be improved to $n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1})) + o(n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1}))) + O(n)$ comparisons and $O(n(1 + \mathcal{H}(r_1, r_2 - r_1, \dots, r_k - r_{k-1})))$ overall time, as Kaligosi et al. did for the multi select problem?
4. Can we identify a general principle at work for those problems (e.g. Input Order Oblivious Instance Optimal Complexity?), or are there counter examples, for example in the form of a problem for which there is an algorithm running in time $O(n \lg n)$, an algorithm taking advantage of particular cases running in $O(nk)$, but provably no algorithm running in $o(n \min\{\lg n, k\})$ in the worst case over all instances of fixed size n and fixed parameter value k ?

Minimum Interference Networks

Pat Morin

Carleton University

morin@scs.carleton.ca

Given a set S of n points in \mathbb{R}^d , the goal is to construct a connected network on S in order to minimize the (*maximum receiver-centric*) *interference*, which is the maximum depth in the set of disks, centered on each $p_i \in S$, of radius equal to the length of the longest edge incident to p_i .

In two and higher dimensions, achieving a better than $5/4$ -approximation is NP-hard, as was shown by Buchin [4]. In 1D, the problem is not known to be NP-hard, yet the only result is a polynomial-time $O(n^{1/4})$ -approximation [20]. Thus, in any dimension, the following question is open: Does there exist a polynomial time $o(n^{1/4})$ -approximation algorithm for constructing a minimum interference network?

Another problem in this area involves bounding the minimum-interference network of a point set by the interference of its minimum spanning tree. This is captured by the following conjecture: If

the minimum spanning tree of a point set, S , has interference k , then there exists a connected network on S with interference $O(\sqrt{k})$. If proven, this would resolve a question, from [6], on constructing minimum-interference networks for random point sets.

Generalized MST: 2-GMST

Bob Fraser

University of Waterloo

r3fraser@uwaterloo.ca

In the generalized (or “one-of-a-set” or “group”) MST, we are given a collection of n finite sets of points, S_1, \dots, S_n , and the goal is to compute a minimum-weight spanning tree that visits at least one point of each set S_i . Pop [18, Theorem 4.3] described a 2δ -approximate algorithm for generalized MST, where δ is the maximum cardinality of any imprecise vertex. The algorithm solves the linear programming relaxation of the integer programming formulation of the problem, and then chooses a spanning tree from the points in the solution. This result follows the approximation framework used by Slavík [21, 22] to establish approximation algorithms for the Generalized Travelling Salesman Problem and the Group Steiner Tree problem with approximation factors of $3\delta/2$ and 2δ , respectively.

We ask here about the 2-GMST in the Euclidean plane, in which each S_i is a pair of points; in fact, we are interested in the special case in which each pair is vertical (i.e., each set S_i consists of two points having the same x -coordinate). For the case that $|S_i| = 2$, for all i , a 4-approximation is immediate from the general 2δ -approximation algorithm of Pop mentioned above. Can geometry be exploited to do better?

Note that the generalized problem is APX-hard (Dror and Orlin[8]), and the restricted version described here (2-GMST) is NP-hard [10, §10.4] (the proof of hardness also rules out the possibility of an FPTAS).

Two Problems

Pankaj Agarwal

Duke University

pankaj@cs.duke.edu

(a). *Forcing a vertex minimum of a terrain.* Given a triangulated terrain with (piecewise-linear) height function $h(x, y)$, and given a vertex v of the terrain, our goal is to compute a new terrain, $h'(x, y)$, such that v is a unique minimum of h' and h' has no critical points other than v . The objective function is to select h' to minimize $\|h - h'\|$ (or $(\|h - h'\|)^2$).

(b). *Separating 3D polytopes (a classic problem).* Given two convex polytopes, P_1 and P_2 in \mathbb{R}^3 , each with a Dobkin-Kirkpatrick hierarchy, our goal is to compute a separating plane, or report that none exists, for P_1 and P_2 . Known methods yield time $O(\log m \log n)$, where m and n are the complexities of P_1 and P_2 . Is it possible to improve the time bound to $O(\log m + \log n)$?

Redrawing a Triangulation

Csaba Tóth

University of Calgary

cdtoth@ucalgary.ca

Given a triangulation T of n points in the plane. Let ℓ be a line intersecting T , crossing edges (e_1, e_2, \dots, e_k) , of T at points (p_1, p_2, \dots, p_k) , in order along ℓ . Now, consider another line, L , with points (q_1, q_2, \dots, q_k) , in order along L . Is it always possible to draw a triangulation T' , equivalent (in the sense of graph isomorphism) to T , with L crossing the edges $(e'_1, e'_2, \dots, e'_k)$ (the mappings of the e_i 's) at the points (q_1, q_2, \dots, q_k) ?

Wavelets and the Golden Ratio

Braxton Carrigan

Auburn University

bac0004@auburn.edu

Given a triangulation in which every triangle is isosceles. The goal is to find a subtriangulation by adding new vertices along edges (while keeping the cell complex property) in order to preserve the ratio of side lengths (in Golden ratio).

Hamiltonian Tetrahedralization of a 3-Polytope

Joe Mitchell

Stony Brook University

Joseph.Mitchell@stonybrook.edu

I conjecture that every three-dimensional convex polytope has a tetrahedralization (without adding Steiner points) whose dual graph has a Hamiltonian path. If true, then every finite point set in 3D has a tetrahedralization (without Steiner points) that is Hamiltonian.

For related work, see Arkin et al. [2].

References

- [1] P. Afshani, J. Barbay, and T. Chan. Instance-optimal geometric algorithms. In *Proc. 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009.
- [2] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. Skiena. Hamiltonian triangulations for fast rendering. *The Visual Computer*, 12(9):429–444, 1996.

- [3] A. A. Belal and A. Elmasry. Distribution-sensitive construction of minimum-redundancy prefix codes. In B. Durand and W. Thomas, editors, *23rd International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 3884 of *Lecture Notes in Computer Science*, pages 92–103. Springer, 2006.
- [4] K. Buchin. Minimizing the maximum interference is hard. *CoRR*, abs/0802.2134, 2008.
- [5] H. T. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry*, Springer-Verlag, 1991.
- [6] L. Devroye and P. Morin. A note on interference in random point sets. In *Proc. 24th Canadian Conference on Computational Geometry (CCCG)*, pages 201–206, 2012.
- [7] D. P. Dobkin and J. I. Munro. Optimal time minimal space selection algorithms. *Journal of the ACM*, 28(3):454–461, 1981.
- [8] M. Dror and J. B. Orlin. Combinatorial Optimization with Explicit Delineation of the Ground Set by a Collection of Subsets, *SIAM Journal on Discrete Mathematics*, 21(4):1019–1034, 2008.
- [9] M. S. Ebeida, A. A. Davidson, A. Patney, P. M. Knupp, S. A. Mitchell, and J. D. Owens. Efficient maximal poisson-disk sampling. *ACM Transactions on Graphics*, 30(4):49, 2011.
- [10] R. Fraser. *Algorithms for Geometric Covering and Piercing Problems*. PhD thesis, University of Waterloo, 2012.
- [11] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.
- [12] R. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.
- [13] K. Kaligosi, K. Mehlhorn, J. I. Munro, and P. Sanders. Towards optimal multiple selection. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2005.
- [14] H. Kaplan and Y. Nussbaum. Minimum $s-t$ cut in undirected planar graphs when the source and the sink are close. In T. Schwentick and C. Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 117–128, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [15] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 1986. 15(1):287–299.
- [16] A. Lubiw, E. D. Demaine, M. L. Demaine, A. Shalit, and J. Shallit. Zipper unfoldings of polyhedral complexes. In *Proc. 22nd Annual Canadian Conference on Computational Geometry (CCCG)*, pages 219–222, 2010.
- [17] Point Set Analysis, <http://code.google.com/p/psa/>.
- [18] P. Pop. *The Generalized Minimum Spanning Tree Problem*. PhD thesis, University of Twente, 2002.
- [19] J. Reif. Minimum $s-t$ cut of a planar undirected network in $o(n \log n)$ time. In S. Even and O. Kariv, editors, *Automata, Languages and Programming*, volume 115 of *Lecture Notes in Computer Science*, pages 56–67. Springer Berlin / Heidelberg, 1981.
- [20] P. von Rickenbach, S. Schmid, R. Wattenhofer, and A. Zollinger. A robust interference model for wireless ad-hoc networks. In *Proc. 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*. IEEE Computer Society, 2005.
- [21] P. Slavík. The errand scheduling problem. Computer Science Technical Report 97-2, State University of New York at Buffalo, 1997.
- [22] P. Slavík. *Approximation Algorithms For Set Cover And Related Problems*. PhD thesis, State University of New York at Buffalo, 1998.
- [23] P. Tilli. *Transactions of the American Mathematical Society*, 2010. 362:4497–4509.
- [24] P. Winkler. *Mathematical Mind Benders*. A K Peters/CRC Press, 2007.

Quantifying Design: the geometric properties behind designer choices and human perception

Alla Sheffer

University of British Columbia, Canada

Over centuries artists and designers had developed effective tools to communicate notions of shape to the general public. Even coarse artistic sketches are surprisingly effective at unambiguously conveying complex 3D shapes. While artistic guidelines provide many useful hints as to how to describe shape effectively, very little is known as to why these tools are effective and what geometric properties of the described shapes they capture.

Our work combines artistic guidelines and insights from perception literature to introduce an explicit mathematical formulation of the relationships between the communication tools used by artists and the geometric properties they aim to convey. In this talk I'll discuss the application of the developed mathematical formalism to several challenging computer graphics problems including shading of concept sketches and surfacing of artist designed 3D wireframe models.

Universal Point Sets for Planar Graph Drawings with Circular Arcs

Patrizio Angelini* David Eppstein†
 Tamara Mchedlidze‡ Monique Teillaud** Alexander Wolff††

Abstract

We prove that there exists a set S of n points in the plane such that every n -vertex planar graph G admits a plane drawing in which every vertex of G is placed on a distinct point of S and every edge of G is drawn as a circular arc.

1 Introduction

It is a classic result of graph theory that every planar graph has a plane *straight-line drawing*, that is, a drawing where vertices are mapped to points in the plane and edges to straight-line segments connecting the corresponding points (achieved independently by Wagner, Fáry, and Stein). Tutte [21] presented the first algorithm, the *barycentric method*, that produces such drawings. Unfortunately, the barycentric method can produce edges whose lengths are exponentially far from each other. Therefore, Rosenstiehl and Tarjan [19] asked whether every planar graph has a plane straight-line drawing where vertices lie on an integer grid of polynomial size. De Fraysseix, Pach, and Pollack [5] and, independently, Schnyder [20] answered this question in the affirmative. Their (very different) methods yield drawings of n -vertex planar graphs on a grid of size $\Theta(n) \times \Theta(n)$, and there are graphs (the so-called “nested triangles”) that require this grid size [10]. Later, it was apparently Mohar (according to Pach [6]) who generalized the grid question to the following problem: What is

the smallest size $f(n)$ of a *universal point set* for plane straight-line drawings of n -vertex planar graphs, that is, the smallest size (as a function of n) of a point set S such that every n -vertex planar graph G admits a plane straight-line drawing in which the vertices of G are mapped to points in S ? The question is listed as problem #45 in the Open Problems Project [6]. Despite more than twenty years of research efforts, the best known lower bound for the value of $f(n)$ is linear in n [4, 17, 18], while the best known upper bound is only quadratic in n , as established by de Fraysseix et al. [5] and Schnyder [20]. Universal point sets for plane straight-line drawings of planar graphs require more than n points whenever $n \geq 15$ [3]. Recently, universal point sets with $o(n^2)$ points have been proved to exist for straight-line planar drawings of several subclasses of planar graphs generalizing outerplanar graphs. Namely, an upper bound of $O(n(\log n / \log \log n)^2)$ has been proven for *simply-nested planar graphs* [1] and an upper bound of $O(n^{5/3})$ for *planar 3-trees* [14], which extends to *planar 2-trees* and hence to *series-parallel graphs*.

Universal point sets have also been studied with respect to different drawing standards. For example, Everett et al. [13] showed that there exist sets of n points that are universal for *plane poly-line drawings with one bend per edge* of n -vertex planar graphs. On the other hand, if bend-points are required to be placed on the point-set, universal point-sets exist of size $O(n^2 / \log n)$ for drawings with one bend per edge, of size $O(n \log n)$ for drawings with two bends per edge, and of size $O(n)$ for drawings with three bends per edge [11].

However, smooth curves may be easier for the eye to follow and more aesthetic than poly-lines. Graph Drawing researchers have long observed that poly-lines may be made smooth by replacing each bend with a smooth curve tangent to the two adjacent line segments [7, 15]. Bekos et al. [2] formalized this observation by considering smooth curves made of line segments and circular arcs; they define the *curve complexity* of such a curve to be the number of segments and arcs it contains. A poly-line drawing with s segments per edge may be transformed into a smooth drawing with curve complexity at most $2s - 1$, but Bekos et al. [2] observed that in many cases the curve complexity can be made smaller than this bound. For instance, replacing poly-lines by curves

*Dipartimento di Ingegneria, Roma Tre University, [angolini@dia.uniroma3.it](mailto:angelini@dia.uniroma3.it)

†Computer Science Department, University of California, Irvine, eppstein@ics.uci.edu. D.E. was supported in part by the National Science Foundation under grants 0830403 and 1217322, and by the Office of Naval Research under MURI grant N00014-08-1-1015.

‡School of Information Technology, The University of Sydney, brillo@it.usyd.edu.au

§Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, mk@informatik.uni.tuebingen.de

¶INRIA Nancy Grand Est – Loria, lazard@loria.fr

||Institute of Theoretical Informatics, Karlsruhe Institute of Technology, mched@iti.uka.de

**INRIA Sophia Antipolis – Méditerranée, monique.teillaud@inria.fr

††Lehrstuhl für Informatik I, Universität Würzburg, www1.informatik.uni-wuerzburg.de/en/staff/wolff.alexander
 A.W. acknowledges support by the ESF EuroGIGA project GraDR (DFG grant Wo 758/5-1).

in the construction of Everett et al. [13] would give rise to a drawing of curve complexity 3, but in fact every set of n collinear points is universal for smooth piecewise-circular drawings with curve complexity 2, as can be derived from the existence of topological book embeddings of planar graphs [8, 16, 2]. A *monotone topological book embedding* of a graph G is a drawing of G such that the vertices lie on a horizontal line, called *spine*, and the edges are represented by non-crossing curves, monotonically increasing in the direction of the spine. In [8, 16], it was shown that every planar graph has a monotone topological book embedding where each edge crosses the spine exactly once and consists of two semi-circles, one below and one above the spine (see Figure 2).

The difficulty of the problem of constructing a universal point set of a linear size for straight-line drawings, the aesthetical properties of smooth curves, the recent developments on drawing planar graphs with circular arcs (see, for example, [2, 12]), and the existence of universal sets of n points for drawings of planar graphs with curve complexity 2 [13] naturally give rise to the question of whether there exists a universal set of n points for drawings of planar graphs with curve complexity 1, that is, for drawings in which every edge is drawn as a single circular arc. In this paper, we answer this question in the affirmative.

We prove the existence of set S of n points on the parabolic arc $\mathcal{P} = \{(x, y) : x \geq 0, y = -x^2\}$ such that every n -vertex planar graph G can be drawn with the vertices mapped to S and the edges mapped to non-crossing circular arcs. In the same spirit as Everett et al. [13], we draw G in two steps. In the first step, we construct a monotone topological book embedding of G . In the second step, we map the vertices of G to the points in S in the same order as they appear on the spine of the book embedding.

2 Circular Arcs Between Points on a Parabola

In this section, we investigate geometric properties of circular-arc drawings whose vertices lie on the parabolic arc \mathcal{P} .

In the following, when we say that a point is *to the left* of another point, we mean that the x -coordinate of the former is smaller than that of the latter. However, when we say that an arc is to the left of a point q , we mean that all the intersection points of the arc with the horizontal line through q are to the left of q . We define similarly *to the right*, *above*, and *below*, and we naturally extend these definition to non-crossing pairs of arcs. We denote by $\mathcal{C}(p, q, r)$ the circle through three points p, q , and r .

We start by stating a classic property of parabolas and circles.

Lemma 1 *For every three points p, q , and r on \mathcal{P} with increasing x -coordinates, the circular arc from p to r and through q is below \mathcal{P} between p and q and above \mathcal{P} between q and r (see Figure 1).*

Proof. We first observe that a circle intersects \mathcal{P} in at most three points with positive x -coordinates (counted with multiplicity). Indeed, substituting y by $-x^2$ in the circle equation yields a degree-4 equation in x with no monomial of degree 3. There are thus at most three changes of sign in the sequence of coefficients, and Descartes' rule of signs implies that there are at most three positive roots, counted with multiplicity.

We now consider three points p, q , and r on \mathcal{P} and consider circle $\mathcal{C}(p, q, r)$. Since there is no other point of intersection with positive x -coordinate, and since the circle is bounded and the parabolic arc is not, the circular arc to the right of r is below the parabolic arc. The result follows since $\mathcal{C}(p, q, r)$ crosses \mathcal{P} at p, q , and r (since, otherwise, the number of intersection points with positive x -coordinates and counted with multiplicity would be larger than three). \square

Given six points $p_0 = (x_0, y_0), \dots, p_5 = (x_5, y_5)$ in this order on \mathcal{P} (that is, $x_0 \leq x_1 \leq \dots \leq x_5$), we consider two circular arcs (see Figure 1); $C_{0,3,4}$ (red) goes through the ordered points p_0, p_3, p_4 and $C_{1,2,5}$ (blue) goes through p_1, p_2, p_5 . We assume that the three points defining each arc are pairwise distinct. It should be stressed that these arcs may not be x -monotone.¹ The two circular arcs are, however, y -monotone—for $C_{0,3,4}$ we argue as follows; the argument for $C_{1,2,5}$ is similar: By Lemma 1, p_0 lies on the right half-circle of $\mathcal{C}(p_0, p_3, p_4)$, and p_3 and p_4 are to the right of p_0 .

We will prove, in Lemma 4, that the arcs $C_{0,3,4}$ and $C_{1,2,5}$ do not intersect each other if the x -coordinate of p_i is at least twice that of p_{i-1} for $i = 3, 4$. For that purpose, we first consider, in the two next lemmas, the special cases where these arcs share one of their endpoints.

Lemma 2 *If $p_4 = p_5$ and $x_3 \geq x_1 + x_2$, the two circular arcs $C_{0,3,4}$ and $C_{1,2,5}$ intersect only at $p_4 = p_5$.*

Proof. Refer to Figure 1(a). We first observe that, by Lemma 1, the circular arc $C_{0,3,4}$ is below \mathcal{P} in a neighborhood of p_0 , it crosses \mathcal{P} at p_3 , and it lies above \mathcal{P} in a neighborhood of p_4 . Similarly $C_{1,2,5}$ is below \mathcal{P} in a neighborhood of p_1 , it crosses \mathcal{P} at p_2 , and it lies above \mathcal{P} in a neighborhood of p_5 .

We now argue that the two arcs $C_{0,3,4}$ and $C_{1,2,5}$ intersect at a point other than $p_4 = p_5$ if and only if the

¹ This could be seen by considering, for instance, the limit case of a circle where p_0 and p_3 lie at the origin and the x -coordinate of p_4 is larger than one. This circle is centered at $(0, -a)$ with $a > 1$. Since $-a > -a^2$, the rightmost point $(a, -a)$ of the circle is above the parabola $y = -x^2$, thus it lies on $C_{0,3,4}$ by Lemma 1.

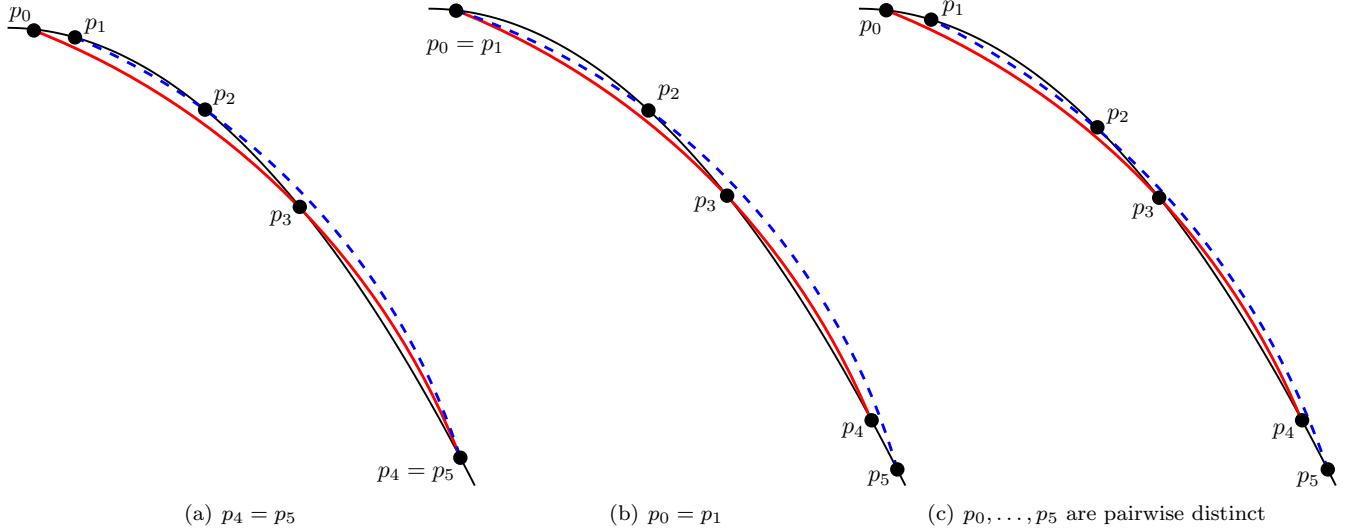


Figure 1: Three configurations of relative position of the circular arcs $C_{0,3,4}$ (red) and $C_{1,2,5}$ (blue dashed) defined by six points p_0, \dots, p_5 lying in that order on \mathcal{P} . For readability, the figure is not to scale.

(red) arc $C_{0,3,4}$ is to the right of the (blue) arc $C_{1,2,5}$ in a neighborhood of p_4 . Since the (red) arc $C_{0,3,4}$ is below \mathcal{P} in a neighborhood of p_0 , and $C_{0,3,4}$ does not intersect \mathcal{P} between p_0 and p_1 (by Lemma 1), the (red) arc $C_{0,3,4}$ is to the left of p_1 . On the other hand, the two circular arcs intersect at most once other than at p_4 (since circles intersect at most twice). Hence, if they intersect at a point q other than p_4 , their horizontal ordering changes in a neighborhood of q and thus the (red) arc $C_{0,3,4}$ is to the right of the (blue) arc $C_{1,2,5}$ in a neighborhood of p_4 .

As a consequence, we can assume without loss of generality that p_0 is at the origin $O = (0, 0)$ (that is, the topmost point of \mathcal{P}). This can be seen as follows. First, by Lemma 1, the origin is inside $\mathcal{C}(p_0, p_3, p_4)$. Furthermore, since the origin is above p_3 and p_4 , the arc p_3p_4 of $\mathcal{C}(O, p_3, p_4)$ lies to the right of the arc p_3p_4 of $\mathcal{C}(p_0, p_3, p_4)$. It follows that if $C_{0,3,4}$ is to the right of $C_{1,2,5}$ in a neighborhood of p_4 , it remains to the right if p_0 is placed at the origin. Hence, in the sequel, we can assume that $x_0 = 0$.

We now prove that if $x_3 \geq x_1 + x_2$, then the tangents at $p_4 = p_5$ of the two circular arcs $C_{0,3,4}$ and $C_{1,2,5}$ are distinct for any position of $p_4 = p_5$ to the right of p_3 on \mathcal{P} .

The following calculations are done in Maple. We consider the equation of $\mathcal{C}(p_0, p_3, p_4)$, which is the determinant

$$\begin{bmatrix} x_0 & -x_2^2 & x_0^2 + x_4^4 & 1 \\ x_3 & -x_3^2 & x_3^2 + x_3^4 & 1 \\ x_4 & -x_4^2 & x_4^2 + x_4^4 & 1 \\ x & y & x^2 + y^2 & 1 \end{bmatrix}$$

and similarly for $\mathcal{C}(p_1, p_2, p_4 = p_5)$. The normals to

these circles at p_4 are the gradient of their implicit equations evaluated at p_4 . We then compute the cross product of these two vectors; more precisely, the last coordinate of the cross product, that is, $M_x N_y - N_x M_y$, where (M_x, M_y) and (N_x, N_y) are the normal vectors.

This expression can be factorized such that it is the product of two terms. The first is the term $x_3 x_4 (x_3 - x_4)(x_2 - x_4)(x_1 - x_2)$, which does not vanish if p_0, \dots, p_4 are pairwise distinct. The second is the following term, which we view as a polynomial in x_4 whose coefficients depend on x_1, x_2 , and x_3 :

$$\begin{aligned} & (x_3 - x_1 - x_2) x_4^4 \\ & + (x_1 + x_2 + x_3) (x_3 - x_1 - x_2) x_4^3 \\ & + (1 + x_1 x_2) (x_3 - x_1 - x_2) x_4^2 \\ & + (x_1 x_2 x_3^2 + x_1 x_2^2 x_3 + x_1^2 x_2 x_3 + x_3^2 - x_1^2 - x_2^2) x_4 \\ & + x_1 x_2 (1 + x_3^2) (x_1 + x_2). \end{aligned}$$

All coefficients are non-negative since $x_3 \geq x_1 + x_2$. Thus, the polynomial has no positive real root. In other words, the two normals are never collinear. Now, considering the limit case where $p_4 = p_3$, the (red) circle $\mathcal{C}(p_0, p_3, p_4)$ is tangent to \mathcal{P} and since, by Lemma 1, the (blue) arc $C_{1,2,5}$ is above and thus to the right of \mathcal{P} in a neighborhood of $p_4 = p_5$ (and is not tangent to \mathcal{P} if $p_2 \neq p_5$), the (blue) arc $C_{1,2,5}$ is to the right of the (red) arc $C_{0,3,4}$ in a neighborhood of p_4 . Hence, the two arcs $C_{0,3,4}$ and $C_{1,2,5}$ do not intersect except at p_4 . \square

Lemma 3 *If $p_0 = p_1$, $x_0 \geq 1$, $x_3 \geq 2x_2$ and $x_4 \geq x_0 + x_3$, the two circular arcs $C_{0,3,4}$ and $C_{1,2,5}$ intersect only at $p_0 = p_1$.*

Proof. Similarly as in the proof of Lemma 2, the two arcs $C_{0,3,4}$ and $C_{1,2,5}$ intersect at a point other than

$p_0 = p_1$ if and only if the (red) arc $C_{0,3,4}$ is to the right of the (blue) arc $C_{1,2,5}$ in a neighborhood of p_0 (see Figure 1(b)).

Furthermore, we can assume without loss of generality that p_5 is at infinity, which means that $C_{1,2,5}$ is the (straight) ray from $p_0 = p_1$ through p_2 . Indeed, for any point p'_5 that lies on \mathcal{P} to the right of p_5 , point p'_5 lies outside the $\mathcal{C}(p_1, p_2, p_5)$ by Lemma 1. Furthermore, since p'_5 lies below p_1 and p_2 , the arc through p_1, p_2 , and p'_5 (in order) lies to the left of $C_{1,2,5}$ between p_1 and p_2 . Hence, if the (blue) arc $C_{1,2,5}$ is to the left of the (red) arc $C_{0,3,4}$ in a neighborhood of p_0 , it remains to the left if p_5 is at infinity.

Now, similarly to the proof of Lemma 2, we prove that the tangents at $p_0 = p_1$ of $C_{0,3,4}$ and $C_{1,2,5}$ never coincide. With the above assumption, this is equivalent to showing that the normal to $C_{0,3,4}$ at p_0 is never orthogonal to the segment p_1p_2 . The corresponding dot product (computed in Maple) is equal to

$$\begin{aligned} & (x_4 - x_3)(x_4 - x_0)(x_3 - x_0)(x_2 - x_0) \\ & \left((x_3 - x_2)x_4^2 + (x_3 - x_2)(x_0 + x_3)x_4 + \right. \\ & \left. ((x_0^2 - 1 - x_3x_0 - x_3^2)x_2 + x_0^3 + x_0) \right). \end{aligned}$$

The first four terms never vanish and we want to show that the last term, seen as a polynomial in x_4 , has no root x_4 larger than $x_0 + x_3$ (it can be shown that this polynomial has a positive root). For that purpose, we make the change of variable $x_4 = t + x_0 + x_3$ which maps the interval $(x_0 + x_3, +\infty)$ of x_4 to the interval $(0, +\infty)$ of t and maps the above degree-2 polynomial in x_4 to

$$\begin{aligned} & (x_3 - x_2)t^2 + 3(x_3 - x_2)(x_0 + x_3)t - \\ & (1 + x_0^2 - 5x_0x_3 + 3x_3^2)x_2 + \\ & x_0 + 4x_0x_3^2 + x_0^3 + 2x_3^3 + 2x_0^2x_3 \end{aligned}$$

whose first and second coefficients are positive and whose last coefficient is positive for any $x_2 \in [x_0, x_3/2]$ since it is linear in x_2 and takes value $x_3(3x_0 + 2x_3)(x_3 - x_0)$ at x_0 and value $\frac{1}{2}x_3(-1 + x_3^2 + 3x_0^2 + 3x_0x_3) + x_0 + x_3^3$ at $x_3/2$ (which is positive since $x_0 \geq 1$).² Hence, if $x_3 \geq 2x_2$, all coefficients of this polynomial are positive, which implies that it has no positive roots. This, in turn, means that the initial degree-2 polynomial in x_4 has no root larger than $x_0 + x_3$.

This implies that there is no position of the points $p_0 = p_1, p_2, \dots, p_5$ such that $x_3 \geq 2x_2$, $x_4 \geq x_0 + x_3$ and such that the tangent to $C_{0,3,4}$ is collinear with p_0p_2 . Furthermore, at the limit case where $p_2 = p_0$, the segment p_0p_2 is tangent to \mathcal{P} , and $C_{0,3,4}$ is below and to the left of that tangent in a neighborhood of p_0 (by Lemma 1). Hence, for any position of the

²Note that the last coefficient is negative when $x_2 = x_3$ which is why we consider x_2 in the range $[x_0, x_3/2]$.

points $p_0 = p_1, p_2, \dots, p_5$ (as defined above) such that $x_3 \geq 2x_2$, $x_4 \geq x_0 + x_3$, the (red) circular arc $C_{0,3,4}$ is to the left of the segment p_1p_2 in a neighborhood of p_0 . Finally, as argued above when we considered p_5 at infinity, this implies that for any position of the points $p_0 = p_1, p_2, \dots, p_5$ such that $x_3 \geq 2x_2$ and $x_4 \geq x_0 + x_3$, the (red) circular arc $C_{0,3,4}$ is to the left of the (blue) circular arc $C_{1,2,5}$ in a neighborhood of $p_0 = p_1$. This concludes the proof since we have proved that this is equivalent to the property that the arcs $C_{0,3,4}$ and $C_{1,2,5}$ intersect only at $p_0 = p_1$. \square

Lemma 4 *If p_0, \dots, p_5 are pairwise disjoint and $x_i \geq 2x_{i-1}$ for $i = 3, 4$, the two circular arcs $C_{0,3,4}$ and $C_{1,2,5}$ do not intersect.*

Proof. We refer to Figure 1(c) and, unless specified otherwise, an arc p_ip_j refers to the arc from p_i to p_j on the arc $C_{0,3,4}$ or $C_{1,2,5}$ that supports both p_i and p_j . We first prove that the arcs p_2p_5 and p_3p_4 do not intersect. For any point q on \mathcal{P} between p_4 and p_5 , the arc p_3q on the circular arc through p_0, p_3, q lies above the concatenation of the arcs p_3p_4 of $C_{0,3,4}$ and p_4q of \mathcal{P} (since the circular arcs p_3q and p_3p_4 lie above \mathcal{P} , by Lemma 1, and $\mathcal{C}(p_0, p_3, p_4)$ and $\mathcal{C}(p_0, p_3, q)$ intersect only at p_0 and p_3). It follows that if arc p_3p_4 intersects arc p_2p_5 , then arc p_3q also intersects arc p_2p_5 for any position of q between p_4 and p_5 on \mathcal{P} . This implies that, for the limit case where $q = p_5$, arc $C_{1,2,5}$ and the circular arc through p_0, p_3 , and $q = p_5$ intersect in some point other than $q = p_5$, which is not the case by Lemma 2.

We now prove, similarly, that the arcs p_0p_3 and p_1p_2 do not intersect. For any point q on \mathcal{P} between p_0 and p_1 , the arc qp_2 on the circular arc through q, p_2, p_5 lies below the concatenation of the arcs qp_1 of \mathcal{P} and p_1p_2 of $C_{1,2,5}$. It follows that if arc p_1p_2 intersects arc p_0p_3 , then arc qp_2 also intersects arc p_0p_3 for any position of q between p_0 and p_1 on \mathcal{P} . This implies that, for the limit case where $q = p_0$, arc $C_{0,3,4}$ and the circular arc through $q = p_0, p_2$, and p_5 intersect in some point other than $q = p_0$, which is not the case by Lemma 3.

Finally, arcs p_1p_2 of $C_{1,2,5}$ and p_3p_4 of $C_{0,3,4}$ do not intersect because they lie on different sides of \mathcal{P} and similarly for arcs p_0p_3 of $C_{0,3,4}$ and p_2p_5 of $C_{1,2,5}$. Hence, the two arcs $C_{0,3,4}$ or $C_{1,2,5}$ do not intersect. \square

3 Universal Point Set for Circular Arc Drawings

In this section, we construct a set of n points on \mathcal{P} and, by using the lemmata of the previous section, we prove that it is universal for plane circular arc drawings of n -vertex planar graphs.

Consider n^2 points q_0, \dots, q_{n^2-1} on the parabolic arc \mathcal{P} such that $x_0 \geq 1$ and $x_i \geq 2x_{i-1}$ for $i = 1, \dots, n^2 - 1$. For our universal point set, we take the n

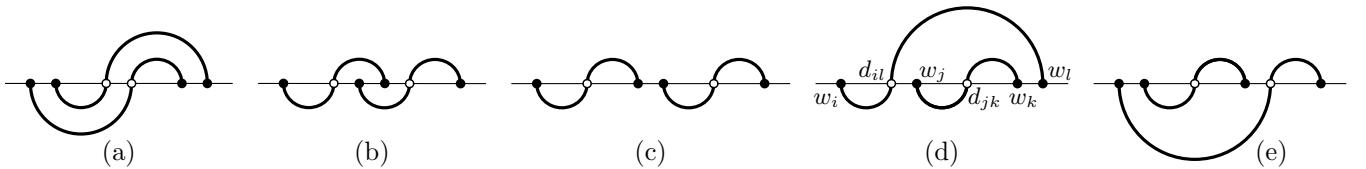


Figure 2: Relative positions of two edges in a monotone topological book embedding.

points $p_i = q_{ni}$ for $i = 0, \dots, n - 1$. We call the points in q_0, \dots, q_{n^2-1} that are not in the universal point set *helper points*.

Theorem 5 Every n -vertex planar graph can be drawn with the vertices on p_0, \dots, p_{n-1} and circular edges that do not intersect except at common endpoints.

Proof. Consider any planar graph G . Construct a monotone topological book embedding Γ of G in which all edges are drawn with a spine crossing [8, 16]. Denote by w_0, \dots, w_{n-1} the order of the vertices of G on the spine in Γ . We substitute every spine crossing with a *dummy* vertex. The relative position of any two edges in Γ is as depicted in Figure 2 (in which two edges may share their endpoints). For $0 \leq i \leq n-1$, we map vertex w_i to point p_i . Furthermore, for each $0 \leq i \leq n-2$, we map the dummy vertices that lie in between w_i and w_{i+1} on the spine in Γ to distinct helper points in between p_i and p_{i+1} , so that the order of the dummy vertices on \mathcal{P} is the same as on the spine in Γ . (We postpone the proof that there are enough points q_i to map the dummy vertices.) We finally draw every edge (w_i, w_j) of G containing a dummy vertex d_l as a circular arc passing through p_i , through p_j , and through the helper point to which vertex d_l has been mapped to. We prove that the resulting drawing is plane.

By Lemmata 2, 3, and 4, two edges whose relative positions in Γ are as depicted in Figure 2(a) do not intersect except possibly at a common endpoint.

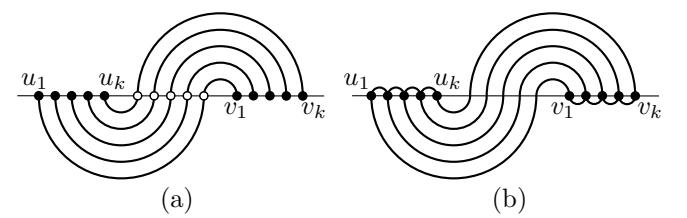
For the pairs of edges whose relative positions in Γ are as depicted in Figures 2(b) and 2(c), it is straightforward to check that they do not intersect either because they are separated by \mathcal{P} , or because they are y -monotone and hence they are separated by a horizontal line.

Consider two edges (w_i, w_l) and (w_j, w_k) whose relative position in Γ is as depicted in Figure 2(d) (the argument for pairs of edges as in Figure 2(e) is analogous). Let d_{il} and d_{jk} be the dummy vertices of (w_i, w_l) and (w_j, w_k) , respectively. Let q_{il} and q_{jk} be the points on \mathcal{P} to which d_{il} and d_{jk} are mapped. Arcs $p_i q_{il}$ and $p_j q_{jk}$ do not intersect because they are both y -monotone and their endpoints are separated by a horizontal line. Arcs $q_{il} p_l$ and $p_j q_{jk}$ do not intersect because they are separated by \mathcal{P} . Hence, it suffices to prove that arcs $q_{jk} p_k$ and $q_{il} p_l$ do not intersect. These two arcs are above and to the right of \mathcal{P} (by Lemma 1)

and q_{il}, q_{jk}, p_k, p_l are ordered from top to bottom. It is thus sufficient to prove that there exists a curve from q_{jk} to p_k that is to the right of $q_{jk} p_k$ and that does not intersect $q_{il} p_l$. Consider the (y -monotone) arc from q_{jk} to p_k of the circle $\mathcal{C}(p_i, q_{jk}, p_k)$. It is indeed to the right of the arc $q_{jk} p_k$ (of $\mathcal{C}(p_j, q_{jk}, p_k)$) because p_i is inside $\mathcal{C}(p_j, q_{jk}, p_k)$ (by Lemma 1) and p_i, q_{jk} , and p_k are ordered on the parabola. Furthermore, this new arc does not intersect $q_{il} p_l$ because in the case where $w_i = w_j$, w_k and w_l are in this order on the spine—that’s the situation depicted in Figure 2(a)—we know that the corresponding circular arcs do not intersect.

It remains to show that there are enough helper points to map the dummy vertices. There are $n - 1$ helper points $q_{ni+1}, \dots, q_{n(i+1)-1}$ between each pair of points $p_i = q_{ni}$ and $p_{i+1} = q_{n(i+1)}$. It thus suffices to prove that there are at most $n - 1$ dummy vertices in between w_i and w_{i+1} along the spine in Γ .

Let $(u_1, v_1), \dots, (u_k, v_k)$ be k edges in the book embedding that define consecutive dummy vertices on the spine. If no vertex w_i lies in between these dummy vertices on the spine in Γ , the k edges are such that $u_1, \dots, u_k, v_1, \dots, v_k$ are ordered from left to right on the spine in Γ ; see Figure 3(a). Now, consider the graph that consists of these edges plus the edges $(u_i, u_{i+1}), (v_i, v_{i+1})$, for $i = 1, \dots, k-1$; see Figure 3(b). This graph is outerplanar. It has at most n vertices and, thus, at most $n - 3$ chords. On the other hand, it has exactly $k - 2$ chords: $(u_2, v_2), \dots, (u_{k-1}, v_{k-1})$. This implies that $k - 2 \leq n - 3$ and $k \leq n - 1$, which concludes the proof. \square

Figure 3: (a) k edges of a monotone topological book embedding that defines k consecutive dummy vertices (spine crossings). (b) Augmented outerplanar graph.

4 Conclusions

We proved the existence of a universal point set with n points for plane circular arc drawings of planar graphs. The universal point set we constructed has an area of $2^{O(n^2)}$. It would be interesting, also for practical visualization purposes, to construct a universal point set with n points for plane circular arc drawings of planar graphs within polynomial area. We remark that (relaxing the requirement that the set have exactly n points) a universal point set with $O(n)$ points and within $2^{O(n)}$ area for plane circular arc drawings of planar graphs is $Q = \{q_0, \dots, q_{4n-7}\}$, where the helper points are defined as in Section 3. To construct a plane circular-arc drawing of a planar graph G on Q , it suffices to map vertices and dummy vertices of a monotone topological book embedding of G to the points of Q in the order they appear in the book embedding. The geometric lemmata of Section 2 ensure that the resulting drawing is plane.

Acknowledgments

This research started during Dagstuhl Seminar 13151 “Drawing Graphs and Maps with Curves” in April 2013. The authors thank the organizers and the participants for many useful discussions.

References

- [1] P. Angelini, G. D. Battista, M. Kaufmann, T. Mchedlidze, V. Roselli, and C. Squarcella. Small point sets for simply-nested planar graphs. In M. van Kreveld and B. Speckmann, editors, *Proc. 19th Int. Symp. Graph Drawing (GD’11)*, volume 7034 of *LNCS*, pages 75–85. Springer, 2012.
- [2] M. Bekos, M. Kaufmann, S. Kobourov, and A. Symvonis. Smooth orthogonal layouts. In Didimo and Patrignani [9], pages 150–161.
- [3] J. Cardinal and V. Kusters. On universal point sets for planar graphs. In *Proc. Thailand-Japan Joint Conf. Comput. Geom. Graphs (TJJCCGG’12)*, LNCS. Springer, 2013. To appear, see arXiv:1209.3594.
- [4] M. Chrobak and H. J. Karloff. A lower bound on the size of universal sets for planar graphs. *SIGACT News*, 20(4):83–86, 1989.
- [5] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
- [6] E. D. Demaine, J. S. B. Mitchell, and J. O’Rourke. The open problems project. Website, 2001. URL cs.smith.edu/~orourke/TOPP, accessed May 5, 2012.
- [7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.
- [8] E. Di Giacomo, W. Didimo, G. Liotta, and S. Wismath. Curve-constrained drawings of planar graphs. *Comput. Geom. Theory Appl.*, 30:1–23, 2005.
- [9] W. Didimo and M. Patrignani, editors. *Proc. 20th Int. Symp. Graph Drawing (GD’12)*, volume 7704 of *LNCS*. Springer, 2013.
- [10] D. Dolev, T. Leighton, and H. Trickey. Planar embedding of planar graphs. *Advances in Computing Research*, 2:147–161, 1984.
- [11] V. Dujmovic, W. S. Evans, S. Lazard, W. Lenhart, G. Liotta, D. Rappaport, and S. K. Wismath. On point-sets that support planar graphs. *Comput. Geom. Theory Appl.*, 46(1):29–50, 2013.
- [12] D. Eppstein. Planar Lombardi drawings for subcubic graphs. In Didimo and Patrignani [9], pages 126–137.
- [13] H. Everett, S. Lazard, G. Liotta, and S. Wismath. Universal sets of n points for one-bend drawings of planar graphs with n vertices. *Discrete Comput. Geom.*, 43(2):272–288, 2010.
- [14] R. Fulek and C. Tóth. Universal point sets for planar three-trees. In F. Dehne, J.-R. Sack, and R. Solis-Oba, editors, *Proc. 13th Int. Algorithms Data Struct. Symp. (WADS’13)*, volume 8037 of *LNCS*. Springer, 2013. To appear.
- [15] E. R. Gansner, S. C. North, and K.-P. Vo. DAG—a program that draws directed graphs. *Softw. Pract. Exper.*, 18(11):1047–1062, 1988.
- [16] F. Giordano, G. Liotta, T. Mchedlidze, and A. Symvonis. Computing upward topological book embeddings of upward planar digraphs. In T. Tokuyama, editor, *Proc. Int. Symp. Algorithms Comput. (ISAAC’07)*, volume 4835 of *LNCS*, pages 172–183. Springer, 2007.
- [17] M. Kurowski. A 1.235 lower bound on the number of points needed to draw all n -vertex planar graphs. *Inf. Process. Lett.*, 92(2):95–98, 2004.
- [18] D. Mondal. Embedding a planar graph on a given point set. Master’s thesis, Department of Computer Science, University of Manitoba, 2012. Available at www.cs.umanitoba.ca/~jyoti/DMthesis.pdf.
- [19] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.*, 1(1):343–353, 1986.
- [20] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Symp. Discrete Algorithms (SODA’90)*, pages 138–148, 1990.
- [21] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13(52):743–768, 1963.

Weighted Region Problem in Arrangement of Lines

Amin Gheibi^{*†1}, Anil Maheshwari^{†1}, and Jörg-Rüdiger Sack^{*†1}

¹School of Computer Science, Carleton University, Ottawa, ON, Canada
 [agheibi,anil,sack]@scs.carleton.ca

Abstract

In this paper, a geometric shortest path problem in weighted regions is discussed. An arrangement of lines \mathcal{A} , a source s , and a target t are given. The objective is to find a weighted shortest path, π_{st} , from s to t . Existing approximation algorithms for weighted shortest paths work within bounded regions (typically triangulated). To apply these algorithms to unbounded regions, such as arrangements of lines, there is a need to bound the regions. Here, we present a minimal region that contains π_{st} , called SP-Hull of \mathcal{A} . It is a closed polygonal region that only depends on the geometry of the arrangement \mathcal{A} and is independent of the weights. It is minimal in the sense that for any arrangement of lines \mathcal{A} , it is possible to assign weights to the faces of \mathcal{A} and choose s and t such that π_{st} is arbitrary close to the boundary of SP-Hull of \mathcal{A} . We show that SP-Hull can be constructed in $\mathcal{O}(n \log n)$ time, where n is the number of lines in the arrangement. As a direct consequence we obtain a shortest path algorithm for weighted arrangements of lines.

1 Introduction

The geometric shortest path problem ranks among the fundamental problems studied in Computational Geometry and related fields. In this problem, the input is a set of regions (often a triangulation), where each region (triangle) has a corresponding weight, and two points, source s and target t . The output is the weighted shortest path from s to t , π_{st} , which is the path with minimum cost. The cost of the path is the total sum of the length of each segment multiplied by the corresponding region's weight.

Mitchell and Papadimitriou [2] introduced this problem and proposed a $(1 + \varepsilon)$ -approximation algorithms. Subsequently, positioning Steiner points to discretize the triangulation became a common technique to obtain an approximation for the geometric shortest path

problem in weighted regions (cf. [3, 4]). The general idea of this technique is to place a set of Steiner points in each triangle and then build a graph by connecting them. The approximation solution is achieved by finding a shortest path inside this graph, by using well-known combinatorial algorithms (e.g., Dijkstra's, BUSHWHACK[4]). Some geometric factors (such as segment lengths, angles) are taken into account in the process of Steiner point placement. Therefore, the number of Steiner points depends on these geometric factors.

To the best of our knowledge, nobody has studied the weighted shortest path problem when the input is an arrangement of lines. It is impossible to cover the whole length of the lines with Steiner points, because lines are infinite and we cannot afford an infinite number of Steiner points. Therefore, in this context, the first challenge is to bound the number of Steiner points. Consequently, we need a bound on the region that weighted shortest paths, from s to t , lie on. After establishing this bound (i.e., a closed region) the infinite lines can be clipped to bounded length segments, and the faces of the arrangement inside that region can be triangulated. Finally, by using the algorithm in [3] a $(1 + \varepsilon)$ -approximation can be obtained.

The formal problem statement is as follows: let s and t be two points in the plane \mathbb{R}^2 and let \mathcal{A} be an arrangement of n lines l_i , $i = 1 \dots n$. For simplicity, assume no two lines in \mathcal{A} are parallel to each other and no three lines have a common intersection. Each face of \mathcal{A} is assigned positive weight w_i . By convention, the weight of each edge of \mathcal{A} is the minimum of the weights of its adjacent faces. The task is to find a closed region in \mathbb{R}^2 that contains a weighted shortest path from s to t , π_{st} .

A naive solution is a circle, centered at s whose radius is the Euclidean distance between s and t multiplied by $w_{max} = \max_i w_i$. It is easy to see that π_{st} will be inside this circle. This circle clips the lines to segments and the lengths of segments are bounded by the diameter of the circle. However, this bound is very sensitive to outliers and if w_{max} is large, then so is the size of the circle. In this paper, we propose an algorithm to construct a closed polygonal region, called SP-Hull (*Shortest Path Hull*), that only depends on the geometry of the arrangement and is independent of the weights. This al-

^{*}Research supported by High Performance Computing Virtual Laboratory and SUN Microsystems of Canada

[†]Research supported by Natural Sciences and Engineering Research Council of Canada

gorithm exploits the fact that in an arrangement of lines, the lines outside of the convex hull of \mathcal{A} diverge. Therefore, any shortest path, started and ended inside of the convex hull of \mathcal{A} , cannot go arbitrarily far from the convex hull (i.e., there is a bound). We show that there are some polygonal chains that define this bound for shortest paths, and they intersect in a restricted way. From this, we construct the SP-Hull. We will prove that any π_{st} lies inside the SP-Hull. We also justify that this is an optimally bounded region, one in which π_{st} is located in the absence of any assumptions on the weights.

The structure of this extended abstract is as follows. In Section 2, necessary preliminaries are presented. In Section 3, some relevant geometric properties are discussed. The construction algorithm for SP-Hull is described and analyzed in Section 4. Due to space limitation, some of the proofs of the lemmas are removed.

2 Preliminaries

Let \mathcal{A} be an arrangement of n lines l_i , $i = 1 \dots n$, and P be the set of intersection points of l_i , $P = \{p_1, p_2, \dots, p_{n(n-1)/2}\}$. The convex hull of P is denoted by $\mathcal{CH}(P) = \langle c_1, \dots, c_H \rangle$. Each line l_i either intersects $\mathcal{CH}(P)$ twice, at a_{i_1} and a_{i_2} , or contributes a segment to the boundary of the convex hull, $\partial\mathcal{CH}(P)$, from $a_{i_1} \in l_i$ to $a_{i_2} \in l_i$. For each l_i , $i = 1 \dots n$, we define two non-intersecting rays (subset of l_i) from a_{i_1} and a_{i_2} , respectively toward infinity. Sort all the rays based on their slopes, and arrange them in a counter-clockwise order around $\mathcal{CH}(P)$. This defines an order for the rays $R = \langle r_1, r_2, \dots, r_{2n} \rangle$ (Figure 1). This is a circular order and the relation “ $<$ ” is well-defined. Note that all the rays diverge and there is no intersection between any two of them in the exterior of $\mathcal{CH}(P)$.

For simplicity, it is assumed that s and t are inside (or on the boundary of) $\mathcal{CH}(P)$. If they are not, a set of at most three lines, passing through s and t , can be added to the arrangement. This ensures that s and t are not outside of $\mathcal{CH}(P)$. However, π_{st} does not necessarily lie inside $\mathcal{CH}(P)$. For example, in Figure 1, suppose the weight of the face f_i is “very large” and the weight of the face f_{i+1} is “very cheap”. Then, the shortest path from s to t goes outside of $\mathcal{CH}(P)$, as depicted in the figure.

In this paper, each ray is identified by a pair $r = \langle a, \vec{d} \rangle$, where a is the starting point on the boundary of $\mathcal{CH}(P)$ and \vec{d} is a vector pointing away from $\mathcal{CH}(P)$. W.l.o.g., it can be assumed for the remainder of the paper that the angle between any two consecutive rays, $r_1 = \langle a_1, \vec{d}_1 \rangle, r_2 = \langle a_2, \vec{d}_2 \rangle \in R$, is less than $\frac{\pi}{2}$. If it is not, (since this angle is less than π) one extra ray $r' = \langle a', \vec{d}_1 + \vec{d}_2 \rangle$ can be added in between, where a' is a point on the boundary of $\mathcal{CH}(P)$, between a_1 and a_2 . The total number of such angles greater than or equal to

$\frac{\pi}{2}$ in R is at most 4. Therefore, by adding a constant number of rays to R this assumption holds.

Definition 1 (Order of the points on a ray) For two points x and y on a ray $r_i = \langle a, \vec{d} \rangle$, $x < y$ if $|\vec{ax}| < |\vec{ay}|$, where $| \cdot |$ denotes the length of a vector.

Note that this is defined for points on a ray $r_i \subset l_j$. The point a is mapped to zero, and the points on the ray r_i are mapped to \mathbb{R}^+ , in the direction of \vec{d} .

Definition 2 (Chains: chain_i^{ccw} and chain_i^{cw})

Let c_i be a vertex of $\mathcal{CH}(P)$ corresponding to the intersection of rays r_{i-1} and r_i . The chain_i^{ccw} is a polygonal chain, starting from c_i , defined as follows. Find the normal from c_i to r_{i+1} . Let it be incident at the point h_{i+1} . Find the normal from h_{i+1} to r_{i+2} and repeat until, either the normal is incident on a vertex of $\mathcal{CH}(P)$ or is incident on a point in the interior of $\mathcal{CH}(P)$. Then, $\text{chain}_i^{ccw} = \langle c_i, h_{i+1}, \dots, h_j \rangle$, where $h_j \in \mathcal{CH}(P)$ (see Figure 1). The chain_i^{cw} is defined analogously.

The inner angle between two consecutive segments of chain_i^{ccw} is the angle on the left-hand side, when the direction is from c_i towards h_j . Analogously, for chain_i^{cw} , it is the angle on the right-hand side.

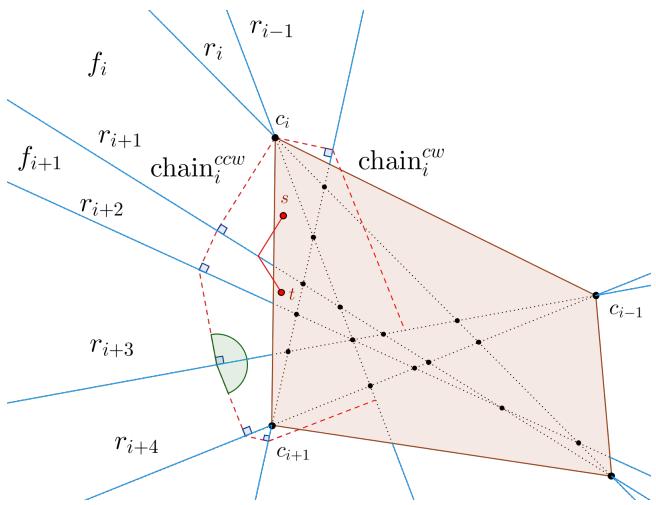


Figure 1: For each line in the arrangement there are two rays (in blue). Also, each vertex of $\mathcal{CH}(P)$, denoted by c_i , has two chains, chain_i^{ccw} and chain_i^{cw} (the red dashed lines in the figure). One of the inner angles of chain_i^{ccw} is shown in the figure (incident at r_{i+3}). Furthermore, suppose the weight of f_i is “very large” and the weight of f_{i+1} is “very cheap”. Then, π_{st} goes outside of $\mathcal{CH}(P)$.

3 Geometric Properties

In this section, some of the geometric properties on the order of the rays in the set R are discussed. Based on

these properties, some lemmas about the chains, which are the primitive elements for constructing SP-Hull are proven.

Property 1 Let $r_h < r_i < r_j$ be three rays in R and the angles between r_h and r_i , and, r_i and r_j , are both less than $\frac{\pi}{2}$. Let x (y) be a point on r_h (r_j).

- (a) The normal from x to r_j , lies on the left side of the normal from x to r_i , directed from x toward r_i . Analogously, the normal from y to r_h , lies on the right side of the normal from y to r_i , directed from y toward r_i (Figure 2 a).
- (b) The normals from x and y to r_i lie on the opposite sides of the straight line that connects x to y , \overline{xy} (or both coincide with \overline{xy}) (Figure 2 b).

Proof. The property follows from the fact that rays in R diverge and do not intersect in the exterior of $\mathcal{CH}(P)$. \square

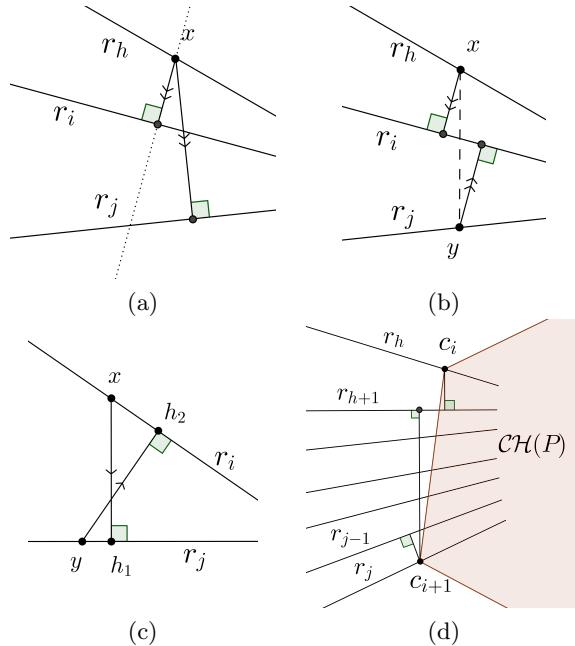


Figure 2: a) Property 1a, the normal from x to r_j lies on the left side of the normal from x to r_i . b) Property 1b, the normals from x and y to r_i lie on the opposite sides of \overline{xy} . c) Property 2b, if xh_1 intersects with yh_2 , then $h_2 < x$ and $h_1 < y$. d) Lemma 1, one of the normals, either from c_i to r_{i+1} or from c_{i+1} to r_{i+k} , lies outside of $\mathcal{CH}(P)$.

Lemma 1 Let $c_i \in r_h$ and $c_{i+1} \in r_j$ be two consecutive vertices of $\mathcal{CH}(P)$. (i) If $r_h < r_i < r_j$, then one of the normals from c_i or c_{i+1} to r_i lies outside of $\mathcal{CH}(P)$ (or on its boundary). (ii) One of the normals, either from c_i to r_{h+1} or from c_{i+1} to r_{j-1} , lies outside of $\mathcal{CH}(P)$ (or on it) (see Figure 2 d).

Proof. (i) There is an edge e of $\mathcal{CH}(P)$ which is connecting c_i and c_{i+1} . By Property 1b, normals from c_i and c_{i+1} lie on the different sides of e or they coincide. Thus, either one of the normals lies outside or both are on the boundary of $\mathcal{CH}(P)$. (ii) If the normal from c_i to r_{h+1} lies outside the lemma is proved. Otherwise, by first part of this lemma, the normal from c_{i+1} to r_{h+1} lies outside (or on) the $\mathcal{CH}(P)$. Therefore, by Property 1a, the normal from c_{i+1} to r_{j-1} lies outside (or on it). \square

Property 2 Let $r_i < r_j$ be two rays in R so that the angle between them is less than $\frac{\pi}{2}$.

- (a) Let $x < y$ be two points on r_i . If the normal from x (y) to r_j is at h_1 (h_2), then $h_1 < h_2$.
- (b) Let x and y be two points on r_i and r_j , respectively. If the normal from x (y) to r_j (r_i) is at h_1 (h_2), and xh_1 intersects with yh_2 , then $h_2 < x$ and $h_1 < y$ (Figure 2 c).

Proof. The proof of (a) follows directly from the fact that rays in R diverge. To prove (b) assume that the axes are rotated until r_i is horizontal. Therefore, $\overline{yh_2}$ is vertical. Since r_i and r_j diverge, if x is chosen s.t. $x < h_2$ then $h_1 < y$. It implies that there will be no intersection. Therefore, to obtain an intersection between $\overline{xh_1}$ and $\overline{yh_2}$, x should be chosen s.t. $h_2 < x$. By this selection for x the only possible choice to pick y is $h_1 < y$. \square

Lemma 2 (i) All inner angles of a chain, are less than π . (ii) Furthermore, let $\text{chain}_i^{\text{ccw}} = \langle c_i, h_{i+1}, \dots, h_{s-1}, h_s, h_{s+1}, \dots \rangle$ and $\text{chain}_j^{\text{cw}} = \langle c_j, h'_{j-1}, \dots, h'_{s+1}, h'_s, h'_{s-1}, \dots \rangle$ intersect between r_s and r_{s+1} (see Figure 3a). Then, the common tangent l_t of $\text{chain}_i^{\text{ccw}}$ and $\text{chain}_j^{\text{cw}}$ passes through $h_s \in \text{chain}_i^{\text{ccw}}$ and $h'_{s+1} \in \text{chain}_j^{\text{cw}}$.

Proof. (i) It follows directly from the fact that the rays diverge and chains are defined by the normals to the rays. (ii) We provide a proof by contradiction for one the cases, when l_t passes through h_{s-1} and h'_{s+1} . Other cases for other pairs of vertices are analogous. Since $\text{chain}_i^{\text{ccw}}$ and $\text{chain}_j^{\text{cw}}$ are intersecting, both lie on the same side of l_t . Therefore, the normal from h_{s-1} to r_s and from h'_{s+1} to r_s both lie on the same side of l_t . This contradicts Property 1b. \square

Definition 3 (Complete revolution) Suppose $R = \langle r_1, \dots, r_{2n} \rangle$ is the counter-clockwise order of the rays and $\text{chain}_i^{\text{ccw}}$ ($\text{chain}_i^{\text{cw}}$) is initiated at $c_i \in \mathcal{CH}(P)$ where $c_i \in r_j$. A $\text{chain}_i^{\text{ccw}}$ ($\text{chain}_i^{\text{cw}}$), initiated at a point $x \in r_j$, is said to achieve a complete revolution, if it successively traverse all the rays in (reverse) order and returns back to r_j at a point x' such that x' is equal to x or $x < x'$.

Lemma 3 No chain starting at a vertex $c_i \in \mathcal{CH}(P)$ achieves a complete revolution.

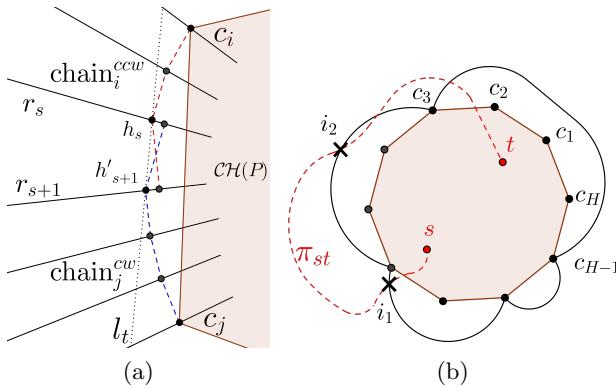


Figure 3: a) Two chains, $\text{chain}_i^{\text{ccw}}$ (the red dashed chain) and $\text{chain}_j^{\text{cw}}$ (the blue dashed chain), and their common tangent, l_t . b) An example of topological structure of the SP-Hull is shown in black solid lines. The red dashed line is the assumed weighted shortest path between s and t .

The proof of this lemma is based on the following observation. Always there exists a circle c_{\max} , passing through c_i with the center inside $\mathcal{CH}(P)$, such that encloses $\mathcal{CH}(P)$. We can prove that the initiated chain at this vertex lies inside c_{\max} . It implies that this chain does not achieve a complete revolution.

Lemma 4 Let $\text{chain}_i^{\text{ccw}} = \langle c_i, h_{i+1}, \dots, h_{s-1}, h_s, h_{s+1}, \dots \rangle$ and $\text{chain}_j^{\text{cw}} = \langle c_j, h'_{j-1}, \dots, h'_{s+1}, h'_s, h'_{s-1}, \dots \rangle$ intersect between r_s and r_{s+1} (Figure 3a). Then, $\text{chain}_{ij} = \langle c_i, h_{i+1}, \dots, h_{s-1}, h_s, h'_{s+1}, h'_{s+2}, \dots, h'_{j-1}, c_j \rangle$ is a polygonal chain, connecting c_i to c_j and the inner angles of chain_{ij} are less than π .

Proof. By Lemma 2, chain_{ij} from c_i to h_s and from h'_{s+1} to c_j is convex. Therefore, it is enough to show that $\angle h_{s-1}h_sh'_{s+1}$ and $\angle h_sh'_{s+1}h'_{s+2}$ are less than π . In Lemma 2 we showed that the common tangent of $\text{chain}_i^{\text{ccw}}$ and $\text{chain}_j^{\text{cw}}$, l_t , passes through h_s and h'_{s+1} . Since l_t is a straight line and both chains lie on the same side of l_t , $\angle h_{s-1}h_sh'_{s+1}$ and $\angle h_sh'_{s+1}h'_{s+2}$ are less than π . \square

Let $\text{CW} = \{\text{chain}_i^{\text{cw}} \mid i = 1..H\}$ and $\text{CCW} = \{\text{chain}_i^{\text{ccw}} \mid i = 1..H\}$.

Lemma 5 Every $r_i \in R$ intersects with at least one of $\text{chain}_j^{\text{ccw}} \in \text{CCW}$ or $\text{chain}_{j+1}^{\text{cw}} \in \text{CW}$.

Proof. Every $r_i \in R$ is between two consecutive vertices of $\mathcal{CH}(P)$, c_j and c_{j+1} . By Lemma 1, one of the normals from c_j and c_{j+1} to r_i is not inside $\mathcal{CH}(P)$. W.l.o.g. assume that the normal from c_j to r_i is not inside. By Property 1a, $\text{chain}_j^{\text{ccw}}$ lies on the left side (or on) the normal from c_j to r_i . Therefore, $\text{chain}_j^{\text{ccw}} \in \text{CCW}$ intersects r_i . \square

Lemma 6 Any two chains in CW (or CCW) are either disjoint or share an end-point at a vertex of $\mathcal{CH}(P)$.

Proof. This proof uses contradiction. Suppose two chains, $\text{chain}_i^{\text{cw}}$ and $\text{chain}_j^{\text{cw}}$, intersect between two rays, r_s and r_{s+1} , not at a vertex of $\mathcal{CH}(P)$. Suppose $\text{chain}_i^{\text{cw}}$ intersects r_s at x and r_{s+1} at h . Also, $\text{chain}_j^{\text{cw}}$ intersects r_s at y and r_{s+1} at h' . W.l.o.g. assume $x < y$. If they intersect, it implies $h' < h$. This contradicts Property 2a. \square

Definition 4 (Maximal chain) Suppose $\text{chain}_i^{\text{ccw}}$ starts at r_j and ends at r_{j+k} , that is, $\text{chain}_i^{\text{ccw}}$ covers rays from r_j to r_{j+k-1} . We represent $\text{chain}_i^{\text{ccw}}$ by a range $[j, \dots, j+k-1]$. It is a subrange¹ of a circular range of integers $[1, \dots, 2n]$. We say $\text{chain}_i^{\text{ccw}}$ is maximal if there is no $\text{chain}_x^{\text{ccw}} \in \text{CCW}$ or $\text{chain}_x^{\text{cw}} \in \text{CW}$ such that its representative range fully covers the range $[j, \dots, j+k-1]$. Analogously, the maximal $\text{chain}_i^{\text{cw}}$ is defined.

Let $\text{CCW}_{\max} = \{\text{chain}_i^{\text{ccw}} \mid i = 1..H, \text{ s.t. } \text{chain}_i^{\text{ccw}}$ is maximal} and $\text{CW}_{\max} = \{\text{chain}_i^{\text{cw}} \mid i = 1..H, \text{ s.t. } \text{chain}_i^{\text{cw}}$ is maximal}. By Lemma 6, CCW_{\max} (CW_{\max}) is a set of chains such that their representative ranges are disjoint.

Lemma 7 Suppose $\text{chain}_i^{\text{ccw}} \in \text{CCW}_{\max}$ and it covers the starting point of $\text{chain}_x^{\text{cw}} \in \text{CW}_{\max}$. Then they do not intersect.

Proof. By definition, $\text{chain}_i^{\text{ccw}}$ starts at the boundary of $\mathcal{CH}(P)$ and ends inside. Therefore $\text{chain}_i^{\text{ccw}}$ forms a closed region with the boundary of $\mathcal{CH}(P)$. By the assumption of the lemma, $\text{chain}_x^{\text{cw}}$ starts from inside the corresponding region of $\text{chain}_i^{\text{ccw}}$. If these two chains intersect, the intersection contradicts Property 2b. \square

Corollary 1 Let $\text{chain}_i^{\text{ccw}}, \text{chain}_j^{\text{ccw}} \in \text{CCW}_{\max}$ ($\text{chain}_i^{\text{cw}}, \text{chain}_j^{\text{cw}} \in \text{CW}_{\max}$) be two disjoint chains. There is no $\text{chain}_x^{\text{cw}} \in \text{CW}_{\max}$ ($\text{chain}_x^{\text{ccw}} \in \text{CCW}_{\max}$) that intersects both of them.

Proof. If $\text{chain}_x^{\text{cw}}$ intersects $\text{chain}_i^{\text{ccw}}$ and $\text{chain}_j^{\text{ccw}}$ without intersecting $\mathcal{CH}(P)$, then by Lemma 7 it must intersects one of them at least twice. W.l.o.g. assume that $\text{chain}_x^{\text{cw}}$ intersects $\text{chain}_i^{\text{ccw}}$ twice, once to enter the closed region formed by $\text{chain}_i^{\text{ccw}}$ and once to leave it. The second intersection contradicts Property 2b. \square

Lemma 8 Each $\text{chain}_i^{\text{ccw}} \in \text{CCW}_{\max}$ intersects exactly one $\text{chain}_j^{\text{cw}} \in \text{CW}_{\max}$, or it ends at a vertex $c_x \in \mathcal{CH}(P)$.

¹For simplicity, we are omitting "modulo" as this is a circular range.

Proof. By definition, $\text{chain}_i^{ccw} \in \text{CCW}_{max}$ starts at a vertex of $\mathcal{CH}(P)$. We prove that if it does not end at another vertex of $\mathcal{CH}(P)$, then it intersects exactly one $\text{chain}_j^{cw} \in \text{CW}_{max}$.

Suppose r_s is the ray that chain_i^{ccw} ends on. Thus, the intersection of chain_i^{ccw} and r_s is in the interior of $\mathcal{CH}(P)$. By Lemma 5, there exists another chain, chain_x , that intersects r_s outside (or on) $\mathcal{CH}(P)$. This chain_x cannot be a member of CCW_{max} because it either intersects with chain_i^{ccw} (which contradicts Lemma 6) or fully covers chain_i^{ccw} (which contradicts maximality). Therefore, it is a member of CW and intersects chain_i^{ccw} . If it is maximal, we have proved that there exist at least one chain in CW_{max} that intersects. If it is not maximal, then there exists a maximal chain, chain_y , that fully covers chain_x . By the same reasoning, chain_y cannot be a member of CCW_{max} . Therefore, chain_y is a member of CW_{max} and intersects chain_i^{ccw} (if it does not intersect, it should fully cover chain_i^{ccw} which contradicts maximality of chain_i^{ccw}).

Now suppose there are two chains chain_x^{cw} and $\text{chain}_y^{cw} \in \text{CW}_{max}$, that intersect chain_i^{ccw} . By Corollary 1, chain_x^{cw} and chain_y^{cw} should either intersect each other (which contradicts Lemma 6) or one should fully cover the other one (which contradicts maximality). Therefore, there exists exactly one $\text{chain}_j^{cw} \in \text{CW}_{max}$ that intersects chain_i^{ccw} . \square

4 The construction algorithm

In this section, we present an algorithm to construct the SP-Hull (Algorithm 1). The input is an arrangement of lines \mathcal{A} , a source s , and a target t . The assumption is that s and t are inside $\mathcal{CH}(P)$. The output is a simple closed polygonal region SP-Hull that encloses $\mathcal{CH}(P)$. The idea to construct SP-Hull is to cover all vertices of $\mathcal{CH}(P)$ by some polygonal chains, chain_{ij} , which lie outside of $\mathcal{CH}(P)$ (see Figure 3b). We will prove that any weighted shortest path from s to t lies inside SP-Hull. Furthermore, we will argue its minimality.

Theorem 9 *Let \mathcal{A} be an arrangement of lines. Any weighted shortest path between two points inside $\mathcal{CH}(P)$, lies inside SP-Hull of \mathcal{A} , constructed by Algorithm 1.*

Proof. This proof has two main steps. First, we prove that SP-Hull, generated by Algorithm 1, is a simple polygon that encloses $\mathcal{CH}(P)$. In the second step we prove, by contradiction, that any weighted shortest path between s and t , π_{st} , does not go outside of SP-Hull, where $s, t \in \mathcal{CH}(P)$.

Based on the construction in Algorithm 1, SP-Hull is a sequence of chains, chain_{ij} , which do not overlap and cover all of the rays (Lemma 5). Figure 3b shows an example of topological structure of SP-Hull around $\mathcal{CH}(P)$. By Lemma 4, each chain_{ij} is a simple chain in

Algorithm 1 SP-Hull

Input: Source (s), target (t), an arrangement of n lines (\mathcal{A})

Output: A simple closed polygon, SP-Hull

```

1: Compute convex_hull( $\mathcal{A}$ ),  $CH = \{c_1, \dots, c_H\}$ ;
2: Mark all  $c_i \in CH$  as not covered;
3: Find  $\text{CCW}_{max}$  and  $\text{CW}_{max}$  sets and sort them
   based on chains' subscripts;
4: while all  $c_i$ 's are not covered do
5:    $\text{chain}_i^{ccw} = \text{First element of } \text{CCW}_{max}$ ;
6:   if  $\text{chain}_i^{ccw}$  intersects with  $\text{chain}_k^{cw}$  then
7:      $\text{chain}_{ik} = \text{Merge}(\text{chain}_i^{ccw}, \text{chain}_k^{cw})$ ;
8:     Mark all  $c_j$  ( $j = i..k$ ) as covered;
9:   else /* $\text{chain}_i^{ccw}$  ends at  $c_x \in CH*/
10:     $\text{chain}_{ix} = \text{chain}_i^{ccw}$ ;
11:    Mark all  $c_j$  ( $j = i..x$ ) as covered;
12: return the list of  $\text{chain}_{ij}$ , sorted by their first index
   (i.e.,  $i$ ), as the SP-Hull;$ 
```

which its inner angles are less than π . It starts and ends at the vertices of $\mathcal{CH}(P)$. Therefore, the SP-Hull is a closed simple polygon. Also, each chain_{ij} by definition is outside of $\mathcal{CH}(P)$. Therefore, SP-Hull encloses $\mathcal{CH}(P)$.

Before continuing the proof, let us introduce some notation. If π_x is a polygonal chain and a and b are two points on π_x , then $\pi_x[a, b]$ denotes the subpath of π_x from a to b .

In the second step of the proof, we show that no point of π_{st} lies in the exterior of SP-Hull. We prove this by contradiction. Since s and t are inside $\mathcal{CH}(P)$, π_{st} intersects SP-Hull at least twice. Let i_1 and i_2 be the first two consecutive intersections of π_{st} and SP-Hull (see Figure 3b). Our claim is $\text{SP-Hull}[i_1, i_2]$ is shorter than $\pi_{st}[i_1, i_2]$ which is a contradiction to the fact that π_{st} is a shortest path.

Suppose there are k regions between i_1 and i_2 which are separated by $k - 1$ rays. W.l.o.g., let the rays in order be $\langle r_1, \dots, r_{k-1} \rangle$. The number of segments in $\text{SP-Hull}[i_1, i_2]$ is at most k . Furthermore, the number of segments in $\pi_{st}[i_1, i_2]$ is at least k , as it must traverse through k diverging regions. We will show that each segment of $\text{SP-Hull}[i_1, i_2]$, o_j , is shorter than the corresponding segment of $\pi_{st}[i_1, i_2]$ in that region, π_j . Then, the total length of $\text{SP-Hull}[i_1, i_2]$ is smaller than the total length of $\pi_{st}[i_1, i_2]$ and we will arrive at a contradiction.

From the fact that there is no intersection between $\text{SP-Hull}[i_1, i_2]$ and $\pi_{st}[i_1, i_2]$ from i_1 to i_2 , o_j and π_j do not intersect. There are two cases: the segment o_j is one of the normals in a chain that is contributing to SP-Hull, or it is a segment introduced by merging of two chains. The first case is shown in Figure 4a. In this case, even if π_j is perpendicular, o_j is shorter because

the rays are diverging.

For case 2, assume that the endpoints of o_j are q_1 and q_2 , and the endpoints of π_j are q'_1 and q'_2 (see Figure 4b). Since r_j and r_{j+1} diverge, translating π_j toward $\mathcal{CH}(P)$ makes it shorter. Therefore, the shortest possible length for π_j while avoiding an intersection between o_j and π_j , is when one of the endpoints of π_j is as close as possible to one of the endpoints of o_j . Assume q'_1 is equal to q_1 . Then, o_j is shorter than π_j because of the following observation. The distance function from a point x to a ray r is a convex function (i.e., there is one line segment that connects x to a point $x_{opt} \in r$ such that it has the minimum length). Assume a line segment from x to $x_0 \in r$. By translating x_0 on r toward x_{opt} its length becomes shorter. \square

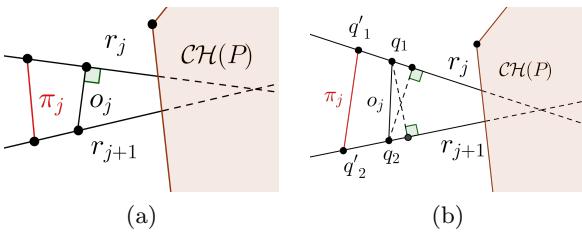


Figure 4: a) Proof of Theorem 9, case 1. b) Proof of Theorem 9, case 2.

Theorem 10 *For an arrangement of n lines, SP-Hull can be computed in $\mathcal{O}(n \log n)$ time.*

Proof. Computing the convex hull of P takes $\mathcal{O}(n \log n)$ time and its size is $\mathcal{O}(n)$ [1].

The key here is that it is possible to find CCW_{max} (CW_{max}) in linear time without computing all chain_i^{ccw} (chain_i^{cw}), $i = 1 \dots H$. Lemma 6 implies that if $c_j \in \mathcal{CH}$ is covered by a chain_i^{ccw} then we can skip computing chain_j^{ccw} and chain_j^{cw} , because they are not maximal. Also, members of CCW_{max} do not overlap. Therefore, the computation of CCW_{max} requires at most two traversals of the rays.

In the While-loop, CCW_{max} (CW_{max}) is a set of non-overlapping ranges that are sorted. Based on Lemma 8, each member of CCW_{max} , either has exactly one intersection with a member of CW_{max} , or both endpoints of that chain are vertices of \mathcal{CH} . Therefore, finding the intersecting chains takes constant time, by comparing only the endpoints of the first and the last chains in the sets. When an intersection is detected, then remove both chains from the sets, merge them and repeat. Since the total number of operations for merging all intersected chains is equal to the number of rays, the While-loop takes linear time. \square

Minimality of SP-Hull

In Theorem 9, we have shown that π_{st} lies inside SP-Hull. Now we address its minimality. We show that

for any arrangement of lines, \mathcal{A} , it is possible to assign weights to the faces of \mathcal{A} and choose $s, t \in \mathcal{CH}(P)$ such that π_{st} is arbitrarily close to the boundary of SP-Hull.

The procedure is as follows. Assign the weight “infinity” to the bounded faces of \mathcal{A} . By this assignment, we make sure that π_{st} does not traverse these faces. Choose one of the chains in SP-Hull, say chain_{ij} . This chain is either chain_i^{ccw} , or chain_j^{cw} , or the result of merging them. Here, we prove the minimality for the merging case. The other cases are analogous.

Let chain_{ij} be the result of merging chain_i^{ccw} and chain_j^{cw} . W.l.o.g., assume that chain_i^{ccw} starts at $c_i \in \mathcal{CH}(P)$ and intersects $\mathcal{CH}(P)$ at point $x \in \partial\mathcal{CH}(P)$. Place s on c_i and t on x . Assume chain_i^{ccw} traverses k unbounded faces in order, $\{f_1, \dots, f_k\}$. The weight for the other unbounded faces that are not visited by this chain, is set to infinity. To make π_{st} close enough to chain_i^{ccw} , the corresponding weights for f_i , $i = 1 \dots k$, are set in such a way that $w_1 \gg w_2 \gg \dots \gg w_k$. It suffices to set the weights of f_i , $i = 1 \dots k$, as z^i . If z goes to zero, then $w_i \gg w_{i+1}$ and π_{st} is arbitrarily close to chain_i^{ccw} . An analogous argument can be used to become as close as possible to chain_j^{cw} .

5 Further Work

Analogous question arises for existence of such bounded region for an arrangement of line segments in an appropriately defined weighted region problem. Suppose P is the set of endpoints of line segments and their intersections. It is not difficult to show that if s and t are inside the $\mathcal{CH}(P)$, then π_{st} will not go further than the boundary of $\mathcal{CH}(P)$. Also, an interesting extension of this problem is the question of existence of such bound for a given arrangement of curves (e.g., algebraic curves).

6 Acknowledgement

The authors would like to thank the anonymous reviewers of CCCG 2013 and Maryam Nasirpour for constructive comments.

References

- [1] M. Atallah. *Computing the convex hull of line intersections*. J. Algorithms 7, 285-288, 1986.
- [2] J.S.B. Mitchell and C.H. Papadimitriou. *The weighted region problem: finding shortest paths through a weighted planar subdivision*. J. ACM 38, 1:18-73, 1991.
- [3] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. *Determining approximate shortest paths on weighted polyhedral surfaces*. J. ACM 52, 1:25-53, 2005.
- [4] Z. Sun, J.H. Reif. *On finding approximate optimal paths in weighted regions*. J. Algorithms 58, 132, 2006.

Combinatorics of Beacon-Based Routing and Coverage

Michael Biro*

Jie Gao†

Justin Iwerks*

Irina Kostitsyna†

Joseph S. B. Mitchell*

Abstract

We consider combinatorial problems motivated by sensor networks for *beacon-based* point-to-point routing and covering. A beacon b is a point that can be *activated* to effect a ‘magnetic pull’ toward itself everywhere in a polygonal domain P . The routing problem asks how many beacons are required to route between any pair of points in a polygonal domain P . In simple polygons with n vertices we show that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary and always sufficient. In polygons with h holes, we establish that $\lfloor \frac{n}{2} \rfloor - h - 1$ beacons are sometimes necessary while $\lfloor \frac{n}{2} \rfloor + h - 1$ beacons are always sufficient. Loose bounds for simple orthogonal polygons are also shown. We consider art gallery problems where beacons function as guards. Loose bounds are given for covering simple polygons, polygons with holes and simple orthogonal polygons.

1 Introduction

The model of beacon-based routing in this paper is an analog of geographical greedy routing in sensor networks in the continuous setting. A comparison of our model with others found in the literature can be found in the current authors’ related paper on beacon-based algorithms [1]. We consider two main questions in this paper: “How many beacons are required to allow for point-to-point routing between any two points in a polygonal domain P ?” and “How many beacons are required to cover P ?”

In our model, a beacon can occupy a point location on the interior or the boundary of P , ∂P . When a beacon is *activated*, an object p in P moves along a straight line toward b until either it reaches b or makes contact with ∂P . If contact is made with ∂P , p will follow along ∂P as long as its straight line distance to b decreases monotonically. p may alternate between moving in a straight line path toward b on the interior of P and following along ∂P . If p is unable to move so that its distance to b decreases monotonically, we say p is ‘stuck’ and has reached a local minimum at a *dead point*. If an object p originating at a point q reaches b we say that b

attracts q . A point s is *routed* to t if there is a sequence of beacons that can be activated and then deactivated, one at a time in order, such that an object beginning at a source s visits each beacon in the sequence after it is activated and terminates at a destination t , which we always assume to be a beacon itself, but is not counted. We restrict each beacon to be activated at most one time during a routing. We say that a polygon P is covered by a set of beacons if every point of P is attracted by at least one beacon in the set.

2 Routing in Simple Polygons

Suppose first that P is a simple polygon. Then we show tight bounds on the number of beacons necessary to route between a pair s, t of points in P and the number of beacons sufficient to route between *any* pair of points s, t in P .

Theorem 1 *Given a simple polygon P , $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary and always sufficient to route between any pair of points in P .*

Proof. We can see from Figure 1 that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary to route between a specific pair s and t .

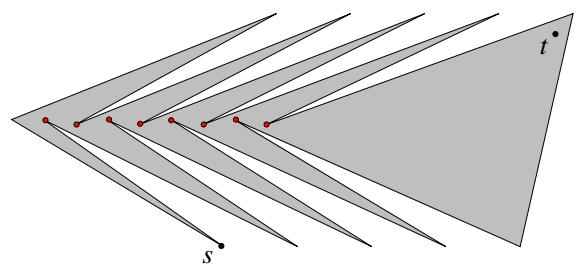


Figure 1: $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary to route between a pair of points in a simple polygon. Here, $n = 19$ and 8 beacons are required to route from s to t .

To establish the upper bound, we first triangulate P and construct the dual graph G of the resulting triangulation, rooted at an arbitrary triangle. Beginning with a lowest leaf node of G , we begin to peel off adjacent triangles. Suppose the leaf triangle is σ_1 and its neighbor is σ_2 . The analysis depends on the degree of σ_2 in the triangulation.

*Department of Applied Mathematics and Statistics, Stony Brook University, mbiro@ams.stonybrook.edu, jiwerks@ams.stonybrook.edu, jsbm@ams.stonybrook.edu

†Department of Computer Science, Stony Brook University, jgao@cs.stonybrook.edu, ikost@cs.stonybrook.edu

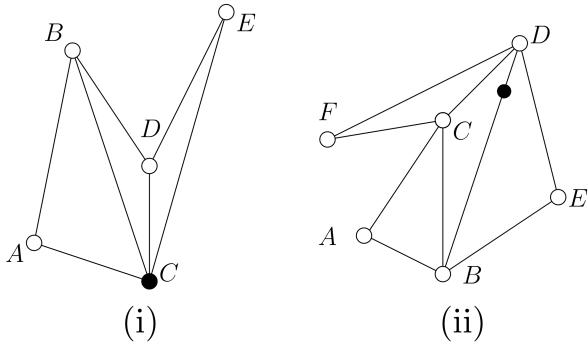


Figure 2: (i) The beacon b is placed at a vertex common to three triangles; (ii) The beacon b is placed appropriately on the edge BD . Any point in the four triangles can then navigate to b and vice-versa without any additional beacons needed.

- (i) σ_2 has only one additional adjacent triangle σ_3 . Suppose $\sigma_1 = \triangle ABC$, $\sigma_2 = \triangle BCD$. σ_3 is then either $\triangle BDE$ or $\triangle CDE$. If $\sigma_3 = \triangle CDE$, then we place a beacon b at the vertex C and otherwise we place b at B . In either case, since b is contained in each of the three triangles, any point p in these three triangles can navigate to b and vice-versa (see Figure 2 (i)).
- (ii) σ_2 has two additional adjacent triangles σ_3 , σ_4 . Assume that $\sigma_1 = \triangle ABC$, $\sigma_2 = \triangle BCD$, $\sigma_3 = \triangle BDE$, $\sigma_4 = \triangle CDF$. Suppose that the path from σ_1 to the root passes through triangle σ_3 . Since $\sigma_1 = \triangle ABC$ was a lowest leaf, $\sigma_4 = CDF$ is also a leaf. We place a beacon on the diagonal BD . The location b along BD is chosen so the pentagon $ABDFC$ is visible to b . This is always possible, by placing b on the correct side of lines CF and AC . Then, any point in triangles $\triangle ABC, \triangle BCD, \triangle CDF$ can be routed to or from b as b is visible to each point in those triangles. Hence, the claim is true (see Figure 2 (ii)).

Given the basic steps as shown above, we will place beacons in a recursive manner: We take any lowest leaf triangle σ_1 of the triangulation of P and place a beacon at points described above.

1. If P is a single triangle, we do nothing. If P has at most one more triangle besides σ_1 and σ_2 or P has only two more triangles but both are adjacent to σ_2 , then we are done after placing one more beacon (see Figure 3 (ii) or (iii)). By the above arguments we can navigate from any starting point to any destination point by using the single beacon: First route from the start to the beacon and then

route from the beacon to the destination (which is always a beacon).

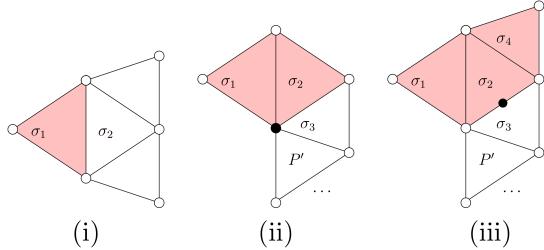


Figure 3: Inductive placement of beacons. (i): Base case; (ii): Peeling off σ_1 and σ_2 leaves a simple polygon; (iii): Peeling off σ_1 , σ_2 , and σ_4 , leaves a simple polygon.

2. Otherwise, we peel off σ_1 at least one more triangle. There are two subcases to consider:

- (a) σ_2 is only adjacent to one more triangle σ_3 (i.e., σ_2 has degree 2 in the dual graph; see Figure 3 (ii)). In this case peeling off σ_1 and σ_2 will still leave a simple polygon P' . We can recursively ‘beaconize’ P' . Now we argue that one can navigate with the union of these beacons. In particular, if the start and destination pair are both in $\sigma_1 \cup \sigma_2$, we may route from s to b and from b to t as b is visible to both s and t . If both s and t are in P' , then we can navigate by induction hypothesis. If the start and destination pair are separated in $\sigma_1 \cup \sigma_2$ and P' , we can route from s to the beacon b and then from b to t by the induction hypothesis and the analysis above. Thus navigation works in this case.
- (b) σ_2 is adjacent to two other triangles σ_3 and σ_4 , with σ_4 also a leaf. Thus peeling off σ_1 , σ_2 , and σ_4 will still leave a simple polygon P' ; see Figure 3 (iii)). We can recursively ‘beaconize’ P' . Now we argue that one can navigate with the union of these beacons. In particular, if the start and destination pair are both in $\sigma_1 \cup \sigma_2 \cup \sigma_4$, we may route from s to b and then from b to t as t is visible to both s and t . If both s and t are in P' , we can navigate by the induction hypothesis. If the start and destination pair are separated, we route from s to b and then from b to t , again by the induction hypothesis and the analysis above. Thus, navigation works in this case as well.

With the algorithm, we can see that each time we place a beacon we peel off at least two triangles. There

are $n - 2$ triangles in any triangulation of a simple polygon with n vertices. Thus the total number of beacons we place would be at most $\lfloor \frac{n-2}{2} \rfloor = \lfloor \frac{n}{2} \rfloor - 1$. \square

3 Routing in Polygons with Holes

If P has n vertices and h holes, then we give bounds on the number of beacons that are sometimes necessary and always sufficient to route between any pair of points in P .

Theorem 2 *Given a polygon P with n vertices and h holes, $\lfloor \frac{n}{2} \rfloor - h - 1$ beacons are sometimes necessary to route between a pair of points in P . Conversely, $\lfloor \frac{n}{2} \rfloor + h - 1$ beacons are always sufficient to route between any pair of points in P .*

Proof. Figure 5 illustrates that $\lfloor \frac{n}{2} \rfloor - h - 1$ beacons are sometimes necessary to route between a specific pair of points s and t . The figure shows the polygon from Figure 1, with another copy of that polygon placed where s was originally in Figure 1, as in Figure 4. The original polygon requires 19 vertices and the additional hole polygon has 19 vertices, plus one extra to close the hole, so 20 vertices, and 39 total. We have that 8 beacons are required for the original, and an additional 9 for the hole, so 17 beacons are required to route from s to t in this polygon. That is $\lfloor \frac{n}{2} \rfloor - h - 1 = \lfloor \frac{39}{2} \rfloor - 1 - 1 = 19 - 2 = 17$. This process may be iteratively repeated to achieve the $\lfloor \frac{n}{2} \rfloor - h - 1$ bound for large numbers of holes and vertices.

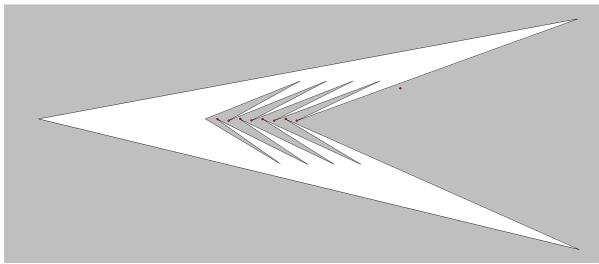


Figure 4: Closing a copy of the polygon in Figure 1 with an additional vertex to create a hole with 20 vertices that requires 9 beacons.

To establish the upper bound, we first triangulate P and construct the dual graph G of the resulting triangulation. Since P is not simple, there may be cycles in the dual graph and so, for each cycle in G we remove an edge. This leaves the dual graph connected and is equivalent to cutting a thin channel in the polygon to remove a hole, thus adding 2 vertices. After h cuts, the resulting polygon is simple and has $n + 2h$ vertices. We then utilize Theorem 1, to say that the resulting polygon may be routed with $\lfloor \frac{n+2h}{2} \rfloor - 1 = \lfloor \frac{n}{2} \rfloor + h - 1$ beacons, as the beacons placed do not depend on the rigidity of the

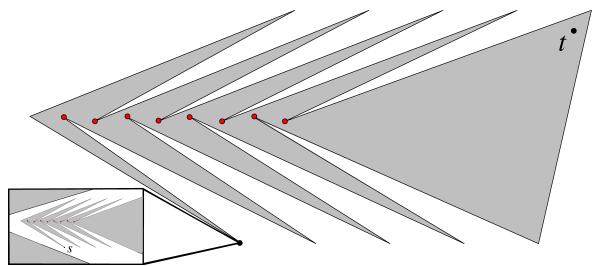


Figure 5: $\lfloor \frac{n}{2} \rfloor - h - 1$ beacons are sometimes necessary to route between a pair of points. Here, $n = 39$, $h = 1$, and 17 beacons are required to route from s to t .

edges of the polygon. Then, a valid routing sequence in the modified simple polygon corresponds to a valid routing sequence in the original polygon P . \square

Conjecture 1 *Given a polygon P with n vertices and h holes, $\lfloor \frac{n}{2} \rfloor - h - 1$ beacons are always sufficient to route between any pair of points in P .*

4 Routing in Simple Orthogonal Polygons

In simple orthogonal polygons we give loose bounds for the number of beacons necessary to route between a pair of points s, t in P and the number of beacons sufficient to route between *any* two points s, t in P . We were unsuccessful in attempting to mimic the proof of Theorem 1 with a peeling process on a convex quadrilateralization of P in order to improve the upper bound.

Theorem 3 *Given a simple orthogonal polygon P with n vertices, $\lfloor \frac{n}{4} \rfloor - 1$ beacons are sometimes necessary while $\lfloor \frac{n}{2} \rfloor - 1$ beacons always sufficient to route between any pair of points in P .*

Proof. We can see that $\lfloor \frac{n}{4} \rfloor - 1$ beacons are sometimes necessary to route between any pair of points in P by constructing an orthogonal ‘zig-zag’ polygon as in Figure 6. The upper bound carries over from Theorem 1 for general simple polygons. \square

5 Coverage in Simple Polygons and Polygons with Holes

Define the *attraction region* of a beacon b to be the locus of points in P that can reach b when b is activated. Also, let the *inverse attraction region* of point p be the locus of points that can attract p . Then we say that a set of beacons B covers a polygon P if P is entirely contained in the union of the attraction regions of the beacons in B . In this section, we give bounds on the number of beacons that are sometimes necessary and

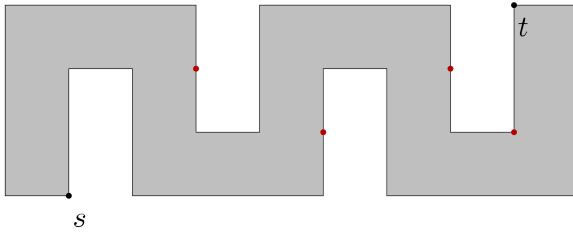


Figure 6: $\lfloor \frac{n}{4} \rfloor - 1$ beacons are sometimes necessary to route between a pair of points in an orthogonal polygon. Here, $n = 20$ and 4 beacons (light filled circles) are required to route from s to t .

always sufficient to cover a simple polygon.

Figure 7 depicts a small polygon with 9 vertices that requires 3 beacons to cover. Due to the angles involved, this small example cannot easily be extended to larger n .

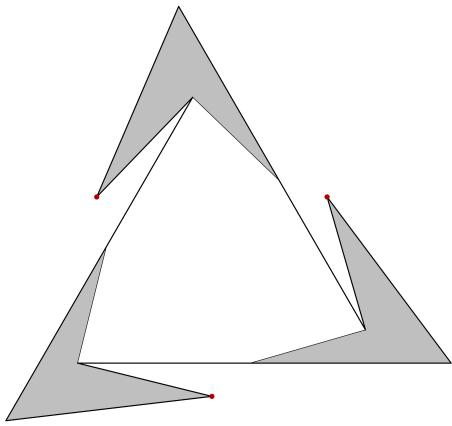


Figure 7: Here, $n = 9$ and $\frac{9}{3} = 3$ beacons are required to cover. The (red) independent witnesses and their disjoint inverse attraction regions are shaded.

In order to get around the angle issue, we try to make multiple copies of the above figure in a linear pattern. This yields a polygon with a repeating spike gadget, requiring $\lfloor \frac{3n}{10} \rfloor$ beacons to cover.

In order to further improve this lower bound, we iteratively ‘glue’ spikes together. We distinguish between spike trunks that are ‘angled-in’ and those that are ‘angled-out’, and proceed to glue two copies of an angled-in spike to form an angled-out spike, as in Figure 9. The next operation takes two such angled-out spikes and glues them together to form a new angled-in spike. This process can then be iterated to generate larger and larger examples.

Note the additional barb added when creating the angled-in spike, which should be removed before start-

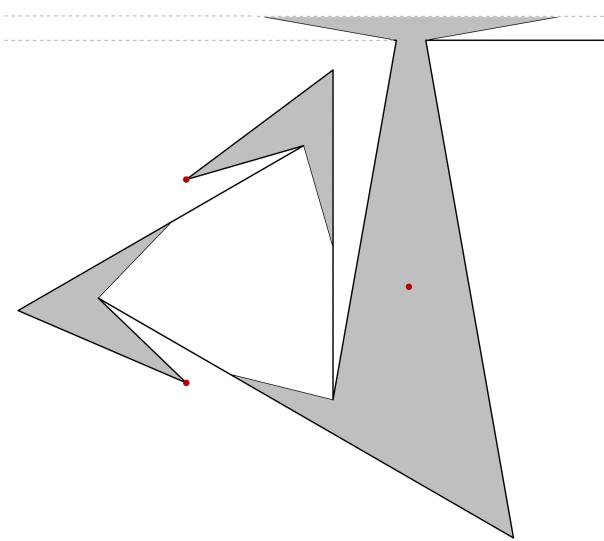


Figure 8: Here, we have a repeatable spike with $n = 10$ and $\frac{3(10)}{10} = 3$ beacons are required to guard. The (red) independent witness points and their disjoint inverse attraction regions are shaded.

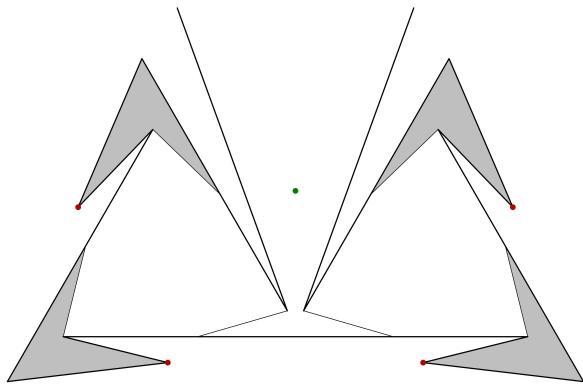


Figure 9: Two copies of the ‘angled-in’ spike from Figure 8 glued together to form an ‘angled-out’ spike.

ing the next iteration. The procedure takes an angled-in spike (with barb) and proceeds as follows:

1. Remove the barb.
2. Glue two copies of the ‘angled-in’ (barb-less) spike together by merging a corresponding pair of angled-in edges and adding new angled-out edges to make an ‘angled-out’ spike.
3. Glue two copies of the ‘angled-out’ spike together by merging a corresponding pair of angled-out edges and adding new angled-in edges to make an ‘angled-in’ spike.
4. Add the barb.

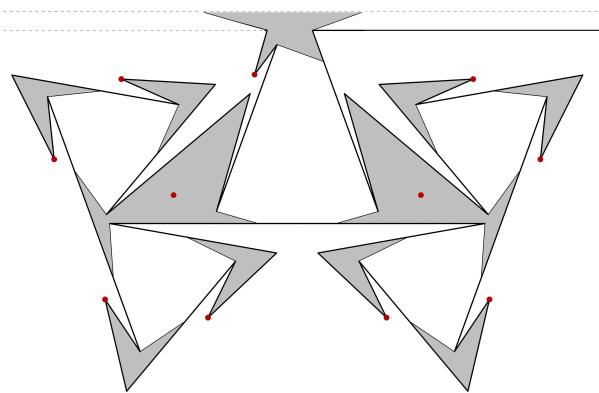


Figure 10: We took two copies of the ‘angled-out’ spike from Figure 9, and glued them together with new vertices to form an ‘angled-in’ spike. Note the extra barb added where the spike attaches to the shaft. It has 36 vertices and requires 11 beacons and may be arbitrarily repeated along the line. Iterating the procedure with this spike as input yields a new spike with 140 vertices that requires 43 beacons.

With respect to the number of vertices, we have that step 1 removes two vertices; step 2 doubles the number of vertices, then merges two edges (deleting two vertices) and adds two new vertices; step 3 doubles the number of vertices again, merges edges (deleting two vertices), then adds two new vertices; step 4 adds two new vertices. Altogether, if the initial spike has n vertices, then after the above procedure we are left with a new closed spike with $4n - 4$ vertices.

With respect to the number of independent witness points, we have that step 1 deletes one witness point, step 2 doubles the number of witness points, step 3 doubles the number of witness points then adds two new witness points, and step 4 adds a new witness point. Altogether, if the original spike has b independent witness points, the new spike has $4b - 1$ independent witness points.

We can now analyze the number of beacons required to cover the above polygons, starting with the closed spike in Figure 8.

Lemma 4 *Starting with the spike depicted in Figure 8, after k iterations of the operation, we are left with an angled-in spike having $\frac{1}{3}(26 \cdot 4^k + 4)$ vertices and $\frac{1}{3}(8 \cdot 4^k + 1)$ independent witness points.*

Proof. Define a function $T(k)$ to be the number of vertices after k iterations. We start with the above spike, so $T(0) = 10$. Using the observations above, we have the recursion $T(k) = 4T(k-1) - 4$. Solving this recursion yields $T(k) = \frac{1}{3}(26 \cdot 4^k + 4)$ vertices.

Define a function $W(k)$ to be the number of independent witness points after k iterations. We start with

the above spike, so $W(0) = 3$. Using the observations above, we have the recursion $W(k) = 4W(k-1) - 1$. Solving this recursion yields $W(k) = \frac{1}{3}(8 \cdot 4^k + 1)$ independent witness points. \square

Using the preceding lemma we may now give an asymptotic lower bound on the number of beacons sometimes necessary to cover arbitrarily large polygons.

Theorem 5 *For an arbitrary polygon P with n vertices and h holes (possibly 0), we may need arbitrarily close to $\lfloor \frac{4n}{13} \rfloor$ beacons to cover P . Conversely, $\lfloor \frac{n+h}{3} \rfloor$ beacons are always sufficient to cover P .*

Proof. We have shown a gluing approach that gives $\frac{1}{3}(8 \cdot 4^k + 1)$ independent witness points and $\frac{1}{3}(26 \cdot 4^k + 4)$ vertices for arbitrary k . The ratio of these as k goes to infinity is $\lim_{k \rightarrow \infty} \frac{\frac{1}{3}(8 \cdot 4^k + 1)}{\frac{1}{3}(26 \cdot 4^k + 4)} = \frac{4}{13}$. In simple polygons, we can then arrange an arbitrary number of angled-in spikes in a line, whereas for polygons with h holes, h angled-out spikes may be closed with an additional vertex and then placed in a convex h -gon. Therefore, we can display a family of polygons that display a requirement for a number of beacons arbitrarily close to $\lfloor \frac{4n}{13} \rfloor$.

The proof of the always sufficient bound is derived from standard art gallery theorems: $\lfloor \frac{n}{3} \rfloor$ or $\lfloor \frac{n+h}{3} \rfloor$ beacons are always sufficient since if a set of beacons ‘sees’ the entire polygon they must also cover the polygon [4, 7]. \square

6 Coverage in Orthogonal Polygons

In this section, we show that, unlike in arbitrary polygons, beacons seem significantly stronger than standard visibility guards in orthogonal polygons. Specifically, we show that if P is an orthogonal polygon, then $\lfloor \frac{n+4}{8} \rfloor$ beacons are sometimes necessary, while $\lfloor \frac{n}{4} \rfloor$ beacons always suffice, due to the standard art gallery bound.

The following figure displays an example of a family of orthogonal polygons that require $\lfloor \frac{n+4}{8} \rfloor$ beacons to cover.

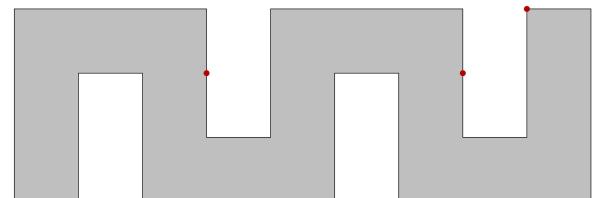


Figure 11: $\lfloor \frac{n+4}{8} \rfloor$ beacons are sometimes necessary to guard an orthogonal polygon. Here, $n = 20$ and $\frac{20+4}{8} = 3$ beacons are required to guard.

There is a gap between the sometimes necessary and always sufficient bounds proved in the above theorem.

We conjecture that the sometimes necessary bound is also sufficient.

Conjecture 2 *Given a simple orthogonal polygon P with n vertices, $\lfloor \frac{n+4}{8} \rfloor$ beacons are sometimes necessary and always sufficient to cover P .*

Acknowledgements

This research has been partially supported by the National Science Foundation (CCF-1018388) and the US-Israel Binational Science Foundation (project 2010074).

References

- [1] M. Biro, J. Iwerks, I. Kostitsyna, J. S. B. Mitchell. Beacon-Based Algorithms for Geometric Routing. *Proc. of the 13th Algorithms and Data Structures Symposium (WADS 2013)*, London, Ontario, Canada, August 2013. To appear.
- [2] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, J. S. B. Mitchell. Beacon-based structures in polygonal domains. In *Abstracts of the 1st Computational Geometry: Young Researchers Forum (CG:YRF 2012)*, 2012.
- [3] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, J. S. B. Mitchell. Beacon-based routing and coverage. In *21st Fall Workshop on Computational Geometry (FWCG 2011)*, 2011.
- [4] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory Series B*, 18:39–41, 1975.
- [5] E. Györi, F. Hoffman, K. Kriegel, T. Shermer. Generalized guarding and partitioning for rectilinear polygons. *Computational Geometry: Theory and Applications*, 6(1):21–44, 1996.
- [6] J. Kahn, M. Klawe, D. Kleitman. Traditional Galleries Require Fewer Watchmen. *SIAM J. on Algebraic and Discrete Methods*, 4(2):194–206, 1983.
- [7] F. Hoffman, M. Kaufmann, K. Kriegel. The art gallery theorem for polygons with holes. *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS 1991)*, 39–48.

Privacy by Fake Data: A Geometric Approach

Victor Alvarez*

Erin Chambers†

László Kozma‡

Abstract

We study the following algorithmic problem: given n points within a finite d -dimensional box, what is the smallest number of extra points that need to be added to ensure that every d -dimensional unit box is either empty, or contains at least k points. We motivate the problem through an application to data privacy, namely k -anonymity. We show that minimizing the number of extra points to be added is strongly NP-complete, but admits a Polynomial Time Approximation Scheme (PTAS). In some sense, this is the best we can hope for, since a Fully Polynomial Time Approximation Scheme (FPTAS) is not possible, unless P=NP.

1 Introduction

Data privacy is a fundamental problem associated to data mining. On one hand, we would like to make data publicly available so that data mining or analysis is possible. On the other hand, we would like to make sure that the identity of an individual is not disclosed and no extra information is revealed as a result of mining. Several approaches have been proposed for alleviating the inherent tension between the two goals. Two of the more popular frameworks are k -anonymity [3] and differential privacy [7].

In differential privacy, one controls the way a database is accessed, and adds noise to the results of queries to the database. The idea is essentially to ensure that the results of any query, or analysis, with or without the data of one individual have similar distributions.

The idea behind k -anonymity is to ensure that for every query there are at least k records that are indistinguishable from each other. This is usually achieved by suppression or generalization, i.e., selective deletion of parts of data - which hopefully does not substantially affect the results of analysis using the data. The larger the value of k , the greater the extent of privacy. Meyerson and Williams [5] have studied the complexity of

computing the minimum amount of generalization and suppression necessary for k -anonymity, and proved that it is an NP-hard problem. They also give an $O(k \log k)$ approximation algorithm. LeFevre, DeWitt and Ramakrishnan [6] studied the optimization problem in a multidimensional model. They proved NP-hardness and gave a greedy algorithm that seems to perform well in practice.

In this paper we concentrate on achieving k -anonymity, assuming that the queries are sufficiently *broad*. This condition describes a situation in which an adversary has only partial or inaccurate information about an individual. The goal is to prevent disclosure of identity in this setting.

Now assume that the query is broad, but still the database returns only a small number of records. What should we do in such a case? One easy solution is to refuse answering such queries. This is not effective, since the adversary can make several broader queries which contain the unanswered query range, and then take their intersection to determine which records belong to the unanswered range. Another easy solution is to append some fake data on the spot, so that the total number of records returned is at least k . This is not effective either since an adversary can make several similar queries and observe that only certain records are present in all of the returned results, thereby finding out that these are the only real data. However, this approach can be made effective if we can be consistent about the fake data. The idea is to insert a fixed set of fake data points all at once into the database such that the answer to any broad query either returns no records or at least k records.

We represent data records having multiple attributes as points in multidimensional space. This view is naturally suited for numeric data. We assume that queries are axis-parallel hyper-rectangles, which we will simply call d -dimensional boxes, that have certain minimum width in every dimension. The specific minimum width in each dimension can be different and needs to be chosen appropriately depending on the data. However, by appropriate scaling we can assure that the minimum width in every dimension is exactly one.

We would like the amount of fake data to be as small as possible. It is intuitive that in general the amount of fake data required is much smaller than the size of the database, since data is often densely concentrated in certain regions, and fake data is required only for

*Fachrichtung Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, 66123, Germany, alvarez@cs.uni-saarland.de

†Department of Mathematics and Computer Science, Saint Louis University, echambe5@slu.edu

‡Fachrichtung Informatik, Universität des Saarlandes, kozma@cs.uni-saarland.de

sparse regions. For example, if the multidimensional volume of the domain of the data is V , our broad query ranges have volume at least v , and there are n data points chosen uniformly at random from the domain, then the expected number of data points in a specific broad query range is nv/V . If this number is much larger than k , then with high probability a broad range is already k -anonymous, and needs no fake data. An easy calculation shows that the number of fake points required goes down exponentially with $nv/(kV)$.

1.1 Recasting to a geometric setting

The main geometric problem studied in this paper is the following: given a set P of n points in a d -dimensional box¹ $\mathbb{D} \subseteq [0, s]^d$ ($s \geq 1$), what is the smallest number of additional points that need to be added to P , so that every d -dimensional unit box contained in \mathbb{D} is either empty or contains at least k points.

Notice that, if we want to hit all unit hypercubes, not just the non-empty ones, then this is a standard hitting set problem, with an obvious solution. The restriction to non-empty hypercubes is precisely what makes the problem difficult. This situation is similar to that of other hitting set problems, in which the ranges that we are interested in are defined implicitly. For example, when studying ε -nets, one is interested in hitting all ranges which have size at least εn . Implicitly defined hitting set problems also appear in combinatorial settings such as the feedback vertex set problem, where the goal is to pick the smallest set of vertices that hit all cycles of a graph.

No general technique is known to solve problems of this kind, and the methods for solving individual problems are varied. Our problem (defined formally in the next section) is motivated by the discussion in the previous section, and focuses on approximation algorithms for achieving k -anonymity.

1.2 Our contribution

Motivated by the above discussion, we define the following notions:

Definition 1 A set P of n points contained in a box $\mathbb{D} \subseteq [0, s]^d$ ($s \geq 1$) is k -anonymous, for some given $k \geq 1$, if and only if every box of unit size contained in \mathbb{D} is either empty or it contains at least k points of P .

Note that any collection of points is trivially 1-anonymous. We therefore concern ourselves only with the case $k \geq 2$.

Definition 2 Given a set P of n points as before, a k -anonymizer of P is a set $\mathcal{A} \subset \mathbb{D}$ of extra points such that $P \cup \mathcal{A}$ is k -anonymous.

¹In this paper, all boxes considered are axis-parallel.

Our goal is to find a k -anonymizer of smallest cardinality. We call this an *optimal k -anonymizer*. The decision version of the problem is the following:

k -Anonymity:² Given a set P of n points as before and an integer l , is there a k -anonymizer of P of size at most l ?

The results achieved in this paper are the following:

Theorem 1 k -ANONYMITY is strongly NP-complete, even for $k = 2$.

While we prove this for the two-dimensional case only, the result trivially implies the NP-completeness of the problem in any dimension $d \geq 2$. On the positive side, we give a polynomial-time approximation scheme (PTAS):

Theorem 2 Let OPT denote the size of an optimal k -anonymizer for a set P of n points in $\mathbb{D} \subset \mathbb{R}^d$. Then, given $0 < \varepsilon \leq 1$, a k -anonymizer of P , of size at most $(1 + \varepsilon)OPT$ can be computed in $O((knd/\varepsilon)^{\text{poly}(k, d/\varepsilon)^d})$ time.

Note that a fully polynomial time approximation algorithm (FPTAS) is not possible for strongly NP-complete problems, unless P=NP. Also, as the exponents in our approximation scheme are prohibitively large, we do not claim direct applicability of the algorithm, thus Theorem 2 should rather be taken as an existential result.

The rest of the paper is organized as follows: in Section 2 we introduce a dual setting, which is equivalent to the original problem but provides a better setting in which to prove our results. In Sections 3 and 4 we prove Theorems 1 and 2, respectively.

2 Dual setting

For convenience, we work in a “dual” setting based on our “primal” setting of input points/boxes, where we replace points by boxes and boxes by points. Each $p \in P$ gets mapped to the full dimensional unit box with its center at p , and every unit box $B \subset \mathbb{D}$ in the primal setting gets mapped to its center. This way, a set of n points P gets mapped to a set \mathcal{Q} of n unit boxes. Observe that incidences between points and boxes are preserved by this transformation.

In all collections of points or boxes that we mention, we allow multiple copies of the same element. For simplicity we call these collections sets, even though technically they are multisets.

Given a set \mathcal{Q} of n unit boxes, and a point $p \in \mathbb{D}$, we define the *depth* of p as the number of elements of \mathcal{Q} that contain p . We now have the following dual definition of k -anonymity:

²Our definition of k -ANONYMITY is slightly different from existing formulations in the literature, however, due to the strong similarity, we retained the term.

Definition 3 Let \mathcal{Q} be a set of n unit boxes of dimension d contained in a box $\mathbb{D} \subseteq [0, s]^d$ ($s \geq 1$). We say that \mathcal{Q} is a k -anonymous arrangement of boxes if and only if the depth of every point $p \in \mathbb{D}$ is either 0 or at least k .

The equivalence between the two definitions follows from this simple observation: If p is a point and B is a unit box containing p , then the unit box centered at p contains the center of B .

Now the task is to find a set \mathcal{A} of unit boxes (representing the extra points in the primal) of minimum cardinality such that $\mathcal{Q} \cup \mathcal{A}$ is k -anonymous. For completeness, we have the following formal definition of the decision version:

k -Anonymity (dual): Given a set \mathcal{Q} of n unit boxes and an integer l , is there a k -anonymizer for \mathcal{Q} with at most l boxes?

3 NP-completeness: proof of Theorem 1

First we show that k -ANONYMITY \in NP. We consider an instance of the problem in the primal version (a set P of n points in a rectangle $\mathbb{D} \subset \mathbb{R}^2$ and a threshold l) and a candidate solution (a set \mathcal{A} of t points in \mathbb{D}). We need to verify in time polynomial in $n + t$ that the solution is correct, i.e. $P \cup \mathcal{A}$ is k -anonymous and that $|\mathcal{A}| \leq l$. The latter can be checked with a simple counting, which takes $O(t)$ time. It remains to be shown that we can verify k -anonymity of a set of points in polynomial time.

Let us call a unit box in the primal setting a *test box*. There are an infinite number of locations in which a test box can be placed, but the following observation shows that it is sufficient to verify $O((n + t)^2)$ of them: If we move a test box continuously, the number of points inside does not change as long as the sides of the box do not cross any point. If we do not meet any points, we stop at the boundary of \mathbb{D} . It is therefore sufficient to look at test boxes in particular locations: one of the vertical sides of the test box touches a point or the boundary of \mathbb{D} and one of the horizontal sides touches a point or the boundary of \mathbb{D} . By convention we do not count points on the top- or on the right side of the test box and we count points on the bottom- or on the left side as well as points inside the test box. Verifying that in all of these test boxes there are either at least k points or none, requires polynomial time.

Now we prove NP-hardness. This is done using a reduction from PLANAR3SAT, a known NP-complete decision problem [2]. In 3SAT, given a formula ϕ in 3-CNF, we ask whether there exists an assignment of truth values to the variables, such that ϕ evaluates to *true*. In PLANAR3SAT we restrict the question to planar formulae: those that can be represented as a planar graph in which vertices correspond to both variables and clauses of the formula and there is an edge between

clause C and variable x if and only if C contains either x or $\neg x$.

Knuth and Ragunathan [4] observed that PLANAR3SAT remains NP-complete if we restrict it to formulae having the following *rectilinear embedding*: variables are placed on a line, clauses are placed on the two sides of the line and the three legs of each clause are properly nested (Figure 1(a)).

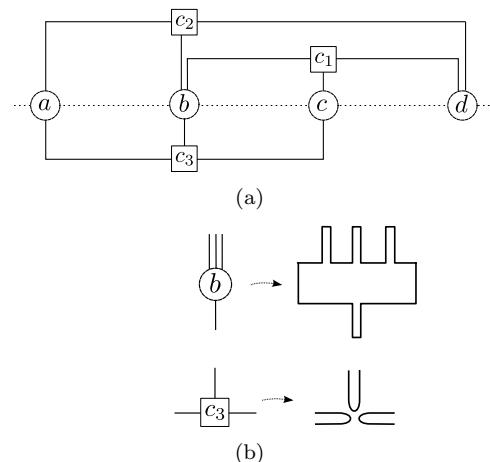


Figure 1: (a) Rectilinear embedding of the formula $(b \vee \neg c \vee \neg d) \wedge (a \vee \neg b \vee d) \wedge (\neg a \vee b \vee c)$.
(b) A variable with three connections on top and one at the bottom and its corresponding gadget (above). Clause and its gadget (below).

Given an instance of PLANAR3SAT with an embedding as described before, we transform it into a two dimensional instance of the (dual) k -ANONYMITY decision problem.

The unit squares of the set \mathcal{Q} are placed so as to align with an orthogonal grid with cells of size $\frac{1}{5} \times \frac{1}{5}$. The placement of \mathcal{Q} will create the following types of regions in \mathbb{D} : (a) *empty* regions, consisting of points with depth 0, (b) *uncovered* regions consisting of points with positive depth less than k , which need to be “fixed” by the k -anonymizer and (c) *safe* regions consisting of points with depth at least k . Our construction will assure that all *uncovered* regions have a depth of exactly $k - 1$, therefore we need not add multiple copies of the same square in the solution.

We can create an *uncovered* square of size $\frac{1}{5} \times \frac{1}{5}$ in the following way: we put $k - 1$ squares in the same place, k squares shifted to the right by $\frac{1}{5}$ and k squares shifted upwards by $\frac{1}{5}$. We call the resulting uncovered square a *patch* and it is the main element in our reduction.

Our construction is such that *patches* are surrounded by large *safe* regions. Consider a box $B \in \mathcal{A}$ that covers one or more patches created by the input set \mathcal{Q} . The parts of B that do not cover a patch fall entirely within the safe region surrounding the patches. In this way

we ensure that the boxes in the k -anonymizer \mathcal{A} do not create new uncovered regions.

Our main gadget is a sequence of patches which we call a *wire* (Figure 2). Note that the wire can be extended infinitely at both ends. The wire has two important properties: (1) a unit square can cover any two neighboring patches, such that the leftover part falls entirely in the safe region, and (2) no unit square can cover more than two patches of the wire. In Figure 2, uncovered patches are colored black, the surrounding safe region is gray. We also show the corresponding points in the primal diagram, together with their multiplicities.

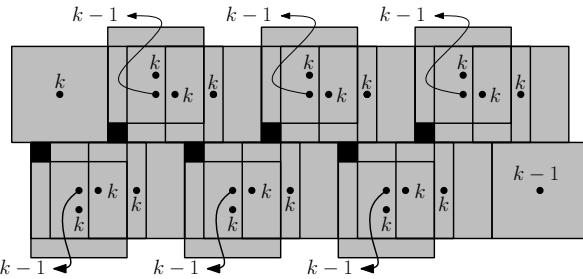


Figure 2: Wire gadget in dual version and primal points with their multiplicities.

We can also design a *bend* gadget, which will introduce 90 degrees turns in a wire, shown in Figure 3. Both of the previously mentioned properties of a wire are also preserved by the bend gadget, which we illustrate with dotted unit squares that cover neighboring patches..

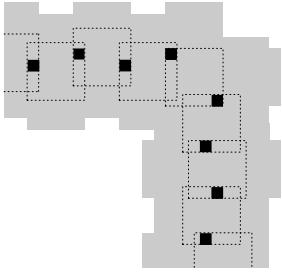


Figure 3: Bend gadget (dual) and unit squares that cover neighboring patches.

Now we can create loops and we will represent each variable by a loop that contains an even number of patches. Let the patches in a loop be numbered from 1 to $2m$. It can be observed that the optimal k -anonymizer of this loop has m boxes, each covering two neighboring patches. The boxes cover either the patches $(1, 2), (3, 4), \dots, (2m - 1, 2m)$, or the patches $(2, 3), (3, 4), \dots, (2m, 1)$. The choice between the two optimal solutions encodes the truth value of a variable.

We transmit the value of a variable with a *tentacle* extending from the main loop. A tentacle consists of two parallel wires with a sufficient distance between them to avoid interference. A clause gadget is the meeting point

of three such tentacles. The variable and the clause gadget are schematically presented in Figure 1(b). The line here indicates a wire, without showing the actual patches.

We will show the clause gadget in more detail in the full version of this paper. However, as mentioned before, it is the meeting point of three variable tentacles. Besides the patches that make up the variable tentacles, the clause contains an extra patch in the middle. This patch is placed in such a way that it is reachable by a square of the optimal covering of either of the three variables, but only if the variable is in one particular state. By convention, we consider this the *true* state. This means that if all three variables are *false*, we need an extra square to cover the patch in the middle. This penalty is the key ingredient of our reduction.

We have not yet discussed negated variables. If a variable appears negated in a clause, we need to lengthen the corresponding tentacle by one patch on each side, so that the two patches nearest to the clause center are now covered by a single square in the *false* state. We can achieve such a shift by replacing a small piece of a wire by a *condensing* gadget. This gadget increases the number of patches by one, while keeping the endpoints in place and maintaining properties (1) and (2) of a wire. We omit the details of this gadget, as it is a straightforward construction.

Our reduction is almost complete; what is left is the computation of the parameter l in the k -ANONYMITY instance. Let the total number of patches in wires (excluding the extra patches in the middle of clauses) be $2m$. Then we set $l = m$ and conclude that there exists a k -anonymizer of size at most l if and only if the original PLANAR3SAT instance has a satisfying assignment.

For completeness, we need to prove that our construction is of polynomial size (in terms of the number of clauses and variables of the PLANAR3SAT instance). First we observe that the resulting construction can be bounded by a box of polynomial size: the height of the box depends on the maximum level of nesting in the embedding of the formula, but each additional level results in an increase of constant size. The number of levels is bounded by the number of clauses in the formula. The width of the rectangle increases with the addition of each new variable or clause, but again, only by a constant additive term. Since our construction consists of points with multiplicity at most k aligned with a grid of size $\frac{1}{5} \times \frac{1}{5}$, the total number of points needed is less than $25k$ times the size of the bounding box, which is clearly polynomial. This concludes the proof of Theorem 1.

4 PTAS: proof of Theorem 2

As before, all this section will take place in the dual setting, where the input set of points P is represented by

an arrangement of boxes \mathcal{Q} . The main result of this section is the proof of correctness of the following randomized algorithm that computes a k -anonymizer of size at most $(1 + \varepsilon)OPT$, where OPT denotes the size of an optimal k -anonymizer of \mathcal{Q} . This algorithm is based on a technique developed by Hochbaum and Maass [1]. Additionally, at the end of the section we will discuss how to derandomize our algorithm..

Algorithm 1

Input: A set \mathcal{Q} of n unit boxes in $\mathbb{D} \subseteq [0, s]^d$ ($s \geq 1$).
Output: An anonymizer of size $(1 + \varepsilon)OPT$ for \mathcal{Q} .

1. Given $0 < \varepsilon \leq 1$, choose $L = (2d/\varepsilon)$ and a random integer $r \in [0, L - 1]$.
2. Impose a grid \mathbb{G} over domain \mathbb{D} of cell size L , and with offset r from the origin in every dimension.
3. Find an optimal k -anonymizer inside every non-empty cell of \mathbb{G} .
4. Output the union of the solutions of the cells of \mathbb{G} .

In Step 2 of the algorithm, offset r from the origin means that the coordinates of every grid point inside domain \mathbb{D} are of the form $(r + cL)$, where $c \geq 0$ is integer. The grid points on the boundary of \mathbb{D} are automatically defined. Note that *non-empty* refers to the dual view: we consider a cell empty if all its points have depth 0 or at least k .

As the reader can note, the only step in the algorithm that is non-trivial is Step 3, in which we have to compute an exact solution of a subproblem contained in a smaller domain. In order to make the presentation simpler, we first focus on the case $d = 2$. We then look at the general case, and then consider derandomization of Algorithm 1.

Given two axis-parallel squares, we say that they are *aligned* if and only if they intersect only at their boundaries, i.e. their intersection is non-empty, but they have disjoint interiors. Now, given a set of unit squares $\mathcal{Q} = \{Q_1, \dots, Q_n\} \subset \mathbb{D} \subseteq [0, s]^2$, we can define a grid $\mathbb{G}_{\mathcal{Q}}$ over \mathbb{D} as follows:

Let $Q \in \mathcal{Q}$ and define \mathbb{G}^Q to be the unit grid over \mathbb{D} having Q as a cell. Denote by $E(\mathbb{G}^Q)$ the set of grid lines of \mathbb{G}^Q . The set of grid lines of $\mathbb{G}_{\mathcal{Q}}$ is $\bigcup_{i=1}^n E(\mathbb{G}^{Q_i})$, and its vertex set is the set of intersection points among its grid lines. We now have the following lemma:

Lemma 3 *Let $\mathcal{Q} \subset \mathbb{D}$ and $\mathbb{G}_{\mathcal{Q}}$ be as defined before. Then there exists an optimal k -anonymizer \mathcal{A} of \mathcal{Q} such that each of its elements has its vertices at grid points of $\mathbb{G}_{\mathcal{Q}}$.*

Proof. We show that there is an optimal k -anonymizer that is aligned in the horizontal direction. By a similar argument it can be shown that there is an optimal k -anonymizer that is aligned in both the horizontal and vertical directions.

Let us proceed by contradiction. Let \mathcal{A} be the optimal k -anonymizer with the least number of elements whose vertical sides are not aligned with the vertical grid lines of $\mathbb{G}_{\mathcal{Q}}$. Denote by U this set of “unaligned” elements of

\mathcal{A} . If $U = \emptyset$ then we are done, so we will assume that $U \neq \emptyset$. If we manage to move the elements of \mathcal{A} around, such that the cardinality of U decreases, we are done.

We will move the boxes in U to the right simultaneously and at the same speed until we are about to de-anonymize some region and are forced to stop. At that point, some element of U gets aligned (horizontally) with some element $B \in \mathcal{Q} \cup (\mathcal{A} \setminus U)$, and thus automatically with a vertical grid line of $\mathbb{G}_{\mathcal{Q}}$, since B was already aligned. Observe that during this process we do not de-anonymize any region of \mathbb{D} but we “align” one element of U . This is a contradiction since we assumed that U was of minimum cardinality.

Once we have an optimal solution that is horizontally aligned to $\mathbb{G}_{\mathcal{Q}}$ we can repeat the argument with a solution whose boxes are horizontally aligned and a minimum number of them are vertically unaligned. \square

We can now perform Step 3 of Algorithm 1 in polynomial time:

Lemma 4 *Let $\mathcal{Q} = \{Q_1, \dots, Q_n\}$ be a set of unit squares contained in a square of side-length L . Then a k -anonymizer of minimum cardinality can be found in time $O((knL)^{\text{poly}(k, L^2)})$.*

The proof is based on Lemma 3. Since there exists an optimal k -anonymizer in which every element is aligned with the grid, we exhaustively search all candidate sets in increasing order of size, until we find a solution. We defer the details to the full version of this paper.

Note that in practice L and k might be small, and independent of n , giving a running time of the sort $O(n^c)$, for some constant c , which is thus polynomial in n . We now prove a result that implies Theorem 2 for the case $d = 2$, by setting $L = 4/\varepsilon$.

Theorem 5 *Let $\mathcal{Q} \subset \mathbb{D}$ be a set of n unit squares defined as before. Then Algorithm 1 computes a k -anonymizer of \mathcal{Q} of size at most $(1 + \varepsilon)OPT$ in time $O((knL)^{\text{poly}(k, L^2)})$.*

Proof. To achieve the desired running time, we run the algorithm of Lemma 4 in each non-empty cell of the grid \mathbb{G} imposed over domain \mathbb{D} in Step 2 of Algorithm 1. Since there are at most n non-empty cells, the overall running time is $O((knL)^{\text{poly}(k, L^2)})$. Observe that the cell-size is at most $L = \frac{4}{\varepsilon}$, which is independent of n .

As for the quality of approximation, let $0 < \varepsilon \leq 1$ be a given parameter. Let OPT denote the size of an optimal solution \mathcal{A} . By the previous discussion, we know that each element of \mathcal{A} , a unit square, can intersect (1) one vertical line and no horizontal line, or (2) one horizontal line and no vertical line, or (3) one vertical and one horizontal line, i.e. it can contain exactly one grid point of \mathbb{G} , or (4) lie entirely in a cell of \mathbb{G} . Now consider

some $q \in \mathcal{A}$. If q is of type (1) or (2), note that q then intersects exactly two cells C, C' of \mathbb{G} . In this case, we will create another copy q' of q and move q to C and q' to C' taking care that $\mathcal{A} \cup \{q'\}$ remains a k -anonymizer, although not of minimum cardinality anymore. If q is of type (3), then we will create three more copies of it and distribute the four of them among the four cells of \mathbb{G} that q intersects. By performing these operations on every element of \mathcal{A} of type (1), (2), or (3) we create a new k -anonymizer \mathcal{A}' with the property of having each element inside some cell of \mathbb{G} .

Let us denote by OPT' the size of the solution obtained by Algorithm 1. Let C be a cell of \mathbb{G} and observe that the local solution given by \mathcal{A}' on C must be at least as large as the local solution provided by Algorithm 1, since the latter is optimal for C . Therefore we obtain that $OPT' \leq |\mathcal{A}'|$. We can think of \mathcal{A}' as a version of \mathcal{A} with a penalty. Note that by the random offset r of \mathbb{G} , an element Q of \mathcal{A} can intersect a vertical or horizontal line with probability $\frac{1}{L}$, since r takes values from 0 to $L - 1$, and Q is a unit square, so Q gets penalized only in one out of the L unit intervals of $[0, L]$. Note as well that Q intersects a vertical and a horizontal line independently, so the expected penalty of Q is $1 \cdot p_1 + 1 \cdot p_2 + 3 \cdot p_3 + 0 \cdot p_4$, where p_i is the probability that Q is of type (i). We obtain (using the fact that $L = \frac{4}{\varepsilon}$):

$$\begin{aligned}\mathbb{E}[\text{Penalty of } Q] &= \frac{1}{L} \left(1 - \frac{1}{L}\right) + \frac{1}{L} \left(1 - \frac{1}{L}\right) + \frac{3}{L^2} \\ &= \frac{2}{L} + \frac{1}{L^2} \leq \frac{3}{L}.\end{aligned}$$

Therefore, $\mathbb{E}[OPT'] \leq \mathbb{E}[|\mathcal{A}'|] \leq |\mathcal{A}| + |\mathcal{A}| \cdot \frac{3}{L} = (1 + \frac{3}{L}) OPT < (1 + \varepsilon) OPT$. \square

Note that all arguments carry over to higher dimensions. We summarize the result in the following theorem, which implies Theorem 2 for general dimension d by setting $L = 2d/\varepsilon$. We leave the details of the proof for the full version of this paper.

Theorem 6 *Let $\mathcal{Q} \subset \mathbb{D}$ be a set of n unit boxes in d dimensions, and let $L = \frac{2d}{\varepsilon}$. Then Algorithm 1 computes a k -anonymizer of \mathcal{Q} of size at most $(1 + \varepsilon)OPT$ in time $O((kdnL)^{\text{poly}(k, L^d)})$.*

Finally, observe that the random offset r is an integer in the interval $[0, L - 1]$. Since the expected size of the computed k -anonymizer is $(1 + \varepsilon)OPT$, we can try each possible value of r , and pick the smallest k -anonymizer. This derandomizes Algorithm 1 adding a factor of L to the running time.

5 Conclusions

In this paper, we presented a new notion of k -anonymity that uses fake data to achieve anonymity assuming that queries are *broad*. We studied the complexity of the associated optimization problem in a geometric framework, which allowed us to leverage techniques available in computational geometry. We proved strong NP-completeness and gave a PTAS in fixed dimensions and for constant k . Note that this is mostly of theoretical interest: the exponent in the running time is very large, even in two dimensions and for small k . It is still not clear whether the exact dependence on the number of data points can be improved. One can easily imagine a situation where the number of points is arbitrarily large but the optimal solution remains the same for a much smaller subset. It may be possible to sample a subset of the input points, from which with high probability a good approximation may be obtained by using our algorithm on the sample.

Acknowledgements

We thank Saurabh Ray for suggesting the problem and for fruitful discussions. Erin Chambers is partially supported by NSF grant CCF 1054779.

References

- [1] D. Hochbaum, W. Maass, *Approximation schemes for covering and packing problems in image processing and VLSI*, J. ACM, **31:1**:130–136, 1985.
- [2] D. Lichtenstein, *Planar formulae and their uses*, SIAM J. Comput., **11:2**:329–343, 1982.
- [3] L. Sweeney, *k -Anonymity: A Model for Protecting Privacy*, Intl J. of Uncertainty, Fuzziness and Knowledge-Based Systems, **10:5**:557–570, 2002.
- [4] D.E. Knuth, A. Raghunathan, *The problem of compatible representatives*, SIAM J. Discret. Math., **5:3**:422–427, 1992.
- [5] A. Meyerson, Ryan Williams, *On the Complexity of Optimal K -Anonymity*, Proc. of the Twenty-third ACM SIGACT-SIGMOD-SIGART, 223–228, 2004.
- [6] K. LeFevre, D. J. DeWitt, R. Ramakrishnan, *Mondrian Multidimensional K -Anonymity*, Proc. of the 22nd Intl Conf. on Data Engineering, ICDE 2006, 3–8 April 2006, Atlanta, GA, USA
- [7] C. Dwork, F. McSherry, K. Nissim, and A. Smith, *Calibrating noise to sensitivity in private data analysis*, Proc. of the 3rd Theory of Cryptography Conf., 265–284, 2006.

Stabbing Polygonal Chains with Rays is Hard to Approximate

Steven Chaplick*

Elad Cohen†

Gila Morgenstern‡

Abstract

We study a geometric hitting set problem involving unidirectional rays and curves in the plane.

We show that this problem is hard to approximate within a logarithmic factor even when the curves are convex polygonal x -monotone chains. Additionally, it is hard to approximate within a factor of $\frac{7}{6}$ even when the curves are line segments with bounded slopes. Lastly, we demonstrate that the problem is $W[2]$ -complete when the curves are convex polygonal x -monotone chains and is $W[1]$ -hard when the curves are line segments.

1 Introduction

Motivated by art-gallery problems such as terrain-guarding and minimum-link watchman route, Katz, Mitchell and Nir [7] studied a family of geometric stabbing/hitting problems involving orthogonal line segments and rays in the plane. Among other problems, they introduced the following problem of *Stabbing Segments with Rays* (SSR).

PROBLEM: SSR.

INPUT: A set S of non-vertical line segments and a set R of upwards rays.

OUTPUT: A minimum cardinality subset R' of R so that for each $s \in S$ there exists $r \in R'$ for which $r \cap s \neq \emptyset$.

In addition to the SSR problem we also consider the more general problem of *Stabbing polygonal Chains with Rays* (SCR); defined below.

PROBLEM: SCR.

INPUT: A set C of polygonal chains and a set R of upwards rays.

OUTPUT: A minimum cardinality subset R' of R so that for each $c \in C$ there exists $r \in R'$ for which

*Department of Applied Mathematics, Charles University in Prague, Czech Republic, chaplick@kam.mff.cuni.cz. The majority of this research was completed when S. Chaplick was a visiting researcher at the Caesarea Rothschild Institute, University of Haifa, Israel in July 2012. Partial support by the ESF GraDR EUROGIGA grant as project GACR GIG/11/E023.

†Caesarea Rothschild Institute, University of Haifa, Israel, eladdc@gmail.com.

‡Caesarea Rothschild Institute, University of Haifa, Israel, gilamor@cri.haifa.ac.il.

$$r \cap c \neq \emptyset.$$

Katz, Mitchell and Nir [7] presented an exact poly-time solution for the variant of SSR in which the line segments in S are non-intersecting, and left the more general problem (involving intersecting line segments) open.

An equivalent problem was studied by Chan and Grant [1] who refer to it as the problem of *hitting-set of downward shadows of line segments in the plane with points*. A *downward shadow* of a set $Y \subset \mathbb{R}^2$ is the set of all points $p = (p_x, p_y)$ in the plane for which there exists a point $q = (q_x, q_y)$ in Y with $q_x = p_x$ and $q_y \geq p_y$. Chan and Grant have also presented an exact poly-time solution; both solutions [1, 7] are similar and based on dynamic programming.

Chan and Grant [1] studied several other related problems, including covering and packing. For example they showed that the hitting set of downward shadows of horizontal line segments by points and its "almost-dual" problem (covering points with downward shadows of horizontal line segments) are both poly-time solvable. They further showed that the covering of points with downward shadows of 2-intersecting¹ x -monotone curves is poly-time solvable.

In contrast to the fact that both covering and hitting problems with downward shadows of horizontal line segments are poly-time solvable, for downward shadows of 2-intersecting x -monotone curves, things are different. Chan and Grant [1] remarked that a naive attempts to generalize their solution to a solution for the hitting problem of downward shadows of curves appear to fail. They left it as an open problem to determine whether the hitting set problem involving downward shadows of 2-intersecting x -monotone curves in the plane is APX-hard, has a PTAS, or perhaps is even poly-time solvable. In this paper we show that it is APX-hard, even for the simple case of downward shadows of line segments with bounded slopes.

In Section 2 we show that SCR is hard to approximate within a logarithmic factor. In Section 3 we show that although simpler, SSR is still APX-hard and is hard to approximate within a factor of $\frac{7}{6}$. We further observe that these results hold even if the slopes of all line segments are bounded. In Section 4 we show that SCR is

¹A collection C of curves is said to be *k-intersecting* if every two curves in C intersect at most k times.

$W[2]$ -complete and that SSR is $W[1]$ -hard; thus both are most unlikely to be fixed-parameter intractable.

2 Stabbing polygonal chains with rays

We show below that SCR is APX-hard via a reduction from HITTING-SET (HS).

PROBLEM: HS.

INPUT: A set U and a collection \mathcal{S} of subsets of U .

OUTPUT: A minimum cardinality subset U' of U so that for every $S \in \mathcal{S}$ we have that $S \cap U' \neq \emptyset$.

Theorem 1 *SCR is APX-hard, and is hard to approximate within a logarithmic factor.*

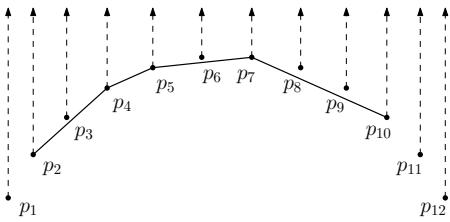


Figure 1: A chain corresponding to $\{2, 4, 5, 7, 10\}$. It can be stabbed only by the rays with origins at the corresponding points.

Proof. We show that HS is reducible to SCR via a cost-preserving reduction; since HS is hard to approximate within a logarithmic factor [4], the desired result follows.

Let \mathcal{S} be a collection of subsets of $[n]$. We construct sets C of polygonal chains and R of upwards rays so that a minimum hitting set for \mathcal{S} corresponds to a minimum subset of R that stabs all chains in C and vice versa. Indeed, let $P = \{p_i : 1 \leq i \leq n\}$ be a point set lying on the upper half of a circle, left to right, and let R be the set of upwards rays with origins in P . Starting with an empty set C , for each set $S \in \mathcal{S}$, add to C the (convex) polygonal chain with vertices at points corresponding to elements of S . More precisely, put $S = \{i_1, i_2, \dots, i_m\} \subseteq [n]$ where $1 \leq i_1 < i_2 < \dots < i_m \leq n$ then the corresponding chain is $c = \langle p_{i_1}, p_{i_2}, \dots, p_{i_m} \rangle$; see Figure 1. It is easy to see that a chain $c \in C$ is stabbed by exactly those rays with origins at its vertices, just as the set $S \in \mathcal{S}$ is hit by exactly its elements. That is, a minimum hitting set for \mathcal{S} corresponds to a minimum subset of R that stabs all chains in C (of the same cardinality) and vice versa. \square

3 Stabbing segments

SSR is a special case of SCR. Although simpler, it is still APX-hard. The reduction in the proof of Theorem 1 is applicable also for special case in which the collection S

is a collection of pairs, and consequently, the resulting chains are line segments. That is, the reduction in the proof of Theorem 1 is also a reduction from VERTEX-COVER (VC) to SSR, as is described in the proof of Theorem 2 below.

Theorem 2 *SSR is APX-hard and is hard to approximate within a factor of $\frac{7}{6}$.*

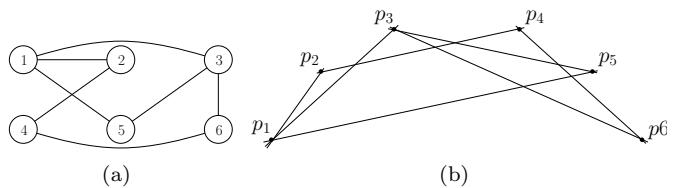


Figure 2: (a) A graph G and (b) its corresponding appearance of SSR.

Proof. Let $G = (V, E)$ be a graph. As in the proof of Theorem 1, P is a point set embedded to the upper half of a circle, now corresponding to the vertices of G , R is a set of upwards rays with origins in P , and S is a set of line segments with endpoints in P where for each edge $(v_i, v_j) \in E$, the line segment $\overline{p_ip_j}$ is in S ; see Figure 2 for an illustration. It is easy to see that a minimum vertex cover of G corresponds to a minimum subset of R that stabs all segments in S (of the same cardinality) and vice versa. Since VC is hard to approximate within a factor of $\frac{7}{6}$ [6], then we are done. \square

Notice that the reduction in proof of Theorems 1 and 2 could be adjusted, by sliding the points of P , to fit restricted versions of SCR and SSR in which all slopes of the segmental links are within a fixed range. Corollary 3 below follows. We first define $SCR(\alpha, \beta)$ and $SSR(\alpha, \beta)$.

PROBLEM: $SCR(\alpha, \beta)$.

INPUT: Two parameters $0 \leq \alpha < \beta \leq \pi$, a set C of polygonal chains whose segmental links have slopes between α and β , and a set R of upwards rays.

OUTPUT: A minimum cardinality subset R' of R so that for each $c \in C$ there exists $r \in R'$ for which $r \cap c \neq \emptyset$.

PROBLEM: $SSR(\alpha, \beta)$.

INPUT: Two parameters $0 \leq \alpha < \beta \leq \pi$, a set S of line segments with slopes between α and β , and a set R of upwards rays.

OUTPUT: A minimum cardinality subset R' of R so that for each $s \in S$ there exists $r \in R'$ for which $r \cap s \neq \emptyset$.

Corollary 3 *For any $0 \leq \alpha < \beta \leq \pi$, $SCR(\alpha, \beta)$ and $SSR(\alpha, \beta)$ are APX-hard.*

4 Fixed parameter intractability

The concept of parameterized complexity was introduced by Downey and Fellows [3] (see also [5, 8]). An instance of a parameterized problem is a pair (I, k) , where k is a parameter; the complexity of the problem is measured not only with respect to the input size, but also with respect to the parameter k . A parameterized problem is said to be *fixed-parameter tractable* (FPT) with respect to the parameter k if there exists an algorithm for the problem with time complexity $f(k) \cdot p(|I|)$ for some computable function f and some polynomial p . Downey and Fellows [3] also introduced a theory of *parameterized intractability*; i.e., a hierarchy of complexity classes called the *W-hierarchy*. Some of the classes in this hierarchy are interrelated as follows: $FPT = W[0] \subseteq W[1] \subseteq W[2] \subseteq W[3] \dots$. Generally speaking, a parameterized problem with parameter k is in the class $W[i]$ if it is reducible to a circuit of height i or less, which assigns 1 to at most k inputs. It is most-likely that all above inclusion are strict, namely, there are problems in $W[1]$ that are most likely fixed-parameter intractable and, in particular, that $W[1]$ -hard problems are fixed-parameter intractable [3, 5, 8].

In this section we observe that SCR is $W[2]$ -complete and demonstrate that SSR is $W[1]$ -hard.

The reduction in the proof of Theorem 1 shows that SCR is exactly the same as Hitting Set. Thus SCR is $W[2]$ -complete since Hitting Set is $W[2]$ -complete [3, 5]. Unfortunately, this is not necessarily true for SSR as the reduction in this case is from 2-HITTING-SET. In general, d -HITTING-SET (i.e., when the size of each subset is at most a constant d) is FPT. Therefore the reduction from VERTEX-COVER does not provide any $W[k]$ -hardness. We show that SSR is $W[1]$ -hard via a reduction from the problem of *stabbing 2-intervals² with points* (S2I) (defined below) which is known to be $W[1]$ -hard [2] (note: S2I is equivalent to the 2-C1P-SET-COVER problem in [2]). We consider all problems to be parameterized with respect to the size of the solution.

PROBLEM: S2I.

INPUT: A collection $2I$ of 2-intervals and a set P of points on the line.

OUTPUT: A minimum cardinality subset P' of P so that for each 2-interval $I \in 2I$, $I = (I_1, I_2)$ there exists a point $p \in P'$ for which $p \in I_1$ or $p \in I_2$.

Theorem 4 *SSR is $W[1]$ -hard.*

Proof. We show S2I is reducible to SSR with similar parameters; since the former is $W[1]$ -hard [2], the desired result follows. Let $2I$ be a collection of 2-intervals, and P a set of points on the line. We construct sets S of line segments and R of upwards rays so that a minimum

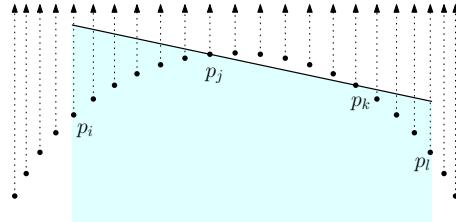


Figure 3: A segment corresponding to a 2-interval $([i, j], [k, l])$. The segment lies above exactly those points that are contained in one of its two intervals.

subset P' of P that collectively stabs all 2-intervals in $2I$ corresponds to a minimum subset of R that stabs all segments in S and vice versa.

Notice that we may assume that the endpoints of intervals in $2I$ as well as the points in P are integers. Moreover, we may also assume that the endpoints of all intervals in $2I$ are taken from P (otherwise an interval with an endpoint which is not in P can be shortened). We embed the points in P on the upper half of a circle in the same left to right order in which they appear on the line. Starting with an empty set S , for each 2-interval $I = ([i, j], [k, l])$ with $i \leq j \leq k \leq l$, add $s(I)$ to S where $s(I)$ denotes the line segment through the points p_j and p_k whose left and right endpoints are, respectively, above p_i and p_l . Finally, let R be the set of upward rays with origins at the points of P ; see Figure 3 for an illustration. It is easy to see that the line segment $s(I)$ where $I = ([i, j], [k, l])$ lies above exactly those points of P that lie in the union of its two intervals. Thus, the rays which can be chosen to stab I are precisely the ones corresponding to points that belong to I . That is, a minimum subset of P that stabs $2I$ corresponds to a minimum subset of R that stabs S (of same cardinality) and vice versa. \square

5 Concluding Remarks

The paper shows that SCR is hard to approximate within a logarithmic factor while SSR is hard to approximate within a constant factor. A first natural open question we raise is either to strengthen the hardness of approximation or to present an approximation algorithm better than the standard $O(\log n)$ -approximation algorithm for HITTING-SET. In particular, we hope that the geometric structure that we have observed may be useful for this purpose.

We showed that SCR is $W[2]$ -complete and that SSR is $W[1]$ -hard. However, it is still open whether SSR is in $W[1]$, $W[2]$ -complete, or in $W[2]$ but not $W[2]$ -complete.

²A 2-interval is a pair of intervals on the line.

References

- [1] T.M. Chan and E. Grant. Exact Algorithms and APX-Hardness Results for Geometric Packing and Covering Problems. *Computational Geometry* (2012), to appear.
- [2] M. Dom, M. Fellows, and F. Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. In Proc. *Workshop on Algorithms and Computation (WALCOM)*, LNCS 5341: 298–309, 2009.
- [3] R.G. Downey and M.R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87: 161–187, 1992.
- [4] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4): 634–652, July 1998.
- [5] J. Flum and M. Grohe. Parameterized complexity theory. *Texts in theoretical computer science*, Springer, 2006.
- [6] J. Håstad. Some optimal inapproximability results. *J. ACM* 48(4): 798–859, 2001.
- [7] M.J. Katz, J.S.B. Mitchell, and Y. Nir. Orthogonal segment stabbing. *Computational Geometry*, 30(2): 197–205, 2005.
- [8] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

Heaviest Induced Ancestors and Longest Common Substrings

Travis Gagie*

Pawel Gawrychowski†

Yakov Nekrich‡

Abstract

Suppose we have two trees on the same set of leaves, in which nodes are weighted such that children are heavier than their parents. We say a node from the first tree and a node from the second tree are induced together if they have a common leaf descendant. In this paper we describe data structures that efficiently support the following heaviest-induced-ancestor query: given a node from the first tree and a node from the second tree, find an induced pair of their ancestors with maximum combined weight. Our solutions are based on a geometric interpretation that enables us to find heaviest induced ancestors using range queries. We then show how to use these results to build an LZ-compressed index with which we can quickly find with high probability the longest substring common to the indexed string and a given pattern.

1 Introduction

In their paper “Range Searching over Tree Cross Products”, Buchsbaum, Goodrich and Westbrook [4] considered how, given a forest of trees T_1, \dots, T_d and a subset E of the cross product of the trees’ node sets, we can preprocess the trees such that later, given a d -tuple u consisting of one node from each tree, we can, e.g., quickly determine whether there is any d -tuple $e \in E$ that *induces* u — i.e., such that every node in e is a descendant of the corresponding node in u . (Unfortunately, some of their work was later found to be faulty; see [1].)

In this paper we assume we have two trees T_1 and T_2 on the same set of n leaves, in which each internal node has at least two children and nodes are weighted such that children are heavier than their parents. We assume E is the identity relation on the leaves. Following Buchsbaum et al., we say a node in T_1 and a node in T_2 are *induced* together if they have a common leaf descendant. We consider how, given a node v_1 in T_1 and a node v_2 in T_2 , we can quickly find a pair of their *heaviest induced ancestors* (HIAs) — i.e., an ancestor u_1 of v_1 and ancestors u_2 of v_2 such that u_1 and u_2 are

induced together and have maximum combined weight.

In Section 2 we give several tradeoffs for data structures supporting HIA queries: e.g., we describe an $\mathcal{O}(n)$ -space data structure with $\mathcal{O}(\log^3 n (\log \log n)^2)$ query time. Our motivation is the problem of building LZ-compressed indexes with which we can quickly find a longest common substring (LCS) of the indexed string and a given pattern. Tree cross products and LZ-indexes may seem unrelated, until we compare figures from Buchsbaum et al.’s paper and Kretz and Navarro’s “On Compressing and Indexing Repetitive Sequences”, shown in Figure 1. In Section 3 we show how, given a string S of length N whose LZ77 parse [23] consists of n phrases, we can build an $\mathcal{O}(n \log N)$ -space index with which, given a pattern P of length m , we can find with high probability an LCS of P and S in $\mathcal{O}(m \log^2 n)$ time.

2 Heaviest Induced Ancestors

An obvious way to support HIA queries is to impose orderings on T_1 and T_2 ; for each node u , store u ’s weight and the numbers of leaves to the left of u ’s leftmost and rightmost leaf descendants; and store a range-emptiness data structure for the $n \times n$ grid on which there is a marker at point (x, y) if the x -th leaf from the left in T_1 is the y -th leaf from the left in T_2 . Suppose there are $x_1 - 1$ and $x_2 - 1$ leaves to the left of the leftmost and rightmost leaf descendants of u_1 in T_1 , and $y_1 - 1$ and $y_2 - 1$ leaves to the left of the leftmost and rightmost leaf descendants of u_2 in T_2 . Then u_1 and u_2 are induced together if and only if the range $[x_1..x_2] \times [y_1..y_2]$ is non-empty. Chan, Larsen and Pătrașcu [5] showed how we can store the range-emptiness data structure in $\mathcal{O}(n)$ space with $\mathcal{O}(\log^\epsilon n)$ query time, or in $\mathcal{O}(n \log \log n)$ space with $\mathcal{O}(\log \log n)$ query time.

Given a node v_1 in T_1 and v_2 in T_2 , we start with a pointer p to v_1 and a pointer q to the root of T_2 . If the nodes u_1 and u_2 indicated by p and q are induced together, then we check whether u_1 and u_2 have greater combined weight than any induced pair we have seen so far and move q down one level toward v_2 ; otherwise, we move p up one level toward the root of T_1 ; we stop when p reaches the root of T_1 or q reaches v_2 . This takes a total of $\mathcal{O}(\text{depth}(v_1) + \text{depth}(v_2))$ range-emptiness queries.

*University of Helsinki and HIIT

†Max-Planck-Institut für Informatik

‡University of Kansas

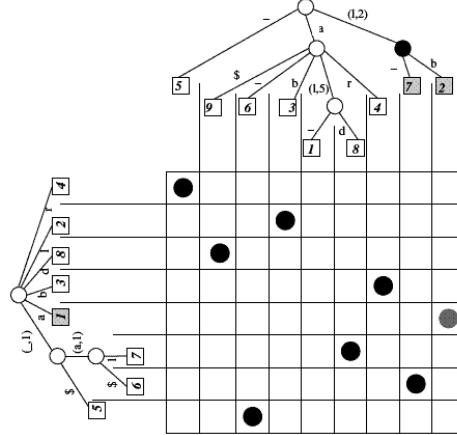
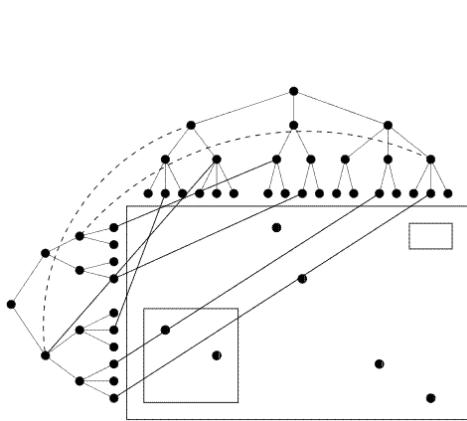


Figure 1: Figure 1 from Buchsbaum et al.’s “Range Searching over Tree Cross Products” and Figure 2b from Kreft and Navarro’s “On Compressing and Indexing Repetitive Sequences”, whose similarity suggests a link between the two problems. We exploit this link when we use HIA queries to implement LCS queries.

2.1 An $\mathcal{O}(n \log^2 n)$ -space data structure with $\mathcal{O}(\log n \log \log n)$ query time

We now describe an $\mathcal{O}(n \log^2 n)$ -space data structure with $\mathcal{O}(\log n \log \log n)$ query time; later we will show how to reduce the space via sampling, at the cost of increasing the query time. We first compute the heavy-path decompositions [20] of T_1 and T_2 . These decompositions have the property that every root-to-leaf path consists of the prefixes of $\mathcal{O}(\log n)$ heavy paths. Therefore, for each leaf w there are $\mathcal{O}(\log^2 n)$ pairs (a, b) such that a and b are the lowest nodes in their heavy paths in T_1 and T_2 , respectively, that are ancestors of w .

For each pair of heavy paths, one in T_1 and the other in T_2 , we store a list containing each pair (a, b) such that a is a node in the first path, b is a node in the second path, a and b are induced together by some leaf, a ’s child in the first path is not induced with b by any leaf, and b ’s child in the second path is not induced with a by any leaf. We call this the paths’ *skyline list*. Since there are n leaves and $\mathcal{O}(\log^2 n)$ pairs for each leaf, all the skyline lists have total length $\mathcal{O}(n \log^2 n)$. We store a perfect hash table containing the non-empty lists.

Let $L = (a_1, b_1), \dots, (a_\ell, b_\ell)$ be the skyline list for a pair of heavy paths, sorted such that $\text{depth}(a_1) > \dots > \text{depth}(a_\ell)$ and $\text{weight}(a_1) > \dots > \text{weight}(a_\ell)$ or, equivalently, $\text{depth}(b_1) < \dots < \text{depth}(b_\ell)$ and $\text{weight}(b_1) < \dots < \text{weight}(b_\ell)$. (Notice that, if a is induced with b , then every ancestor of a is also induced with b . Therefore, if (a_i, b_i) and (a_j, b_j) are both pairs in L and a_i is deeper than a_j then, by our definition of a pair in a skyline list, b_j must be deeper than b_i .) Let v_1 be a node in the first path and v_2 be a node in the second path. Suppose we want to find the pair of induced ancestors in these paths of v_1 and v_2 with maximum combined

weight. With the approach described above, we would start with a pointer p to v_1 and a pointer q to the highest node in the second path, then move p up toward the highest node in the first path and q down toward v_2 .

A geometric visualization is shown in Figure 2: the filled markers (from right to left) have coordinates $(\text{weight}(a_1), \text{weight}(b_1)), \dots, (\text{weight}(a_\ell), \text{weight}(b_\ell))$, the hollow marker has coordinates $(\text{weight}(v_1), \text{weight}(v_2))$, and we seek the point (x, y) that is dominated both by some filled marker and by the hollow marker, such that $x + y$ is maximized. Notice $(\text{weight}(a_1), \text{weight}(b_1)), \dots, (\text{weight}(a_\ell), \text{weight}(b_\ell))$ is a skyline — i.e., no marker dominates any other marker. There are five cases to consider: neither v_1 nor v_2 are induced with any other nodes in the paths; v_1 is induced with some node in the second path, but v_2 is not induced with any node in the first path; v_1 is not induced with any node in the second path, but v_2 is induced with some node in the first path; both v_1 and v_2 are induced with some nodes in the paths, but not with each other; and v_1 and v_2 are induced together.

It follows that finding the pair of induced ancestors in these paths of v_1 and v_2 with maximum combined weight is equivalent to finding the interval $(a_i, b_i), \dots, (a_j, b_j)$ in L such that $\text{depth}(a_{i-1}) > \text{depth}(v_1) \geq \text{depth}(a_i)$ and $\text{depth}(b_j) \leq \text{depth}(v_2) < \text{depth}(b_{j+1})$, then finding the maximum in

$$\begin{aligned} \text{weight}(v_1) &+ \text{weight}(b_{i-1}), \\ \text{weight}(a_i) &+ \text{weight}(b_i), \\ \text{weight}(a_{i+1}) &+ \text{weight}(b_{i+1}), \\ &\vdots \\ \text{weight}(a_{j-1}) &+ \text{weight}(b_{j-1}), \\ \text{weight}(a_j) &+ \text{weight}(b_j), \\ \text{weight}(a_{j+1}) &+ \text{weight}(v_2). \end{aligned}$$

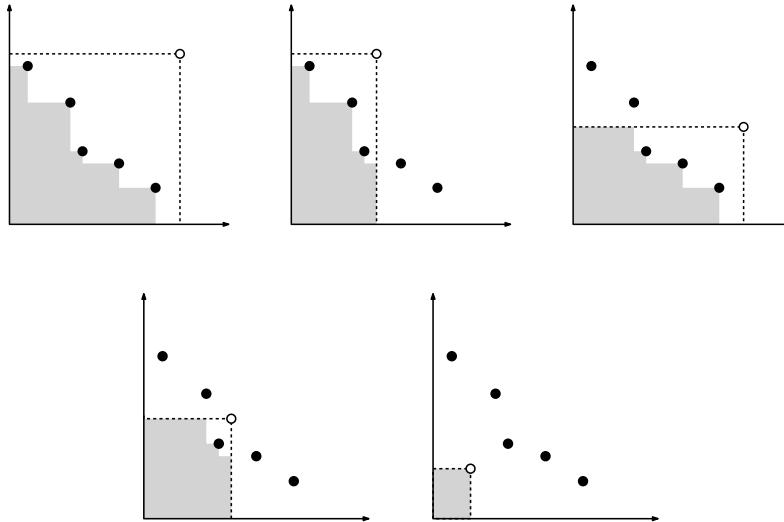


Figure 2: Finding the pair of induced ancestors of v_1 and v_2 with maximum combined weight is equivalent to storing a skyline such that, given a query point, we can quickly find the point (x, y) dominated both by some point on the skyline and by the query point, such that $x + y$ is maximized.

Therefore, if we store $\mathcal{O}(\ell)$ -space predecessor data structures with $\mathcal{O}(\log \log n)$ query time [21] for $\text{depth}(a_1), \dots, \text{depth}(a_\ell)$ and $\text{depth}(b_1), \dots, \text{depth}(b_\ell)$ and an $\mathcal{O}(\ell)$ -space range-maximum data with $\mathcal{O}(1)$ query time [8] for $\text{weight}(a_1) + \text{weight}(b_1), \dots, \text{weight}(a_\ell) + \text{weight}(b_\ell)$, then in $\mathcal{O}(\log \log n)$ time we can find the pair of induced ancestors in these paths of v_1 and v_2 with maximum combined weight. Notice that we can assign v_1 and v_2 different weights when finding this pair of induced ancestors; this will be useful in Section 3.

Lemma 1 *We can store T_1 and T_2 in $\mathcal{O}(n \log^2 n)$ space such that, given nodes v_1 in T_1 and v_2 in T_2 , in $\mathcal{O}(\log \log n)$ time we can find a pair of their induced ancestors in the same heavy paths with maximum combined weight, if such a pair exists.*

To find a pair of HIAs of v_1 and v_2 , we consider the heavy-path decompositions of T_1 and T_2 as trees \mathbf{T}_1 and \mathbf{T}_2 of height $\mathcal{O}(\log n)$ in which each node is a heavy path and V is a child of U in \mathbf{T}_1 or \mathbf{T}_2 if the highest node in the path V is a child of a node in the path U in T_1 or T_2 . We start with a pointer \mathbf{p} to the path V_1 containing v_1 and a pointer \mathbf{q} to the root of \mathbf{T}_2 . If the skyline list for the nodes U_1 and U_2 indicated by \mathbf{p} and \mathbf{q} is non-empty, then we apply Lemma 1 to the deepest ancestors of v_1 and v_2 in U_1 and U_2 , check whether the induced ancestors we find have greater combined weight than any induced pair we have seen so far and move \mathbf{q} down one level toward V_2 (to execute the descent efficiently, in the very beginning we generate the whole path from V_2 containing v_2 to the root of \mathbf{T}_2); otherwise, we move \mathbf{p} up one level toward the root of \mathbf{T}_1 . This takes a total

of $\mathcal{O}(\log n \log \log n)$ time. Again, we have the option of assigning v_1 and v_2 different weights for the purpose of the query.

Theorem 2 *We can store T_1 and T_2 in $\mathcal{O}(n \log^2 n)$ space such that, given nodes v_1 in T_1 and v_2 in T_2 , in $\mathcal{O}(\log n \log \log n)$ time we can find a pair of their HIAs.*

In the full version of this paper we will reduce the query time in Theorem 2 to $\mathcal{O}(\log n)$ via fractional cascading [6]; however, this is not straightforward, as we need to modify our approach such that predecessor searches keep the same target as we change pairs of heavy paths and the hive or catalogue graph has bounded degree.

2.2 An $\mathcal{O}(n \log n)$ -space data structure with $\mathcal{O}(\log^2 n)$ query time

To reduce the space bound in Theorem 2 to $\mathcal{O}(n \log n)$, we choose the orderings to impose on T_1 and T_2 such that each heavy path consists either entirely of leftmost children or entirely of rightmost children (except possibly for the highest nodes). We store an $\mathcal{O}(n \log^\epsilon n)$ -space data structure [2] that supports $\mathcal{O}(\log \log n + k)$ -time range-reporting queries on the grid described at the beginning of this section, where k is the number of points reported.

Notice that, if u_1 is an ancestor of w_1 in the same heavy path in T_1 and u_2 is an ancestor of w_2 in the same heavy path in T_2 , then we can use a range-reporting query to find, e.g., the leaves that induce u_1 and u_2 together but not w_1 and w_2 together. Suppose there are $x_1 - 1$ and $x_2 - 1 > x_1 - 1$ leaves to the left of the

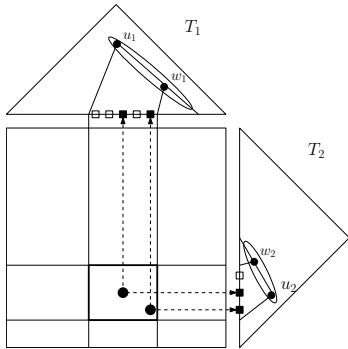


Figure 3: Suppose u_1 is an ancestor of w_1 in the same heavy path (shown as an oval) in T_1 and u_2 is an ancestor of w_2 in the same heavy path (also shown as an oval) in T_2 . We can use a range-reporting query to find the leaves (shown as filled boxes) that induce u_1 and u_2 together but not w_1 and w_2 together.

leftmost leaf descendants of u_1 and w_1 in T_1 , and $y_1 - 1$ and $y_2 - 1 > y_2 - 1$ leaves to the left of the rightmost leaf descendants of w_2 and u_2 in T_2 ; the cases when $x_2 < x_1$ or $y_2 < y_1$ are symmetric. Then the leaves that induce u_1 and u_2 together but not w_1 and w_2 together are indicated by markers in $[x_1..x_2 - 1] \times [y_1..y_2 - 1]$, as illustrated in Figure 3. That is, we query the cross product of the ranges of leaves in the subtrees of u_1 and u_2 but not w_1 and w_2 . Similarly, we can find the leaves that induce u_1 and w_2 together but not u_2 and w_1 together (or vice versa), but then we query the cross product of the ranges of leaves in the subtrees of u_1 and w_2 but not w_1 (or of u_2 and w_1 but not w_2).

For each pair of heavy paths, we build a list containing each pair (a, b) such that, for some leaf x , a is the lowest ancestor of x in the first path and b is the lowest ancestor of x in the second path. We call this the paths' *extended list*, and consider it in decreasing order by the depth of the first component. Notice that an extended list is a supersequence of the the corresponding skyline list, but all the extended lists together still have total length $\mathcal{O}(n \log^2 n)$.

We do not store the complete extended lists; instead, we sample only every $(\log n)$ -th pair, so the sampled lists take $\mathcal{O}(n \log n)$ space. We store a perfect hash function containing the non-empty sampled lists; we can still tell if a list was empty before sampling by using a range-reporting query to find any common leaf descendants of the highest nodes in the heavy paths. Given two consecutive sampled pairs from an extended list, in $\mathcal{O}(\log n)$ time we can recover the pairs between them using a range-reporting query, as described above.

With each sampled pair from an extended list, we store the preceding and succeeding pairs (possibly unsampled) that also belong to the corresponding skyline

list; recall that the extended list is a supersequence of the skyline list. This gives us an irregular sampling (which may include duplicates) of pairs from the skyline lists, which has total size $\mathcal{O}(n \log n)$. Instead of storing predecessor and range-maximum data structures over the complete skyline lists, we store them over these sampled skyline lists, so we use a total of $\mathcal{O}(n \log n)$ space. Since these data structures are over sampled skyline lists, querying them indicates only which $(\log n)$ -length block in a complete extended list contain the pair that would be returned by a query on a corresponding complete skyline list. We can recover any $(\log n)$ -length block of a complete extended list in $\mathcal{O}(\log n)$ time with a range-reporting query, however, and then scan that block to find the pair with maximum combined weight.

If we sample only every $(\log^2 n)$ -th pair from each extended list and use Chan et al.'s linear-space data structure for range reporting, then we obtain an even smaller (albeit slower) data structure for HIA queries.

Theorem 3 *We can store T_1 and T_2 in $\mathcal{O}(n \log n)$ space such that, given nodes v_1 in T_1 and v_2 in T_2 , in $\mathcal{O}(\log^2 n)$ time we can find a pair of their HIAs. Alternatively, we can store T_1 and T_2 in $\mathcal{O}(n)$ space such that, given v_1 and v_2 , in $\mathcal{O}(\log^{3+\epsilon} n)$ time we can find a pair of their HIAs.*

3 Longest Common Substrings

LZ-compressed indexes can use much less space than compressed suffix arrays or FM-indexes (see [3, 13, 14, 17]) when the indexed string is highly repetitive (e.g., versioned text documents, software repositories or databases of genomes of individuals from the same species). Although there is an extensive literature on the LCS problem, including Weiner's classic paper [22] on suffix trees and more recent algorithms for inputs compressed with the Burrows-Wheeler Transform (see [18]) or grammars (see [15]), we do not know of any grammar- or LZ-compressed indexes designed to support fast LCS queries.

Most LZ-compressed indexes are based on an idea by Kärkkäinen and Ukkonen [11]: we store a data structure supporting access to the indexed string $S[1..N]$; we store one Patricia tree [16] T_{rev} for the reversed phrases in the LZ parse, and another T_{suf} for the suffixes starting at phrase boundaries; we store a data structure for 4-sided range reporting for the grid on which there is a marker at point (x, y) if the x -th phrase in right-to-left lexicographic order is followed in the parse by the lexicographically y -th suffix starting at a phrase boundary; and we store a data structure for 2-sided range reporting for the grid on which there is a marker at point (x, y) if a phrase source begins at position x and ends at position y .

Given a pattern $P[1..m]$, for $1 \leq i \leq m$ we search for $(P[1..i])^{\text{rev}}$ in T_{rev} (where the superscript rev indicates that a string is reversed) and for $P[i+1..m]$ in T_{suf} ; access S to check that the path labels of the nodes where the searches terminate really are prefixed by $(P[1..i])^{\text{rev}}$ and $P[i+1..m]$; find the ranges of leaves that are descendants of those nodes; and perform a 4-sided range-reporting query on the cross product of those ranges. This gives us the locations of occurrences of P in S that touch phrase boundaries. We then use recursive 2-sided range-reporting queries to find the phrase sources covering the occurrences we have found so far.

Rytter [19] showed how, if the LZ77 parse of S consists of n phrases, then we can build a balanced straight-line program (BSLP) for S with $\mathcal{O}(n \log N)$ rules. A BSLP for S is a context-free grammar in Chomsky normal form that generates S and only S such that, in the parse tree of S , every node's height is logarithmic in the size of its subtree. We showed in a previous paper [9, 10] how we can store a BSLP for S in $\mathcal{O}(n \log N)$ space such that extracting a substring of length m from around a phrase boundary takes $\mathcal{O}(m)$ time. Using this data structure for access to S and choosing the rest of the data structures appropriately, we can store S in $\mathcal{O}(n \log N)$ space such that listing all the occ occurrences of P in S takes $\mathcal{O}(m^2 + \text{occ} \log \log N)$ time.

Our solution can easily be modified to find the LCS of P and S in $\mathcal{O}(m^2 \log \log n)$ time: we store the BSLP for S ; the two Patricia trees T_{rev} and T_{suf} , with the nodes weighted by the lengths of their path labels; and an instance of Chan et al.'s $\mathcal{O}(n \log \log n)$ -space range-emptiness data structure with $\mathcal{O}(\log \log n)$ query time, instead of the data structure for 4-sided range reporting. All these data structures together take a total of $\mathcal{O}(n \log N)$ space. By the definition of the LZ77 parse, the first occurrence of every substring in S touches a phrase boundary. It follows that we can find the LCS of P and S by finding, for $1 \leq i \leq m$, values h and j such that some phrase ends with $P[h..i]$ and the next phrase starts with $P[i+1..j]$ and $j - h + 1$ is maximum.

For $1 \leq i \leq m$ we search for $(P[1..i])^{\text{rev}}$ in T_{rev} and for $P[i+1..m]$ in T_{suf} , as before; access S to find the longest common prefix (LCP) of $(P[1..i])^{\text{rev}}$ and the path label of the node where the search in T_{rev} terminates, and the LCP of $P[i+1..m]$ and the path label of the node where the search in T_{suf} terminates; take v_1 and v_2 to be the loci of those LCPs, and treat them as having weights equal to the lengths of the LCPs; and then use the range-emptiness data structure and the simple HIA algorithm described at the beginning of Section 2 to find h and j for this choice of i . For each choice of i this takes $\mathcal{O}(m \log \log n)$ time, so we use $\mathcal{O}(m^2 \log \log n)$ time in total.

Lemma 4 *We can store S in $\mathcal{O}(n \log N)$ space such that, given a pattern P of length m , we can find the*

LCS of P and S in $\mathcal{O}(m^2 \log \log n)$ time.

We now show how to use our data structure for HIA queries to reduce the dependence on m in Lemma 4 from quadratic to linear.

Ferragina [7] showed how, by storing path labels' Karp-Rabin hashes [12] and rebalancing the Patricia trees via centroid decompositions, in a total of $\mathcal{O}(m \log n)$ time we can find with high probability the nodes where the searches for $(P[1..i])^{\text{rev}}$ and $P[i+1..m]$ terminate, for all choices of i . In our previous paper we showed how, by storing the hash of the expansion of each non-terminal in the BSLP for S , in $\mathcal{O}(m \log m)$ time we can then verify with high probability that the path labels of the nodes where the searches terminate really are prefixed by $(P[1..i])^{\text{rev}}$ and $P[i+1..m]$.

Using the same techniques, in $\mathcal{O}(m \log m)$ time we can find with high probability for all choices of i , the LCP of $(P[1..i])^{\text{rev}}$ and any reversed phrase, and the LCP of $P[i+1..m]$ and any suffix starting at a phrase boundary. If $m = n^{\mathcal{O}(1)}$ then $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$. If $m = n^{\omega(1)}$, then we can preprocess P and batch the searches for the LCPs, to perform them all in $\mathcal{O}(m)$ time.

More specifically, to find the LCPs of $P[2..m], \dots, P[m..m]$ and suffixes starting at phrase boundaries, we first build the suffix array and LCP array of P . For $1 \leq i \leq m$ we use Ferragina's data structure to find the suffix starting at a phrase boundary whose LCP with $P[i+1..m]$ is maximum. For each phrase boundary, we use the suffix array and LCP array of P to build a Patricia tree for the suffixes of P whose LCPs we will seek at that phrase boundary. We then balance these Patricia trees via centroid decompositions. For each phrase boundary, we determine the length of the LCP of any suffix of P and the suffix starting at that phrase boundary. We then use the LCP array of P to find the LCPs of $P[2..m], \dots, P[m..m]$ and suffixes starting at phrase boundaries. This takes a total of $\mathcal{O}(m)$ time. Finding the LCPs of $P[1], (P[1..2])^{\text{rev}}, \dots, P^{\text{rev}}$ is symmetric.

Suppose we already know the LCPs of $P[1], (P[1..2])^{\text{rev}}, \dots, P^{\text{rev}}$ and the reversed phrases, and the LCPs of $P[2..m], \dots, P[m]$ and the suffixes starting at phrase boundaries. Then in a total of $\mathcal{O}(m \log^2 n)$ time we can find with high probability values h and j such that some phrase ends with $P[h..i]$ and the next phrase starts with $P[i+1..j]$ and $j - h + 1$ is maximum, for each choice of i . To do this, we use m applications of Theorem 3 to T_{rev} and T_{suf} , one for each partition of P into a prefix and a suffix. This gives us the following result:

Theorem 5 *Let S be a string of length N whose LZ77 parse consists of n phrases. We can store S in $\mathcal{O}(n \log N)$ space such that, given a pattern P of length*

m , we can find with high probability a longest substring common to P and S in $\mathcal{O}(m \log^2 n)$ time.

We can reduce the time bound in Theorem 5 to $\mathcal{O}(m \log n \log \log n)$ at the cost of increasing the space bound to $\mathcal{O}(n(\log N + \log^2 n))$, by using the data structure from Theorem 2 instead of the one from Theorem 3. In fact, as we noted in Section 2, in the full version of this paper we will also eliminate the $\log \log n$ factor here.

References

- [1] A. Amir, G. M. Landau, M. Lewenstein and D. Sokol. Dynamic text and static pattern matching. *ACM Transactions on Algorithms*, 3(2), 2007.
- [2] S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proc. Symposium on Foundations of Computer Science*, pages 198–207, 2000.
- [3] D. Arroyuelo, G. Navarro, and K. Sadakane. Stronger Lempel-Ziv based compressed text indexing. *Algorithmica*, 62(1–2):54–101, 2012.
- [4] A. L. Buchsbaum, M. T. Goodrich, and J. Westbrook. Range searching over tree cross products. In *Proc. European Symposium on Algorithms*, pages 120–131, 2000.
- [5] T. M. Chan, K. G. Larsen, and M. Pătrașcu. Orthogonal range searching on the RAM, revisited. In *Proc. Symposium on Computational Geometry*, pages 1–10, 2011.
- [6] B. Chazelle and L. J. Guibas. Fractional cascading. *Algorithmica*, 1(2):133–191, 1986.
- [7] P. Ferragina. On the weak prefix-search problem. *Theoretical Computer Science*, 483:75–84, 2013.
- [8] J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.
- [9] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi. A faster grammar-based self-index. In *Proc. Conference on Language and Automata Theory and Applications*, pages 240–251, 2012.
- [10] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi. A faster grammar-based self-index. Technical Report 1109.3954v6, arxiv.org, 2012.
- [11] J. Kärkkäinen and E. Ukkonen. Lempel-Ziv parsing and sublinear-size index structures for string matching. In *Proc. South American Workshop on String Processing*, pages 141–155, 1996.
- [12] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.
- [13] S. Kreft and G. Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115–133, 2013.
- [14] V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.
- [15] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theoretical Computer Science*, 410(8–10):900–913, 2009.
- [16] D. R. Morrison. PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968.
- [17] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007.
- [18] E. Ohlebusch, S. Gog, and A. Kügel. Computing matching statistics and maximal exact matches on compressed full-text indexes. In *Proc. Symposium on String Processing and Information Retrieval*, pages 347–358, 2010.
- [19] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003.
- [20] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *Proc. Symposium on Theory of Computing*, pages 114–122, 1981.
- [21] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.
- [22] P. Weiner. Linear pattern matching algorithms. In *Proc. Symposium on Switching and Automata Theory*, pages 1–11, 1973.
- [23] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3), 1977.

Maximum-Weight Planar Boxes in $O(n^2)$ Time (and Better)

Jérémie Barbay*

Timothy M. Chan†

Gonzalo Navarro‡

Pablo Pérez-Lantero§

Abstract

Given a set P of n points in \mathbb{R}^d , where each point p of P is associated with a weight $w(p)$ (positive or negative), the MAXIMUM-WEIGHT Box problem consists in finding an axis-aligned box B maximizing $\sum_{p \in B \cap P} w(p)$. We describe algorithms for this problem in two dimensions that run in the worst case in $O(n^2)$ time, and much less on more specific classes of instances. In particular, these results imply similar ones for the MAXIMUM BICHROMATIC DISCREPANCY Box problem. These improve by a factor of $\Theta(\log n)$ on the best worst-case complexity previously known for these problems, $O(n^2 \lg n)$ [Cortés et al., J. Alg., 2009; Dobkin et al., J. Comput. Syst. Sci., 1996].

1 Introduction

Consider a set P of n points in \mathbb{R}^d , such that the points are in general position (i.e., no pair of points share the same x or y coordinate). Each point p of P is assigned a weight $w(p) \in \mathbb{R}$ that can be either positive or negative. For any subset $B \subseteq \mathbb{R}^d$ let $W(B) := \sum_{p \in B \cap P} w(p)$. A *box* is an axis-aligned hyper-rectangle, and we say that the *weight* of a box B is $W(B)$. We consider the MAXIMUM-WEIGHT Box problem, which given P and $w()$ consists in finding a box B with maximum weight $W(B)$.

Related work. In one dimension, the coordinates of the points matter only in the order they induce on their weights, and the problem reduces to the MAXIMUM-SUM CONSECUTIVE SUBSEQUENCE problem [3], which can be solved in $O(n)$ time if the coordinates are already sorted. Cortés et al. [6] solved the dynamic version of this problem supporting updates of weights

for a fixed point set. They described a data structure called MCS-tree, which supports in $O(\lg n)$ time both updates and MAXIMUM-SUM CONSECUTIVE SUBSEQUENCE queries on any interval of the sequence of points. The MAXIMUM-WEIGHT Box problem in two dimensions was introduced by Cortés et al. [6], who gave an algorithm running in $O(n^2 \lg n)$ time within $O(n)$ space. Their algorithm is based on MCS-trees: they reduce any instance of the MAXIMUM-WEIGHT Box problem in two dimensions to $O(n^2)$ instances of the problem in one dimension, each solved dynamically in $O(\lg n)$ time by using the MCS-tree.

We consider the MAXIMUM-WEIGHT Box problem in two dimensions on a set P of n weighted points, such that no pair of points share the same x or y coordinate.

Basic definitions. A *strip* is the area delimited by two lines parallel to the same axis. Given the point set P , we say that a strip S is *monochromatic* if $S \cap P$ is not empty and the weights of all elements of $S \cap P$ have the same sign. A monochromatic strip S is *positive* (resp. *negative*) if S contains points of P with positive (resp. negative) weights. We say that P is composed of δ strips if P can be covered by δ pairwise disjoint monochromatic strips of alternating signs. Given any bounded set $S \subset \mathbb{R}^2$, let $\text{Box}(S)$ denote the smallest box covering S .

Results. We present the following results for the MAXIMUM-WEIGHT Box problem in two dimensions. All our algorithms use space linear in the number of input points. Over instances composed of n weighted points, our general algorithm runs in $O(n^2)$ time (Theorem 2). Although this result can be deduced from new results on the KLEE’S MEASURE problem [5], it is a more direct and simplified (non-trivial) solution, which further provides smaller running times on specific classes on instances. Namely, if the point set P is composed of $\delta \in [1..n]$ (either horizontal or vertical) strips, our algorithm executes adaptively in $\text{SORT}(n) + O(\delta n) \subset O(n \lg n + \delta n) \subset O(n^2)$ time (Theorem 4), where $\text{SORT}(n)$ is the time required to sort the elements of P by their x -coordinates and by their y -coordinates ($O(n \lg n)$ in the Comparison Model, $O(n \lg \lg n)$ in the RAM model, and $O(n \sqrt{\lg \lg n})$ with randomization in the RAM model if the coordinates of the points are integer numbers [10]).

*Department of Computer Science, University of Chile, Chile. jbarbay@dcc.uchile.cl. Partially supported by grant CONICYT, FONDECYT/Regular 1120054, Chile.

†Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. tmchan@cs.uwaterloo.ca.

‡Department of Computer Science, University of Chile, Chile. gnavarro@dcc.uchile.cl. Partially funded by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F, Mideplan, Chile.

§Escuela de Ingeniería Civil en Informática, Universidad de Valparaíso, Chile. pablo.perez@uv.cl. Partially supported by grant CONICYT, FONDECYT/Iniciación 11110069, Chile.

Applications to other known problems. Let P be a set of n planar points, each being colored either red or blue.

The MAXIMUM BICHROMATIC DISCREPANCY Box problem [6, 7] consists in finding a box that maximizes the absolute difference between the numbers of red and blue points that it contains, and was solved in $O(n^2 \lg n)$ time by Dobkin et al. [7]. Any instance of this problem can be reduced to two particular instances of the MAXIMUM-WEIGHT Box problem [6]. In the first one red points have weight +1 and blue points weight -1 , and conversely in the second one. Then our results can be applied and imply an $O(n^2)$ worst-case time algorithm for the MAXIMUM BICHROMATIC DISCREPANCY Box problem, improving upon previous $O(n^2 \lg n)$ -time algorithms [6, 7].

The MAXIMUM Box problem [6, 8, 11] consists in finding a box B containing the maximum number of blue points and no red point. Eckstein et al. [8] introduced it in general dimension, proving that if the dimension d of the points is part of the input then the problem is NP-hard. In two dimensions it was later solved in $O(n^2 \lg n)$ time by Liu and Nediak [11]. In 2010 Backer et al. [1] showed that the MAXIMUM Box problem in two dimensions can be solved in $O(n \lg^3 n)$ time and $O(n \lg n)$ space, and that for any fixed dimension $d \geq 3$ it can be solved in time within $O(n^d \lg^{d-2} n)$.

Any instance of the MAXIMUM Box problem is equivalent to a particular case of the MAXIMUM-WEIGHT Box problem in which blue points have weight +1 and red points have weight $-\infty$ [6]. Then our techniques can be applied and imply $O(n^2)$ worst-case time algorithms for this problem. While this time complexity is worse than the best known solution [1], it requires only linear space, which in some cases can be an important advantage over the $O(n \lg n)$ space required by Backer et al.'s solution [1].

Note that our specialized results are faster on some classes of instances which arise naturally in applications, such as instances where one needs to find a maximum box over an imbalanced red-blue dataset in data mining and/or data analysis [8, 9, 13]. Generally, if the ratio of the number of blue points over the number of red points is within $\omega(1)$, then our techniques yield a running time within $\text{SORT}(n) + o(n^2)$ on an instance of n points.

Outline. In Section 2 we describe the general $O(n^2)$ -time algorithm. In Section 3 we obtain the adaptive algorithm running in $\text{SORT}(n) + O(\delta n)$ time, where δ is the number of strips of the point set. Finally, in Section 4, we discuss further adaptive results, a connection to KLEE'S MEASURE problem, potential polylogarithmic-factor speedups, and open problems.

2 Quadratic worst-case time algorithm

Assume the elements of P are sorted twice, first by x -coordinates and second by y -coordinates, in $\text{SORT}(n)$ time.

We say that $X \subseteq P$ is a *box set* if X is the intersection of P with some box. For any box set $X \subseteq P$ we define the *score* of X , $\mathbf{S}(X)$, as the following four boxes (see Figure 1). Let $[x_1, x_2] \times [y_1, y_2] := \text{Box}(X)$:

- (1) $\text{Box}(X)$;
- (2) a maximum-weight box $\mathbf{B}_L(X) \subseteq \text{Box}(X)$ of X of the form $[x_1, x] \times [y_1, y_2]$, $x_1 \leq x \leq x_2$;
- (3) a maximum-weight box $\mathbf{B}_R(X) \subseteq \text{Box}(X)$ of X of the form $[x, x_2] \times [y_1, y_2]$, $x_1 \leq x \leq x_2$; and
- (4) a maximum-weight box $\mathbf{B}_0(X) \subseteq \text{Box}(X)$ of X of the form $[x, x'] \times [y_1, y_2]$, $x_1 \leq x \leq x' \leq x_2$.

For each of these boxes we keep only two opposed vertices defining it and its weight, so representing a box set X by $\mathbf{S}(X) := (\text{Box}(X), \mathbf{B}_L(X), \mathbf{B}_R(X), \mathbf{B}_0(X))$ requires only constant space.

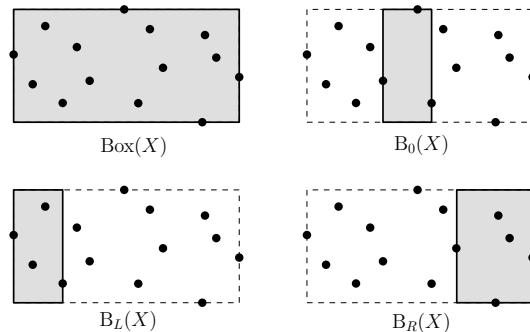


Figure 1: The score $\mathbf{S}(X) = (\text{Box}(X), \mathbf{B}_L(X), \mathbf{B}_R(X), \mathbf{B}_0(X))$ of a box set $X \subseteq P$.

We say that a box set $X \subseteq P$ is *scored* if $\mathbf{S}(X)$ is computed, and we use $\text{Box}(X)$ to represent X instead of X itself. Let the operator $\oplus : 2^P \times 2^P \rightarrow 2^P$ be defined over all pairs (X_1, X_2) of scored box sets of P such that: X_1 and X_2 can be separated with a vertical line, X_1 is to the left of X_2 , and $X_1 \cup X_2$ is a box set. Then $X_1 \oplus X_2$ returns the scored set $X_1 \cup X_2$, and it can be computed in $O(1)$ time from the next observations:

$$W(X_1 \cup X_2) = W(X_1) + W(X_2).$$

$$W(\mathbf{B}_L(X_1 \cup X_2)) = \max \begin{cases} W(\mathbf{B}_L(X_1)) \\ W(X_1) + W(\mathbf{B}_L(X_2)) \end{cases}$$

$$W(\mathbf{B}_R(X_1 \cup X_2)) = \max \begin{cases} W(\mathbf{B}_R(X_2)) \\ W(X_2) + W(\mathbf{B}_R(X_1)) \end{cases}$$

$$W(\mathbb{B}_0(X_1 \cup X_2)) = \max \begin{cases} W(\mathbb{B}_0(X_1)) \\ W(\mathbb{B}_0(X_2)) \\ W(\mathbb{B}_R(X_1)) + W(\mathbb{B}_L(X_2)). \end{cases}$$

Notice that by applying the operators \oplus to singletons $\{p\}$ over all points p of P in left-to-right order, we can compute $\mathbb{B}_0(P)$, i.e., the maximum-weight vertical strip, in $O(n)$ time. After projection to the x -axis, this immediately gives a linear-time algorithm for the MAXIMUM-SUM CONSECUTIVE SUBSEQUENCE problem, studied by Bentley [3] and often taught in undergraduate algorithms classes.

Let \mathcal{S} be a horizontal strip such that exactly m points of P are not in \mathcal{S} . The vertical lines passing through the m points of $P \setminus \mathcal{S}$ split \mathcal{S} into $m+1$ boxes denoted $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{m+1}$ from left to right. Let B be a box of maximum weight that has its top side above \mathcal{S} and its bottom side below \mathcal{S} . Suppose that the left and right sides of B intersect \mathcal{S}_i and \mathcal{S}_j ($1 \leq i \leq j \leq m+1$), respectively. If $i \neq j$, then $W(B \cap \mathcal{S}_i)$ and $W(B \cap \mathcal{S}_j)$ are precisely $W(\mathbb{B}_R(P \cap \mathcal{S}_i))$ and $W(\mathbb{B}_L(P \cap \mathcal{S}_j))$, respectively (see Figure 2). Therefore we have $W(B) = W(\mathbb{B}_R(P \cap \mathcal{S}_i)) + \sum_{t=i+1}^{j-1} W(\mathcal{S}_t) + W(\mathbb{B}_L(P \cap \mathcal{S}_j)) + W(B \setminus \mathcal{S})$. On the other hand, if $i = j$, then $W(B)$ equals $W(\mathbb{B}_0(P \cap \mathcal{S}_i))$.

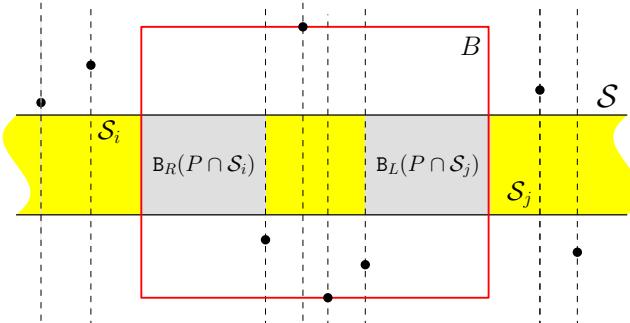


Figure 2: The strip \mathcal{S} is partitioned into $m+1$ boxes $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{m+1}$ by the vertical lines passing through the m points in $P \setminus \mathcal{S}$. If the left and right sides of an optimal box B cross \mathcal{S}_i and \mathcal{S}_j , respectively, then they are determined by $\mathbb{B}_R(P \cap \mathcal{S}_i)$ and $\mathbb{B}_L(P \cap \mathcal{S}_j)$.

Consider the following STRIP-CONSTRAINED MAXIMUM-WEIGHT Box problem: Let \mathcal{P} be a weighted point set and \mathcal{S} be a horizontal strip so that: $\mathcal{P} \setminus \mathcal{S}$ consists of n points already sorted from left to right; \mathcal{S} splits $\mathcal{P} \setminus \mathcal{S}$ into two halves; the vertical lines through the points of $\mathcal{P} \setminus \mathcal{S}$ split \mathcal{S} into the boxes $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{n+1}$ from left to right; and the points of $\mathcal{P} \cap \mathcal{S}$ are summarized by the scored box sets $\mathcal{P} \cap \mathcal{S}_1, \dots, \mathcal{P} \cap \mathcal{S}_{n+1}$. Find a maximum-weight box of \mathcal{P} , with the top side above \mathcal{S} and the bottom side below \mathcal{S} .

The key to our new solution is an $O(n^2)$ -time algorithm for this constrained problem, using an approach which may be nick-named “divide-summarize-and-conquer”.

Lemma 1 *The STRIP-CONSTRAINED MAXIMUM-WEIGHT BOX problem admits a solution in $O(n^2)$ time and $O(n)$ space.*

Proof. Let $F(n)$ denote the time required to solve a given instance of the STRIP-CONSTRAINED MAXIMUM-WEIGHT BOX problem over n points. We apply divide-and-conquer: Split the points of \mathcal{P} above (resp. below) \mathcal{S} into two halves with a horizontal line ℓ_1 (resp. ℓ_2). Let P_1 denote the points above ℓ_1 , P_2 denote the points in between ℓ_1 and \mathcal{S} , P_3 denote the points in between \mathcal{S} and ℓ_2 , and P_4 denote the points below ℓ_2 . Then the problem can be reduced to the next four subproblems:

- (1) the points of $P_1 \cup P_4$ outside a strip \mathcal{S}' covering $P_2 \cup P_3 \cup \mathcal{S}$;
- (2) the points of $P_1 \cup P_3$ outside a strip \mathcal{S}' covering $P_2 \cup \mathcal{S}$;
- (3) the points of $P_2 \cup P_3$ outside the strip $\mathcal{S}' = \mathcal{S}$; and
- (4) the points of $P_2 \cup P_4$ outside a strip \mathcal{S}' covering $P_3 \cup \mathcal{S}$.

The reduction to subproblem (1) can be done in $O(n)$ time as follows: Take each point p of $P_2 \cup P_3$ and compute the score $S(\{p\})$. Simulate the merging of the left-to-right orders of $P_1 \cup P_4$, $P_2 \cup P_3$, and $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{n+1}$ (each of which can be obtained in $O(n)$ time) to compute the corresponding scored box sets in the new strip \mathcal{S}' . This computation can be done by applying the operator \oplus to successive scored box sets in between consecutive points of $P_1 \cup P_4$ in the left-to-right order. The reductions to subproblems (2)–(4) are similar.

The base case occurs when $n \in \{1, 2\}$. In the most general setting ($n = 2$) we have one point p_1 above \mathcal{S} and one point p_2 below \mathcal{S} , defining boxes $\mathcal{S}_1, \mathcal{S}_2$, and \mathcal{S}_3 on \mathcal{S} . Assume w.l.o.g. that p_1 is to the left of p_2 and $w(p_1), w(p_2) > 0$ (for example, if $w(p_1) < 0$, we can eliminate p_1). Then the solution is $\mathbb{B}_0((\mathcal{P} \cap \mathcal{S}_1) \cup \{p_1\} \cup (\mathcal{P} \cap \mathcal{S}_2) \cup \{p_2\} \cup (\mathcal{P} \cap \mathcal{S}_3))$, which can be computed in constant time by applying the \oplus operator.

This yields the recurrence

$$F(n) \in 4F(n/2) + O(n),$$

where $F(1) \in O(1)$. Then $F(n) \in O(n^2)$. As for the space $G(n)$, since the four subproblems are solved independently one after the other, the recurrence is $G(n) \in G(n/2) + O(n)$, whose solution is within $O(n)$. \square

The reduction from the original MAXIMUM-WEIGHT Box problem to the constrained problem follows from a more straightforward divide-and-conquer:

Theorem 2 *The MAXIMUM-WEIGHT Box problem admits a solution in $O(n^2)$ time and $O(n)$ space on instances of n points.*

Proof. We first sort the points of P by their x -coordinates in $\text{SORT}(n)$ time and then apply a recursive procedure, whose time over n weighted points will be $T(n)$. The recursion applies divide-and-conquer as follows: Draw a horizontal strip \mathcal{S} (a line) splitting P into two halves P_1 and P_2 , where P_1 is above \mathcal{S} and P_2 is below \mathcal{S} . Then we can find a maximum-weight box B_1 for P_1 , a maximum-weight box B_2 for P_2 , and a maximum-weight box $B_{1,2}$ for $P_1 \cup P_2$ restricted to intersect \mathcal{S} . Then the box among B_1 , B_2 , and $B_{1,2}$ maximizing $W()$ is the solution. To compute $B_{1,2}$ we will use the solution for the STRIP-CONSTRAINED MAXIMUM-WEIGHT BOX problem over P and \mathcal{S} , for which we split \mathcal{S} into $n+1$ empty scored boxes $\mathcal{S}_1, \dots, \mathcal{S}_n$ according to all the x -coordinates of P . This requires $O(n)$ time and then Lemma 1 allows us to compute $B_{1,2}$ in $O(n^2)$ time and $O(n)$ space. Since B_1 and B_2 are computed recursively, the time complexity is

$$T(n) \in 2T(n/2) + O(n^2),$$

where $T(1) \in O(1)$. Hence $T(n) \in O(n^2)$. As for the space $S(n)$, the three subproblems are solved independently one after the other, and thus it holds $S(n) \in \max\{S(n/2), S(n/2), O(n)\} \subseteq O(n)$. \square

3 δ -sensitive analysis

Assume that P is composed of $\delta \in [1..n]$ strips, and suppose w.l.o.g. that these strips are horizontal. These strips can be identified in $O(n)$ time from the sorting of the points in P by their y -coordinates. One does not need to consider boxes whose horizontal sides are in the middle of some of these strips: there always exists an optimal box such that each horizontal side is aligned with an edge of some strip; specifically, the top (resp. bottom) of an optimal box will align with a positive point at the top (resp. bottom) of a positive strip. Using this observation we refine the results of Section 2.

Lemma 3 *The STRIP-CONSTRAINED MAXIMUM-WEIGHT Box problem admits a solution in $O(\delta n)$ time and $O(n)$ space if the points of \mathcal{P} above (resp. below) \mathcal{S} are composed of $\delta/2$ strips.*

Proof. Let $F(n, \delta)$ denote the time required to solve the problem. We modify the divide-and-conquer algorithm from the proof of Lemma 1 as follows: We split the points above \mathcal{S} with a horizontal line ℓ_1 and the points below \mathcal{S} with a horizontal line ℓ_2 , and define P_1, \dots, P_4 as before. However, we choose ℓ_1 and ℓ_2 differently, not to ensure that each P_i has $n/4$ points, but to ensure

that each P_i is composed of $\delta/4$ strips. Let n_i denote the size of P_i (so that $n_1 + n_2 + n_3 + n_4 = n$).

The base case arises when there is at most one strip above (resp. below) \mathcal{S} , and can be solved as follows: Assume w.l.o.g. that the weights of these at most two strips are positive (if one of the strips has all negative weights, we can eliminate all of its points). Then the solution is $B_0(P)$, which can be computed by applying the operator \oplus to the sequence, arranged in left-to-right order, consisting of $\mathcal{P} \cap \mathcal{S}_1, \dots, \mathcal{P} \cap \mathcal{S}_{n+1}$ together with singletons $\{p_i\}$ over all p_i in $\mathcal{P} \setminus \mathcal{S}$. The base case then requires $O(n)$ time.

The recurrence is now modified to the following:

$$\begin{aligned} F(n, \delta) &\in F(n_1 + n_3, \delta/2) + F(n_1 + n_4, \delta/2) \\ &\quad + F(n_2 + n_3, \delta/2) + F(n_2 + n_4, \delta/2) \\ &\quad + O(n). \end{aligned}$$

where $F(n, 1) \in O(n)$. Observe that the recursion tree for $F(n, \delta)$ has at most $\lg \delta$ levels, and that in the i -th level the computation time besides recursive calls is $O(2^i n)$. Then $F(n, \delta) \in O(\delta n)$. The space is within $O(n)$ as in Theorem 2. \square

Theorem 4 *The MAXIMUM-WEIGHT Box problem admits a solution in $\text{SORT}(n) + O(\delta n)$ time and $O(n)$ space on instances of n points composed of δ strips.*

Proof. Let $T(n, \delta)$ denote the time required to solve the MAXIMUM-WEIGHT Box problem over n points composed of δ strips. We apply divide-and-conquer as in Theorem 2, but selecting strip \mathcal{S} such that both resulting sets P_1 and P_2 are composed of $\delta/2$ strips, and n_1 points and n_2 points respectively. If there is only $\delta = 1$ strip then the solution is either empty (if the strip is negative) or all the points (if it is positive), so in the base case it holds $T(n, 1) \in O(n)$. In the recursive case we have:

$$\begin{aligned} T(n, \delta) &= T(n_1, \delta/2) + T(n_2, \delta/2) + F(n, \delta) \\ &\in T(n_1, \delta/2) + T(n_2, \delta/2) + O(\delta n). \end{aligned}$$

The recursion tree of $T(n, \delta)$ has at most $\lg \delta$ levels and in the i -th level the computation time besides recursive calls is $O(\delta n/2^i)$, and thus $T(n, \delta) \in O(\delta n)$. Again, the space is $O(n)$ as before. \square

Some naturally occurring instances will have a low number of strips. For example, instances with an unbalanced number of positive and negative points are due to contain few strips. The following corollary captures this observation.

Corollary 5 *Let n_+ and n_- be the number of points with positive and negative weight of an instance of $n = n_+ + n_-$ points, respectively. Then the MAXIMUM-WEIGHT Box problem admits a solution in $\text{SORT}(n) + O(\min\{n_+, n_-\} \cdot n)$ time.*

Proof. Observe that $\delta \leq 2 \min\{n_+, n_-\} + 1$ and apply Theorem 4. \square

4 Discussion

Improved adaptive results. Our $SORT(n) + O(\delta n)$ time algorithm adapts well to instances where points associated with weights of same sign can be clustered into a small number of vertical or horizontal strips. It improves a previous $O(\delta n \lg(n/\delta))$ result [2].

To obtain better adaptive algorithms, we can consider more general clusterings into rectangles. One approach is as follows: Call (C_1, \dots, C_k) a *cluster partition* of P if $\{C_1, \dots, C_k\}$ is a partition of P and in every axis the orthogonal projections of $\text{Box}(C_1), \dots, \text{Box}(C_k)$ are pairwise disjoint.

Given a single rectangular cluster, the optimal box of the whole instance can intersect the cluster boundaries in 10 distinct ways. Considering the top, bottom, left or right edges of the cluster, an optimal box can either intersect none of them (1 case where the optimal box is strictly contained in the cluster), exactly two (4 cases where it contains exactly one of the corners of the cluster), exactly three of them (4 cases where it entirely contains exactly one cluster edge), or exactly four (1 case where it contains the whole cluster). Note that if a box intersects exactly one edge, or exactly two opposite edges (e.g., top and bottom), then there is a box of the same score which intersects no cluster boundaries, since by the definition of a cluster partition, there are no other points exactly above, below, to the left or to the right of the cluster.

In the extended version of this article we will show how, given a partition of the n points into k clusters of respective sizes n_1, \dots, n_k , one can compute the 10 optimal boxes (extending the 4 boxes Box , \mathcal{B}_L , \mathcal{B}_R , and \mathcal{B}_0 from Section 2) corresponding to the cases described above in time within $O(\sum_{i=1}^k n_i^2)$ and combine these results in time $O(k^2)$ to obtain the optimal box of the whole instance. This yields an $O(\sum_{i=1}^k n_i^2 + k^2)$ time algorithm. Finding an optimal cluster partition seems hard.

Connection to Klee's measure problem and higher dimensions. Our $O(n^2)$ worst-case time algorithm is actually a special case of a more general result for a problem related to the well known KLEE'S MEASURE problem (computing the volume of a union of n boxes).

In the D -dimensional WEIGHTED DEPTH problem, we are given a set of n weighted boxes in \mathbb{R}^D and we want a point $p \in \mathbb{R}^D$ that maximizes the *depth*, defined as the sum of the weights of the boxes that contain p . All known algorithms for KLEE'S MEASURE problem can be modified to solve the WEIGHTED DEPTH problem. In particular, Overmars and Yap's algorithm [12]

runs in $O(n^{D/2} \lg n)$ time, Chan's algorithm [4] runs in $O(n^{D/2} 2^{O(\lg^* n)})$ time, and a new simple algorithm by Chan [5] runs in $O(n^{D/2})$ time.

The following observation has not been noted before:

Observation 6 *The MAXIMUM-WEIGHT Box problem in any constant dimension d can be reduced to the WEIGHTED DEPTH problem in dimension $D = 2d$.*

Proof. Given a point set P in \mathbb{R}^d , we map each point $p = (a_1, \dots, a_d) \in P$ to a region R_p in \mathbb{R}^{2d} , consisting of all $2d$ -tuples $(x_1, \dots, x_d, x'_1, \dots, x'_d)$ such that p lies inside the box with opposite corners (x_1, \dots, x_d) and (x'_1, \dots, x'_d) ; in other words, $R_p = \{(x_1, \dots, x_d, x'_1, \dots, x'_d) \mid [(x_1 \leq a_1 \leq x'_1) \vee (x'_1 \leq a_1 \leq x_1)] \wedge \dots \wedge [(x_d \leq a_d \leq x'_d) \vee (x'_d \leq a_d \leq x_d)]\}$. We can decompose R_p into a constant number of boxes in \mathbb{R}^{2d} . The maximum-weight box for P corresponds to a point $(x_1, \dots, x_d, x'_1, \dots, x'_d)$ that has the maximum depth among these regions. \square

According to the above observation, our $O(n^2)$ result for the MAXIMUM-WEIGHT Box problem in two dimensions is thus not new, but can be deduced from Chan's latest result for the WEIGHTED DEPTH problem in $D = 4$ dimensions [5]. In fact, the $O(n^2)$ time algorithm presented in this paper is inspired by the algorithm in [5], which is also based on a “divide-summarize-and-conquer” approach. We feel that the algorithm here is nevertheless interesting, because it is a more direct solution, and can be viewed as a further simplification of [5], avoiding the need to work explicitly in 4-dimensional space. (Besides, our $O(n^2)$ time algorithm is a stepping stone towards our $SORT(n) + O(\delta n)$ time algorithm.)

The above observation also implies that the MAXIMUM-WEIGHT Box problem in d dimensions can be solved in $O(n^d)$ time by Chan's new algorithm. Previously, only an $O(n^{2d-2} \lg n)$ time bound was reported [6].

Polylogarithmic-factor speedups and applications. Chan [5] also showed how to further speed up his algorithm by a polylogarithmic factor for the WEIGHTED DEPTH problem, but only when the dimension is sufficiently large (in particular, not for $D = 4$).

However, in the unweighted case of the DEPTH problem, it is shown [4, 5] that polylogarithmic speedup is possible for any $D \geq 3$: the running time can be improved to $O((n^{D/2} / \lg^{D/2} n)(\lg \lg n)^{O(1)})$. This extends to the case where the weights are integers with absolute value bounded by $O(1)$, since we can replace a box with positive weight c by c copies of the box, and we can replace a box with negative weight $-c$ by c copies of its complement (which can be decomposed into a constant number of boxes).

In particular, we can thus solve the MAXIMUM-WEIGHT BOX problem for the case of +1 and -1 weights in $O((n^d / \lg^d n)(\lg \lg n)^{O(1)})$ time. The same bound thus follows for the MAXIMUM BICHROMATIC DISCREPANCY problem mentioned in the introduction. Previously, only an $O(n^2 \lg n)$ bound was known for $d = 2$ [6, 7]. Similarly, by straightforward changes to incorporate $-\infty$ weights, the MAXIMUM BOX problem can be solved in $O((n^d / \lg^d n)(\lg \lg n)^{O(1)})$ time, improving the previous $O(n^d \lg^{d-2} n)$ bound for $d \geq 3$ [1].

Lower bounds? We conjecture that $O(n^d)$ is the best possible time complexity for the MAXIMUM-WEIGHT BOX problem, ignoring polylogarithmic factors. Unconditional lower bounds are probably difficult to prove. If one could show a converse to Observation 6 (a reduction from some problem related to KLEE'S MEASURE problem in $2d$ dimensions to the MAXIMUM-WEIGHT BOX problem in d dimensions), that might provide evidence for the conjecture. We are only able to show the following:

Observation 7 *The WEIGHTED DEPTH problem in any constant dimension d can be reduced to the MAXIMUM-WEIGHT BOX problem in dimension d .*

Proof. We first reduce the WEIGHTED DEPTH problem to a special case of the WEIGHTED DEPTH problem where all the input boxes are “dominance” ranges of the form $(-\infty, b_1] \times \cdots \times (-\infty, b_d]$. To see this, for a given $i \in [1..d]$, we replace any input box $[a_1, b_1] \times \cdots \times [a_d, b_d]$ of weight w with two boxes: $[a_1, b_1] \times \cdots \times [a_{i-1}, b_{i-1}] \times (-\infty, b_i] \times [a_{i+1}, b_{i+1}] \times \cdots \times [a_d, b_d]$ of weight w , and $[a_1, b_1] \times \cdots \times [a_{i-1}, b_{i-1}] \times (-\infty, a_i] \times [a_{i+1}, b_{i+1}] \times \cdots \times [a_d, b_d]$ of weight $-w$. By repeating this for each $i \in [1..d]$, each original box is replaced with 2^d boxes of the desired special form.

Now, given an instance of this special case of the WEIGHTED DEPTH problem, we map each input box $b = (-\infty, b_1] \times \cdots \times (-\infty, b_d]$ to the point $p_b = (b_1, \dots, b_d)$, of the same weight. We have the obvious property that p_b lies inside the box $[x_1, \infty) \times \cdots \times [x_d, \infty)$ iff (x_1, \dots, x_d) lies inside b . We add an extra point at (∞, \dots, ∞) with weight M for a sufficiently large number M . The maximum-weight box containing the resulting point set must be of the form $[x_1, \infty) \times \cdots \times [x_d, \infty)$ because of this extra point, and so corresponds to a point of maximum depth of the given boxes. \square

The above observation implies the $W[1]$ -hardness of the MAXIMUM-WEIGHT BOX problem with respect to d , since KLEE'S MEASURE problem and the WEIGHTED DEPTH problem are $W[1]$ -hard [4]. It also implies the unlikelihood of an algorithm that runs faster than $n^{d/2}$ time with current knowledge about KLEE'S MEASURE problem.

References

- [1] J. Backer and J. Keil. The mono- and bichromatic empty rectangle and square problems in all dimensions. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN'10)*, pages 14–25, 2010.
- [2] J. Barbay, G. Navarro, and P. Pérez-Lantero. Adaptive techniques to find optimal planar boxes. In *Proceedings of the 24th Canadian Conference on Computational Geometry (CCCG'12)*, pages 71–76, 2012.
- [3] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, 1984.
- [4] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Comput. Geom.*, 43(3):243–250, 2010.
- [5] T. M. Chan. Klee's measure problem made easy. Submitted, https://cs.uwaterloo.ca/~tmchan/easyklee4_13.pdf, 2013.
- [6] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: A general approach. *J. Algorithms*, 64(2-3):79–88, 2009.
- [7] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *J. Comput. Syst. Sci.*, 52(3):453–470, 1996.
- [8] J. Eckstein, P. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Comput. Optim. App.*, 23(3):285–298, 2002.
- [9] X. Guo, Y. Yin, C. Dong, G. Yang, and G. Zhou. On the class imbalance problem. In *Proceedings of the Fourth International Conference on Natural Computation*, pages 192–201, 2008.
- [10] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In *Proceedings of the Thirty-Third IEEE Symposium on Foundations of Computer Science*, pages 135–144, 2012.
- [11] Y. Liu and M. Nediak. Planar case of the maximum box and related problems. In *Proceeding of the 15th Canadian Conference on Computational Geometry (CCCG'03)*, pages 14–18, 2003.
- [12] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.
- [13] S. Visa and A. Ralescu. Issues in mining imbalanced data sets - a review paper. In *Proceedings of the Sixteen Midwest Artificial Intelligence and Cognitive Science Conference*, pages 67–73, 2005.

An Optimal Algorithm Computing Edge-to-Edge Visibility in a Simple Polygon

Mikkel Abrahamsen*

Abstract

Let P be a given, simple polygon with n vertices. We present a new $O(n)$ -time algorithm to compute the visible part of one edge from another edge of P . The algorithm does not alter the input and only uses $O(1)$ variables and is therefore a constant-workspace algorithm. The algorithm can be used to make a constant-workspace algorithm for computing the weak visibility polygon from an edge in $O(mn)$ time, where m is the number of vertices of the resulting polygon, and a constant-workspace algorithm for computing a minimum link path between two points inside a simple polygon in $O(n^2)$ time.

1 Introduction

Much research has been done about visibility problems in the plane, see the book by Ghosh [8] for an overview of the most important problems and results.

Let \mathcal{P} be a simple polygon with vertices $v_0v_1 \dots v_{n-1}$ in counterclockwise (CCW) order, and let $v_n = v_0$. A point $q \in \mathcal{P}$ is said to be *visible* from v_jv_{j+1} if there exists a point $p \in v_jv_{j+1}$ such that the segment pq is contained in \mathcal{P} . In this paper we show how to compute the visible part of an edge v_iv_{i+1} from the edge v_jv_{j+1} . Without loss of generality, we assume that $j = 0$ for the rest of this paper. The algorithm uses $O(n)$ time and is therefore optimal. The input is given in read-only memory and only $O(1)$ variables are needed in the workspace, each consisting of $O(\log n)$ bits. Therefore, the algorithm is a *constant-workspace algorithm*.

The problem of computing visibility between two edges was first addressed by Toussaint [12], who gave a query algorithm deciding if two edges are visible to each other if a triangulation of \mathcal{P} is provided. Later, Avis et al. [4] described an $O(n)$ -time algorithm to compute the visible part of one edge from another which does not require a triangulation or other involved data structures, but uses $\Omega(n)$ variables in the workspace. De et al. [7] claimed to present an $O(n)$ -time algorithm using constant workspace. However, their algorithm has a fault, as we shall see.

One of the best-known constant-workspace algorithms for a geometric problem is *Jarvis' march* [10] for the computation of the convex hull of n points in the plane in $O(hn)$ time, where h is the number of points on the hull. Recently, Asano et al. [2], Asano et al. [3], and Barba et al. [5] gave constant-workspace algorithms solving many elementary tasks in planar computational geometry. The research presented in this paper is a part of a master's thesis [1], which contains more details and space-efficient solutions to some other planar visibility problems.

1.1 Notation and definitions

Given two points in the plane a and b , the line segment with endpoints a and b is written ab . Both endpoints are included in segment ab . If s is a line segment, the line containing s which is infinite in both directions is written \overleftrightarrow{s} . The *half line* \overrightarrow{ab} is a line infinite in one direction, starting at a and passing through b . The *right half plane* $RHP(ab)$ is the closed half plane with boundary \overleftrightarrow{ab} lying to the right of ab . The *left half plane* $LHP(ab)$ is just $RHP(ba)$.

If \mathcal{P} is a simple polygon, the boundary of \mathcal{P} is written $\partial\mathcal{P}$. Let $\mathcal{P}(p, q)$ for two points $p, q \in \partial\mathcal{P}$ be the set of points on $\partial\mathcal{P}$ we meet when traversing $\partial\mathcal{P}$ CCW from p to q , both included. A *chain* of \mathcal{P} is such a set $\mathcal{P}(p, q)$ for some points $p, q \in \partial\mathcal{P}$. We have the general position assumption that no three vertices of \mathcal{P} are collinear.

Consider the edge v_0v_1 of a simple polygon \mathcal{P} . A *ray* emanating from v_0v_1 is a segment pq where $p \in v_0v_1$ and pq is contained in \mathcal{P} . Thus, a point q is visible from v_0v_1 if and only if there exists a ray pq emanating from v_0v_1 . A *right support* of the ray pq is a reflex vertex v of \mathcal{P} such that $v \in pq$ and the edges meeting at v are both contained in $RHP(pq)$. A *left support* is defined analogously. Since no ray emanates from a point to the left of v_0 , we make the convention that v_0 is a left support of any ray v_0q . Likewise, v_1 is a right support of any ray v_1q . A *support* is a right support or a left support.

The edge v_iv_{i+1} is *totally facing* the edge v_jv_{j+1} if both of the points v_j and v_{j+1} are in $LHP(v_iv_{i+1})$. Notice that v_iv_{i+1} can be totally facing v_jv_{j+1} even though no point on v_jv_{j+1} is visible from v_iv_{i+1} . Edge v_iv_{i+1} is *partially facing* v_jv_{j+1} if exactly one of the points v_j

*Department of Computer Science, University of Copenhagen,
mikkel.abrahamsen@gmail.com

and v_{j+1} is in $\text{LHP}(v_iv_{i+1})$ and *not facing* v_jv_{j+1} if none of the points are in $\text{LHP}(v_iv_{i+1})$. We say that v_iv_{i+1} is *facing* v_jv_{j+1} if v_iv_{i+1} is partially or totally facing v_jv_{j+1} . It follows from the definitions that v_iv_{i+1} is either totally facing, partially facing or not facing v_jv_{j+1} . That gives $3^2 = 9$ different combinations of how v_iv_{i+1} is facing v_jv_{j+1} and how v_jv_{j+1} is facing v_iv_{i+1} . However, only 8 of the cases are possible when v_iv_{i+1} and v_jv_{j+1} are edges of a simple polygon, since they cannot both partially face each other. That would imply that they intersect each other properly. All of the remaining 8 cases are possible, see for instance the paper of Avis et al. [4].

2 Visibility Between Two Edges of a Polygon

2.1 Point-to-point and point-to-edge visibility

If the edge v_iv_{i+1} is not facing edge v_0v_1 , the only point on v_iv_{i+1} that can be visible from v_0v_1 is one of the endpoints v_i or v_{i+1} . Likewise, if v_0v_1 is not facing v_iv_{i+1} , the only point on v_0v_1 that can possibly see v_iv_{i+1} is one of the endpoints v_0 or v_1 by means of rays contained in $\text{RHP}(v_0v_1)$. In such cases, the problem of computing the visible part of v_iv_{i+1} is reduced to point-to-point and point-to-edge visibility.

Point-to-point visibility is the problem of, given two points a and b in \mathcal{P} , to determine if ab is contained in \mathcal{P} . That can easily be tested in $O(n)$ time using constant workspace by traversing all edges of $\partial\mathcal{P}$, seeing if $\partial\mathcal{P}$ crosses ab somewhere.

Point-to-edge visibility is the slightly more complicated task of computing the visible part of an edge from a point p . This can also be done using constant workspace and $O(n)$ time by traversing all edges of $\partial\mathcal{P}$ once while keeping track of the vertices shadowing the largest part of the edge in each of the ends [1].

We now turn our attention to the more interesting case of computing the visible part of v_iv_{i+1} from v_0v_1 if the edges are facing each other. We motivate the development of a new algorithm by giving a counterexample to the constant-workspace algorithm of De et al. [7].

2.2 Counterexample to the algorithm proposed by De et al. [7]

The textual description and the pseudocode in [7] do not agree. Figure 1 is an example of a polygon where the algorithm computes a wrong result in both cases. After *PASS1()*, the line segment L is still $p_{i+1}p_{j+1}$. After *PASS2()*, L is θp_{j+1} . The text says that *PASS3()* is to check if a vertex on $\mathcal{P}(p_{j+1}, p_i)$ is to the right of L . No one is, so the algorithm returns that the rightmost visible point on p_jp_{j+1} from p_ip_{i+1} is p_{j+1} , which is wrong. The pseudocode gives another definition of *PASS3()*, according to which we also check if a vertex

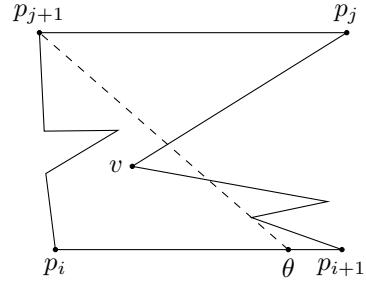


Figure 1: The algorithm from [7] reports the wrong visible part of p_jp_{j+1} from p_ip_{i+1} in this polygon.

on $\mathcal{P}(p_{i+1}, p_j)$ is to the left of L . Vertex v is, so the algorithm reports that nothing of p_jp_{j+1} is visible. That is clearly also wrong.

2.3 Computing visibility between edges facing each other

Assume for the rest of this section that the edges v_0v_1 and v_iv_{i+1} are facing each other and let $\square = \square v_0v_1v_iv_{i+1}$ be the quadrilateral with vertices $v_0v_1v_iv_{i+1}$ in that order. The possible rays from v_0v_1 to v_iv_{i+1} are all contained in \square , so when computing the visible part of v_iv_{i+1} , we are only concerned about the edges of \mathcal{P} that are (partially) in \square . A ray pq is a *proper ray* if $pq \subset \text{LHP}(v_0v_1)$ and $pq \subset \text{LHP}(v_iv_{i+1})$. An *improper ray* is a ray that is not proper. Each ray pq where p is an interior point on v_0v_1 and q is an interior point on v_iv_{i+1} is necessarily proper. Therefore, if pq is improper, $p = v_0$, $p = v_1$, $q = v_i$, or $q = v_{i+1}$. The visibility due to improper rays can be computed using point-to-edge visibility, so in this section, we focus on the visibility due to proper rays only. We leave out the proof of the following lemma due to limited space [1].

Lemma 1 *Let $v_R \in \mathcal{P}(v_1, v_i) \cap \square$ and $v_L \in \mathcal{P}(v_{i+1}, v_0) \cap \square$. Every proper ray pq from v_0v_1 to v_iv_{i+1} satisfies $p \in \text{LHP}(v_Rv_L)$ and $q \in \text{RHP}(v_Rv_L)$. In particular, if $v_0v_1 \cap \text{LHP}(v_Rv_L) = \emptyset$ or $v_iv_{i+1} \cap \text{RHP}(v_Rv_L) = \emptyset$, then no proper ray from v_0v_1 to v_iv_{i+1} exists.*

Assume that there are some proper rays from v_0v_1 to v_iv_{i+1} . We say that the ray pq is the *rightmost ray* from v_0v_1 to v_iv_{i+1} if p is as close to v_1 as possible and q is as close to v_{i+1} as possible among all proper rays. Similarly, pq is the *leftmost ray* from v_0v_1 to v_iv_{i+1} if p is as close to v_0 as possible and q is as close to v_i as possible. If v_0v_1 and v_iv_{i+1} are totally facing each other, all rays from v_0v_1 to v_iv_{i+1} are proper, so the visible part of v_iv_{i+1} is the points between the endpoints of the leftmost and rightmost rays. If one of the edges is only partially facing the other, the visible part of v_iv_{i+1}

can be computed using the leftmost and rightmost rays in combination with point-to-edge visibility.

If pq is a ray from v_0v_1 to v_iv_{i+1} , a *generalized left support* of pq is v_{i+1} if $q = v_{i+1}$ or a left support of pq otherwise. The following lemma characterizes rightmost rays by means of their supports.

Lemma 2 *Let pq be a proper ray from v_0v_1 to v_iv_{i+1} . The ray pq is a rightmost ray if and only if pq has a right support v_R and a generalized left support v_L and $v_L \in v_Rq$.*

Proof. If pq does not have any supports, we can choose p to be closer to v_1 , so pq is no rightmost ray. If pq has a generalized left support v_L , but no right support on $p v_L$, we can turn pq CCW around v_L to get a proper ray from a point closer to v_1 to a point closer to v_{i+1} . The same is true if pq has a right support v_R but no generalized left support on v_Rq .

It is also clear that if pq has a right support v_R and a generalized left support v_L , every other segment $p'q'$ from $p v_1$ to $q v_{i+1}$ crosses $\partial\mathcal{P}$, so $p'q'$ is no ray. Therefore, pq is a rightmost ray. \square

If the edges v_0v_1 and v_iv_{i+1} are totally facing each other and no edge obstructs the visibility between the edges, then the rightmost ray is $pq = v_1v_{i+1}$ and it has supports $v_L = v_1$ and $v_R = v_{i+1}$.

Algorithm 1 returns the indices (R, L) of the supports of the rightmost ray if it exists. The algorithm iteratively computes the correct value of R and L , taking the edges into consideration one by one. Initially, R is set to 1 and L is set to $i + 1$, as if no edges obstructed the visibility between the edges. The points p and q on v_0v_1 and v_iv_{i+1} , respectively, are always defined such that the segment pq contains v_R and v_L . The algorithm alternately traverses $\mathcal{P}(v_1, v_i)$ and $\mathcal{P}(v_{i+1}, v_n)$ one edge at a time using the index variables r and l . The variable $side$ is 1 when an edge in $\mathcal{P}(v_1, v_i)$ is traversed and -1 when an edge in $\mathcal{P}(v_{i+1}, v_n)$ is traversed. Each time an edge $v_{r-1}v_r$ or $v_{l-1}v_l$ is found that crosses pq , the value of R or L is updated to r or l , respectively. If the value of R is updated, we reset l to L , since it is possible that there are some edges on $\mathcal{P}(v_L, v_n)$ that did not intersect the old segment pq , but intersect the updated one. Likewise, when L is updated, we reset r to R . Although segment pq is changed when R or L is updated, $\mathcal{P}(v_1, v_R)$ or $\mathcal{P}(v_{i+1}, v_L)$ does not cross pq after the update. That is because pq is rotated clockwise (CW) away from the chains.

All our figures illustrate the case where v_0v_1 and v_iv_{i+1} are totally facing each other, but that assumption is not used in any of the proofs. If v_iv_{i+1} is partially facing v_0v_1 such that $v_0 \in \text{LHP}(v_iv_{i+1})$, then v_i might be the right support of the rightmost ray from v_0v_1 to v_iv_{i+1} . Likewise, if v_0v_1 is partially facing v_iv_{i+1} such

that $v_i \in \text{LHP}(v_0v_1)$, v_0 can be the left support of the rightmost ray.

Algorithm 1: FindRightmostRay(i)

Input: A polygon \mathcal{P} defined by its vertices v_0, v_1, \dots, v_{n-1} in CCW order and an index i such that v_0v_1 and v_iv_{i+1} are facing each other.
Output: If no proper ray from v_0v_1 to v_iv_{i+1} exists, NULL is returned. Otherwise, a pair of indices (R, L) is returned such that the rightmost ray from v_0v_1 to v_iv_{i+1} has right support v_R and generalized left support v_L .

```

1   $R \leftarrow 1, L \leftarrow i + 1$ 
2   $r \leftarrow R, l \leftarrow L$ 
3   $p \leftarrow v_1, q \leftarrow v_{i+1}$ 
4   $side \leftarrow 1$  (* 1 is right side, -1 is left side *)
5  while  $r < i$  or  $l < n$ 
6    if  $side = 1$ 
7      if  $r < i$ 
8         $r \leftarrow r + 1$ 
9        if  $v_r \in \text{LHP}(pq) \cap \square$ 
10          if  $v_{r-1}v_r$  intersects  $v_Lq$ 
11            return NULL
12           $R \leftarrow r, l \leftarrow L$ 
13    else (*  $side = -1$  *)
14      if  $l < n$ 
15         $l \leftarrow l + 1$ 
16        if  $v_l \in \text{RHP}(pq) \cap \square$ 
17          if  $v_{l-1}v_l$  intersects  $v_Rp$ 
18            return NULL
19           $L \leftarrow l, r \leftarrow R$ 
20  Let  $p$  be the intersection point between  $\overrightarrow{v_Lv_R}$  and  $v_0v_1$ 
21  Let  $q$  be the intersection point between  $\overrightarrow{v_Rv_L}$  and  $v_iv_{i+1}$ 
22  if  $p$  or  $q$  does not exist
23    return NULL
24   $side \leftarrow -side$ 
25  if  $pq \subset \text{LHP}(v_0v_1) \cap \text{LHP}(v_iv_{i+1})$ 
26    return  $(R, L)$ 
27  else
28    return NULL

```

Lemma 3 *Let R_j, L_j, p_j , and q_j be the values of R , L , p , and q , respectively, in the end of iteration j of the loop at line 5 in Algorithm 1, $j = 1, 2, \dots, k$. The initial values are R_0, L_0, p_0 , and q_0 . Then $R_{j-1} = R_j$ or $L_{j-1} = L_j$ for $j = 1, \dots, k$. $p_0, p_1, p_2, \dots, p_k$ is a sequence of points moving monotonically along v_0v_1 from*

v_1 towards v_0 . Likewise, $q_0, q_1, q_2, \dots, q_k$ is a sequence of points moving monotonically along v_iv_{i+1} from v_{i+1} towards v_i . Let a_j be the CW angle from $\overrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ to $\overrightarrow{v_Rv_L}$. Then $\sum_{j=1}^k a_j < 180^\circ$. In particular, $a_j < 180^\circ$ for each $j = 1, \dots, k$.

Proof. See Figure 2. It is clear that at most one of R and L changes in iteration j , since the lines 12 and 19 cannot both be executed. Therefore, $R_{j-1} = R_j$ or $L_{j-1} = L_j$. If R is redefined in iteration j , then $\overrightarrow{v_Rv_L}$ is rotating around v_L and the new value of R , namely R_j , satisfies $v_{R_j} \in \text{LHP}(v_{R_{j-1}}v_{L_{j-1}})$. Therefore, p_j is on the segment v_0p_{j-1} and q_j is on the segment $q_{j-1}v_i$. The same is true if L is updated. Hence, p_0, \dots, p_k is monotonically moving along v_0v_1 from v_1 towards v_0 and q_0, \dots, q_k is monotonically moving along v_iv_{i+1} from v_{i+1} towards v_i . Because of the monotonicity, the angles are additive, so that the CW angle from $\overrightarrow{v_{R_0}v_{L_0}}$ to $\overrightarrow{v_{R_k}v_{L_k}}$ is $\sum_{j=1}^k a_j$. If v_0v_1 is totally facing v_iv_{i+1} , every q_j is contained in $\text{LHP}(v_0v_1)$. Otherwise v_iv_{i+1} is totally facing v_0v_1 so that every p_j is contained in $\text{LHP}(v_iv_{i+1})$. In either case, $\sum_{j=1}^k a_j$ is bounded by 180° . That bound cannot be reached, since it would require that v_0v_1 or v_iv_{i+1} was infinitely long in both directions. \square

Theorem 4 Algorithm 1 works correctly.

Proof. First, consider the cases where the algorithm returns **NULL**. In line 11, we have found an intersection point x between $\mathcal{P}(v_R, v_i)$ and v_Lq . That means that $\mathcal{P}(v_1, v_i)$ intersects pq properly at x , since no three vertices are collinear. Lemma 1 gives that the only possible proper rays from v_0v_1 to v_iv_{i+1} are on the form $p'q'$, where $p' \in v_0p$ and $q' \in qv_i$. At the same time, if we use Lemma 1 with v_0v_1 and v_iv_{i+1} interchanged by each other and using x as ' v_L ' and v_L as ' v_R ', we get that $p'q'$ satisfies $p' \in pv_1$ and $q' \in v_{i+1}q$. Therefore, $p' = p$ and $q' = q$, but pq is no ray. Hence, there are no proper rays from v_0v_1 to v_iv_{i+1} . The case in line 18 is analogous.

Due to Lemma 3, we know that p is moving monotonically from v_1 towards v_0 and q is moving monotonically from v_{i+1} towards v_i . The case of line 23 happens if p has moved outside v_0v_1 , so that $v_0v_1 \cap \text{LHP}(v_Rv_L) = \emptyset$, or q has moved outside v_iv_{i+1} , so that $v_iv_{i+1} \cap \text{RHP}(v_Rv_L) = \emptyset$. In each of these cases, it follows from Lemma 1 that there are no proper rays from v_0v_1 to v_iv_{i+1} .

Now, assume that the algorithm returns (R, L) , but that pq is not a ray since some edge obstructs the visibility from p to q . Assume that $\mathcal{P}(v_1, v_i)$ intersects pq properly, and let x be the intersection point closest to p . $\mathcal{P}(v_1, v_i)$ enters $\text{LHP}(pq) \cap \square$ at x . Let y be the first point where $\mathcal{P}(x, v_i)$ leaves $\text{LHP}(pq) \cap \square$. Then $\mathcal{P}(x, y)$ is contained in $\text{LHP}(pq) \cap \square$ and $y \in xq$. We have two

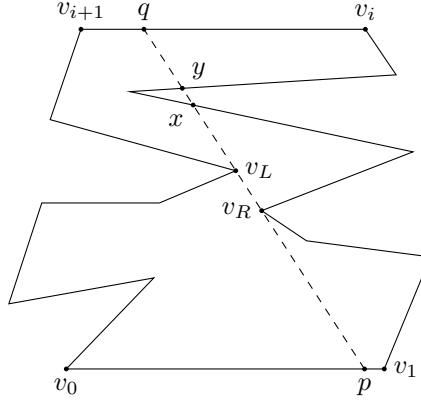


Figure 3: Case 2 in the proof of Theorem 4.

cases: $x \in \mathcal{P}(v_1, v_R)$ (case 1) and $x \in \mathcal{P}(v_R, v_i)$ (case 2). Assume that we are in case 2, see Figure 3. Assume that the final value of R is defined in a later iteration of the loop at line 5 than the final value of L . After R is defined in line 12, every edge $v_{r-1}v_r$ in $\mathcal{P}(v_R, v_i)$ is traversed and it is checked in line 10 if some edge intersects pq . In particular the edges in $\mathcal{P}(x, y)$ are traversed, in which case the algorithm either returns **NULL** or updates R , which is a contradiction. If R is defined in an earlier iteration than L , then r is set to R in line 19 when L is defined, and it is checked if some edge in $\mathcal{P}(v_R, v_i)$ intersects pq , so that cannot happen either.

Assume that we are in case 1, i.e. $x \in \mathcal{P}(v_1, v_R)$. Consider the first iteration, say iteration j , at the end of which $\mathcal{P}(v_1, v_R)$ intersects pq properly, and let x' be the intersection point closest to p . Let y' be the first point where $\mathcal{P}(x', v_i)$ leaves $\text{LHP}(pq) \cap \square$, such that $\mathcal{P}(x', y')$ is contained in $\text{LHP}(pq) \cap \square$ and $y' \in x'q$ (x' and y' might not be the same as x and y , since R and L can change before the algorithm terminates). We must have $v_R \in \mathcal{P}(y', v_i)$. There are three possible cases to consider: $v_R \in px'$ (case 1.1), $v_R \in x'y'$ (case 1.2), and $v_R \in y'q$ (case 1.3).

Assume case 1.3. Let R_k , L_k , p_k , and q_k be defined as in Lemma 3 for each iteration k . Either R or L is redefined in iteration j due to the minimality of j . Therefore, $R_{j-1} \neq R_j$ or $L_{j-1} \neq L_j$ (case 1.3.1 and 1.3.2, respectively). First, assume $R_{j-1} \neq R_j$ but $L_{j-1} = L_j$. Again, there are three cases to consider: $v_{R_{j-1}} \in \mathcal{P}(v_1, x')$ (case 1.3.1.1), $v_{R_{j-1}} \in \mathcal{P}(x', y')$ (case 1.3.1.2), and $v_{R_{j-1}} \in \mathcal{P}(y', v_{R_j})$ (case 1.3.1.3). Assume case 1.3.1.3, see Figure 4(a). According to Lemma 3, the CW angle between $\overrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ and $\overrightarrow{v_Rv_L}$ is less than 180° . Therefore, a subset of $\mathcal{P}(x', y')$ would also be contained in $\text{LHP}(v_{R_{j-1}}v_{L_{j-1}}) \cap \square$. That implies that $\mathcal{P}(v_1, v_{R_{j-1}})$ intersects $p_{j-1}q_{j-1}$, a contradiction because of the choice of j . $v_{R_{j-1}}$ cannot be in $\mathcal{P}(x', y')$ (case 1.3.1.2), because then v_{R_j} would be in $\text{RHP}(v_{R_{j-1}}v_{L_{j-1}})$, so R would not have been redefined

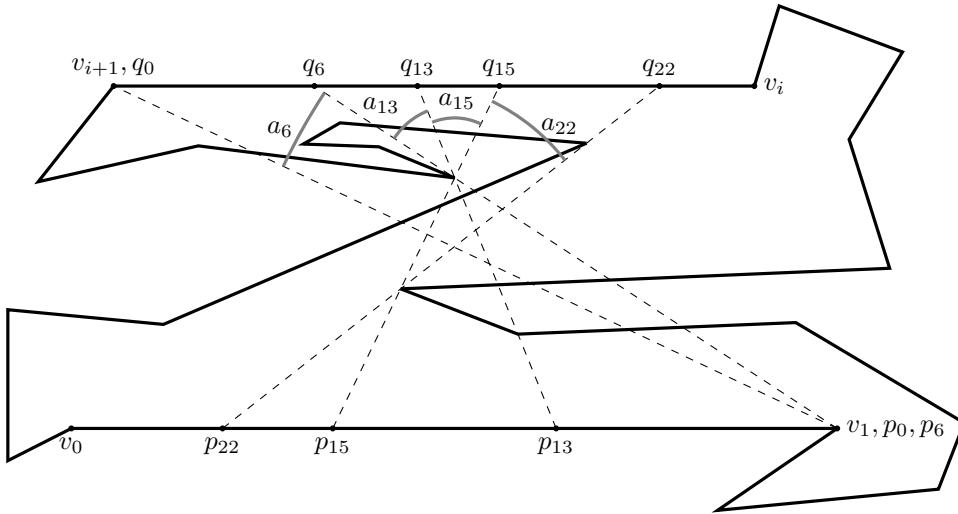


Figure 2: Illustration for Lemma 3. The points p_j, q_j are shown with the number j of the first iteration where they occur. The segments $p_j q_j$ are drawn dashed. The angles $a_j > 0$ are indicated with grey arcs.

to R_j in iteration j . Finally, if $v_{R_{j-1}}$ was a vertex in $\mathcal{P}(v_1, x')$ (case 1.3.1.1), $\mathcal{P}(x', y')$ would be contained in $\text{LHP}(v_{R_{j-1}} v_{L_{j-1}}) \cap \square$, and therefore v_R would be redefined to a vertex in $\mathcal{P}(x' y')$ when the edges of that chain were traversed. Hence, v_R would not be redefined to v_{R_j} in iteration j , which is a contradiction.

Now, assume $L_{j-1} \neq L_j$ (case 1.3.2), see Figure 4(b). The CW angle between $\overrightarrow{v_{R_{j-1}} v_{L_{j-1}}}$ and $\overrightarrow{v_{R_j} v_{L_j}}$ is less than 180° . Therefore, a part of $\mathcal{P}(x', y')$ is also in $\text{LHP}(v_{R_{j-1}} v_{L_{j-1}})$. That implies that $\mathcal{P}(v_1, v_{R_{j-1}})$ intersects $p_{j-1} q_{j-1}$, which contradicts the choice of j .

The case where $v_R \in x' y'$ (case 1.2) can be eliminated in a similar way. Consider case 1.1, i.e. $v_R \in px'$. The chain $\mathcal{P}(p, x')$ and the segment $x'p$ forms a simple, closed curve, because x' is the intersection point between $\mathcal{P}(v_1, v_i)$ and pq closest to p . The curve can, for instance, be seen in Figure 4(a). Consider the region of \mathcal{P} enclosed by the curve. In order to get to v_R , $\mathcal{P}(y', v_i)$ has to cross $x'p$ to get into the region. That contradicts that x' was the intersection point closest to p .

If we assume that $\mathcal{P}(v_{i+1}, v_n)$ intersects pq , we get a contradiction in an analogous way.

The test at line 25 is to ensure that pq is a proper ray, which is not always the case if $v_0 v_1$ is only partially facing $v_i v_{i+1}$. The conclusion is that if (R, L) is returned, v_R and v_L defines a proper ray pq with right support v_R and generalized left support v_L in that order. Therefore, pq must be the rightmost ray from $v_0 v_1$ to $v_i v_{i+1}$ according to Lemma 2. \square

Even though we reset l to L in line 12 or r to R in line 19, the running time is linear since the other variable is not reset, so half of the traversed are never traversed again, as the following theorem explains.

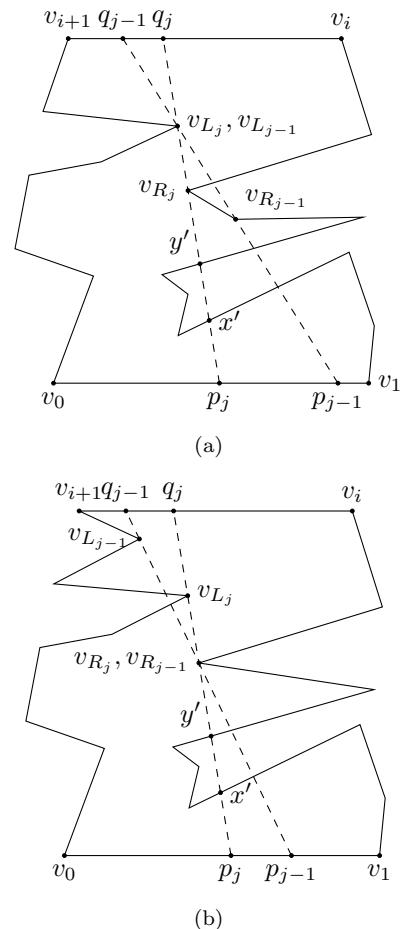


Figure 4: Cases in the proof of Theorem 4. (a) Case 1.3.1.3. (b) Case 1.3.2.

Theorem 5 *There are at most $2n - 6$ iterations of the loop at line 5 of Algorithm 1.*

Proof. Let $N(n)$ be the maximal number of edge visits for a polygon with n vertices. Consider the first time line 12 or 19 is executed, assume it is line 12. There has been made $2(r-1) - 1 < 2(r-1)$ iterations, because $\mathcal{P}(v_1, v_i)$ is traversed every second time, beginning with the first. The $r-1$ edges in $\mathcal{P}(v_1, v_r)$ are never traversed again. Therefore, N satisfies the recurrence $N(n) \leq 2k + N(n-k)$, where $k = r-1$. The same bound holds for some $k \geq 1$ if line 19 is executed first. We know that $N(4) = 2$, so induction yields that $N(n) \leq 2n - 6$ is an upper bound. \square

It is clear that an algorithm to compute a leftmost ray from v_0v_1 to v_iv_{i+1} can be constructed symmetrically. That gives us the following theorem:

Theorem 6 *The visible part of an edge v_iv_{i+1} from v_0v_1 in a simple polygon can be computed in $O(n)$ time using constant workspace.*

3 Weak Visibility Polygons and Minimum Link Paths

The weak visibility polygon of the polygon \mathcal{P} from the edge v_0v_1 consists of all the points in \mathcal{P} visible from v_0v_1 . Guibas et al. [9] presented an $O(n)$ -time algorithm to compute the weak visibility polygon if a triangulation of \mathcal{P} was provided, where n is the number of vertices of \mathcal{P} . Later, Chazelle [6] described an $O(n)$ -time deterministic triangulation algorithm, implying that the weak visibility polygon can be computed in $O(n)$ time using $O(n)$ space. Using Algorithm 1, one can make a $O(mn)$ -time algorithm using constant workspace, where m is the number of edges of the weak visibility polygon [1]. It is well-known that $m = O(n)$.

A minimum link path between two points s and t in a simple polygon is a polygonal path from s to t which is contained in \mathcal{P} and which consists of as few segments as possible. Suri [11] showed how to compute a minimum link path using $O(n)$ time if a triangulation of \mathcal{P} is provided. Using the algorithm to compute the weak visibility polygon, it is possible to devise an $O(n^2)$ -time algorithm to compute a minimum link path using constant workspace. The details are given in [1].

Acknowledgements

We would like to thank Jyrki Katajainen, Ashwini Joshi, and Kristian Mortensen for suggesting many improvements to the present paper.

References

- [1] M. Abrahamsen. Constant-workspace algorithms for visibility problems in the plane. Master's thesis. University of Copenhagen, Department of Computer Science, 2013. Available at <http://www.diku.dk/forskning/performance-engineering/Mikkel/thesis.pdf>.
- [2] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. E-print arXiv:1112.5904, 2011, available at <http://arxiv.org/abs/1112.5904>.
- [3] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *Journal of Computational Geometry*, 2(1):46–68, 2011.
- [4] D. Avis, T. Gum, and G. Toussaint. Visibility between two edges of a simple polygon. *The Visual Computer*, 2(6):342–357, 1986.
- [5] L. Barba, M. Korman, S. Langerman, and R. Silveira. Computing the visibility polygon using few variables. *Algorithms and Computation, Lecture Notes in Computer Science Volume 7074*, pages 70–79, 2011.
- [6] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.
- [7] M. De, A. Maheshwari, and S. Nandy. Space-efficient algorithms for visibility problems in simple polygon. E-print arXiv:1204.2634, 2012, available at <http://arxiv.org/abs/1204.2634>.
- [8] S. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007.
- [9] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.
- [10] R. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.
- [11] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.
- [12] G. Toussaint. Shortest path solves edge-to-edge visibility in a polygon. *Pattern Recognition Letters*, 4(3):165–170, 1986.

Counting Carambolas

Maarten Löffler*

André Schulz†

Csaba D. Tóth‡

Abstract

We give upper and lower bounds on the maximum and minimum number of certain geometric configurations hidden in a triangulation of n points in the plane. Configurations of interest include *star-shaped polygons* and *monotone paths*. We also consider related problems in *directed planar straight-line graphs*.

1 Introduction

Problems in extremal combinatorics typically ask the minimum or maximum number of certain subconfigurations in a configuration of a given size. We consider extremal problems where both the configuration and the subconfiguration have geometric attributes. Consider a (straight-line) triangulation of n points in the plane. Buchin et al. [1] showed that every triangulation contains $O(2.893^n)$ simple cycles, and there are triangulations that contain $\Omega(2.4262^n)$ simple cycles and $\Omega(2.0845^n)$ Hamilton cycles. Buchin and Schulz [2] proved that every n -vertex triangulation contains $O(5.2852^n)$ spanning trees. These techniques are instrumental for bounding the total number of noncrossing Hamilton cycles and spanning trees that n points in the plane admit [4, 6].

Van Kreveld et al. [7] were the first to consider substructures with geometric attributes. They proved that every triangulation contains $O(1.62^n)$ convex polygons (cycles), and some contain $\Omega(1.5028^n)$ convex polygons. Their upper bound is based on counting *star-shaped* polygons in a “reduced” graph, which is a carefully constructed subgraph of a given triangulation.

In this note, we consider subgraphs of a straight-line triangulation with other common geometric attributes. A *star-shaped* polygon (a.k.a. *carambola*, see Fig. 1) is a simple polygon P such that there is a point o with the property that every ray emanating from o intersects the boundary of P in exactly one point. Star-shaped polygons are closely related to monotone paths. A path P is *monotone* in direction $\underline{u} \in \mathbb{R}^2$, $\underline{u} \neq \underline{0}$, if every line orthogonal to \underline{u} intersects P in at most one point. A special case is an *x-monotone* path, which is monotone

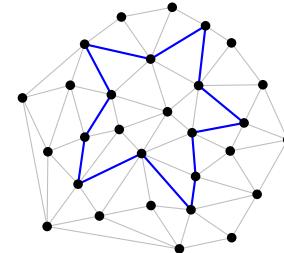


Figure 1: A “carambola” in a triangulation..

in horizontal direction $\underline{u} = (1, 0)$. Table 1 summarizes known and new results:

configurations	lower bound	upper bound
convex polygons	$\Omega(1.50^n)$ [7]	$O(1.62^n)$ [7]
star-shaped polygons	$\Omega(1.70^n)$	$O(n^3\alpha^n)$
monotone paths	$\Omega(1.70^n)$	$O(n\alpha^n)$
directed simple paths	$\Omega(\alpha^n)$	$O(n^23^n)$

Table 1: Bounds for the maximum number of configurations in an n -vertex plane (di)graph. Results in row 1 are and included for comparison; the bounds in rows 2-4 are proved in the paper. Note that $\alpha \approx 1.84$ is the real root of $x^3 = x^2 + x + 1$.

2 Lower Bounds

We construct plane straight-line graphs on n vertices that contain $\Omega(1.70^n)$ x -monotone paths (Fig. 2 and 3). By orienting all edges from left to right, we obtain a directed planar graph that contains $\Omega(1.70^n)$ directed paths. By connecting the leftmost and rightmost vertices by an extra edge, we obtain a plane straight-line graph that contains $\Omega(1.70^n)$ monotone polygons. By arranging three copies of this graph around the origin in a cyclic fashion (Fig. 4), we obtain a plane straight-line graph that contains $\Omega(1.70^n)$ star-shaped polygons.

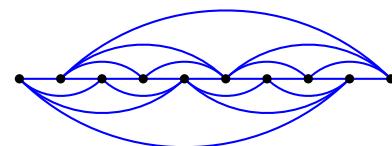


Figure 2: There are 1.70^n monotone paths in this graph.

Let $n = 2^\ell + 2$ for an integer $\ell \in \mathbb{N}$. We define the plane graph G on n vertices $V = \{v_1, \dots, v_n\}$: it consists

*Department of Computing and Information Sciences, Utrecht University, m.loffler@uu.nl.

†Institut für Mathematische Logik und Grundlagenforschung, Universität Münster, andre.schulz@uni-muenster.de.

‡Department of Mathematics, California State University Northridge, and University of Calgary, cdtoth@acm.org.

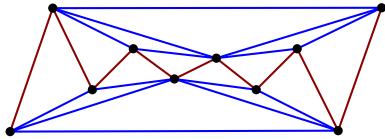


Figure 3: A straight-line embedding of the graph in Fig. 2, where the points lie alternately on two circular arcs, preserving x -monotonicity.

of a path (v_1, \dots, v_n) and two balanced binary triangulation of the vertices $\{v_1, \dots, v_{n-1}\}$ and $\{v_2, \dots, v_n\}$, respectively, one on each side of the path (Fig. 2). A straight-line embedding is shown in Fig. 3, where the odd (resp., even) vertices lie on a concave (convex) polygonal chain.

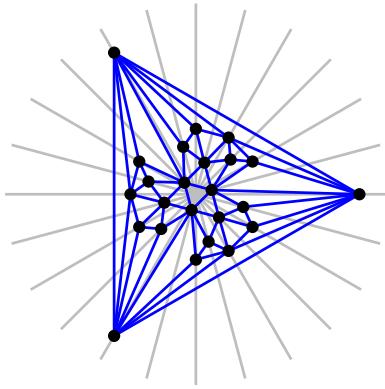


Figure 4: Cyclic embedding of 3 copies of the graph in Fig. 2. The monotone order becomes cyclic order.

Theorem 1 *The graph G described in the above paragraph has $\Omega(1.700^n)$ x -monotone paths.*

Proof. We count the number of x -monotone paths in a sequence of subgraphs of G . Let G_0 be a Hamilton path (v_1, \dots, v_n) ; and we recursively define G_k from G_{k-1} by adding the edges (v_i, v_{i+2^k}) for $i = j2^k + 1$ and $i = j2^k + 2$ for $j = 0, 1, \dots, \ell/k - 1$. Denote by $p_k(v_i)$ the number of x -monotone paths in G_k that end at vertex v_i . Since every monotone path can be extended to the rightmost vertex v_n , the number of maximal[□] monotone paths in G_k is $p_k(v_n)$. We establish recurrence relations for $p_k(v_i)$. The initial values are $p_k(v_1) = p_k(v_2) = 1$ for all $k = 0, \dots, \ell$. For $k = 1$ and $i \geq 2$, we have $p_1(v_i) = p_1(v_{i-1}) + p_1(v_{i-2})$, therefore $p_1(v_i) = F_i$ is the i th Fibonacci number and $p_1(v_n) = \Theta(1.619^n)$.

The recurrence for $p_k(v_i)$, $k \geq 2$, is more nuanced, due to the asymmetry between the triangulations on the two sides of the Hamilton path. We partition the vertices of G_k into disjoint groups of consecutive vertices of size 2^k . Let $a_i = v_{i2^k+1}$ denote the first vertex of group i ,

[□]Maximal for containment.

and let $b_i = v_{i2^k+2}$ be the second vertex of group i . We count in how many ways one can route an x -monotone path through a group i . A path through group i starts at either a_i or b_i , and ends at either a_{i+1} or b_{i+1} . Thus, it is enough to keep track of four different type of paths. By our choice, the edge (a_i, b_i) belongs to group i but not to group $i + 1$. We record the number of paths connecting a_i or b_i to a_{i+1} or b_{i+1} in a 2×2 matrix M_k , such that

$$M_k \cdot (p_k(a_i), p_k(b_i))^T = (p_k(a_{i+1}), p_k(b_{i+1}))^T.$$

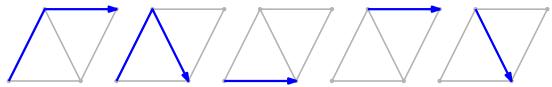


Figure 5: The five possible x -monotone paths in the group of G_2 .

Once the matrix M_k is known, we can compute the number of paths by $(p(v_{n-1}), p(v_n))^T = M_k^{n/2^k} \cdot (1, 1)^T$. By the Perron-Frobenius theorem, $\lim_{p \rightarrow \infty} M_k^p / \lambda^p = A$, for some matrix A , and for λ being the largest eigenvalue of M_k . Hence, we have $\lim_{n \rightarrow \infty} T_k(v_n) = \Theta(\lambda^{n/2^k})$ maximal x -monotone paths in G_k .

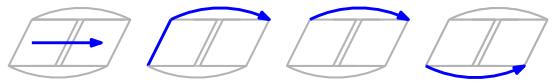


Figure 6: Schematic drawing of the paths counted by M_k .

In the last step we show how to compute the matrices M_k . The matrix M_2 can be easily obtained by hand (see Fig. 5). For a block of size 2^k in G_k , we have all the paths that use only the edges in G_{k-1} (and can therefore be decomposed in two paths of G_{k-1} 's blocks) plus one additional path of type $a_i \rightarrow a_{i+1}$, $a_i \rightarrow b_{i+1}$, and $b_i \rightarrow b_{i+1}$ each (see Fig. 6). Therefore, we can compute the matrices M_k as

$$M_2 := \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \text{ and } M_i := M_{i-1}^2 + \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

k	2	3	4	5	6
$\lambda^{2^{-k}}$	1.61803	1.69605	1.70034	1.70037	1.70037

Table 2: The asymptotic growth of x -monotone paths in the graphs G_k . For $k = 6$ it follows that there are $\Omega(1.70037^n)$ monotone paths.

Table 2 shows the values $\lambda^{1/2^k}$ for $k = 2, \dots, 6$. We observed that when going from $k = 5$ to $k = 6$, there is no change in $\lambda^{1/2^k}$ up to 8 digits after the point. The precise value for $k = 5$ equals $\lambda = 1/2(4885 + 9\sqrt{294153})$. \square

Directed plane graphs. We also construct n -vertex directed plane straight-line graphs that contain $\Omega(1.83^n)$ directed paths (Fig. 7). The directed paths, however, cannot be extended to a cycle because in a planar embedding the start and end vertex are not in the same face. Similarly, most of the directed paths are not monotone in any direction in a planar embedding.

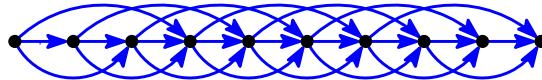


Figure 7: There are $\Theta(\alpha^n)$, $\alpha \approx 1.84$, directed paths in this graph. A plane embedding of the graph is depicted in Fig. 8.

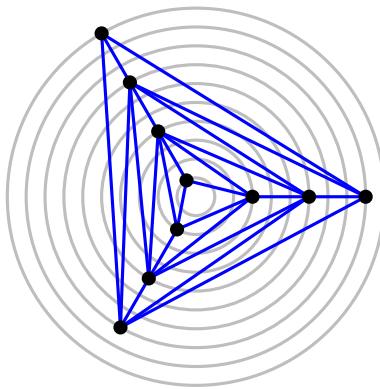


Figure 8: A plane embedding of the graph in Fig. 7. The edges are directed from the outer circles to inner inner circles.

Denoting by $T(i)$ the number of directed paths ending at vertex v_i , we have $T(1) = T(2) = 1$, $T(3) = 2$, and a linear recurrence relation

$$T(i) = T(i-1) + T(i-2) + T(i-3)$$

for $i \geq 4$. The recurrence solves to $T(i) = O(\alpha^i)$, where $\alpha \approx 1.83929$ is the real root of $x^3 = x^2 + x + 1$. Therefore the total number of directed paths, starting from any vertex, is $\Theta(\alpha^n)$.

3 Upper Bounds

Monotone Paths. We start with x -monotone paths in plane straight-line graph. We prove the bound for a broader class of graphs, *plane monotone graphs*, in which every edge is an x -monotone Jordan arc (since some of the operations in our argument may not preserve straight-line edges).

Let $n \in \mathbb{N}$, $n \geq 3$, and let $G = (V, E)$ be a plane monotone graph with $|V| = n$ vertices that maximizes the number of x -monotone paths. We may assume that the vertices have distinct x -coordinates (otherwise we can perturb the vertices without decreasing the number of x -monotone paths). We may also assume that G

is fully triangulated (i.e., it is an edge-maximal planar graph), otherwise we add extra edges which only increase the number of x -monotone paths. Label the vertices in V as v_1, v_2, \dots, v_n sorted by their x -coordinates. Orient each edge $\{v_i, v_j\} \in E$ from v_i to v_j if $i < j$.

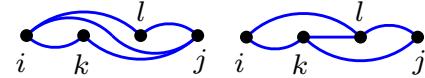


Figure 9: The flip operation.

Consider an edge $\overline{v_i v_j} \in E$ that is not on the boundary. There are two triangular faces incident to $\overline{v_i v_j}$, and two other vertices v_k and v_l that are connected to both v_i and v_j .

Claim 2 If $i < k < j$ and $i < l < j$, then flipping $\overline{v_i v_j}$ to $\overline{v_k v_l}$ or $\overline{v_l v_k}$ (depending on whether $k < l$ or $l < k$) only increases the number of paths. (See Fig. 9.)

Since G has the maximum number of x -monotone paths, we may now assume that for all edges in G , there is a vertex either to the left or to the right of the edge that is adjacent to both endpoints. We show next that G contains an x -monotone Hamilton path.

Lemma 3 All edges $\overline{v_i v_{i+1}}$ are present in G .

Proof. Suppose, to the contrary, that there are two non-adjacent vertices v_i and v_{i+1} . Since G is a triangulation, there must be an edge $\overline{v_j v_k}$, with $j < k$, that separates v_i and v_{i+1} . Since the edge $\overline{v_j v_k}$ is x -monotone, we have $j < i < i+1 < k$. Assume w.l.o.g. that v_i lies below $\overline{v_j v_k}$ and v_{i+1} lies above $\overline{v_j v_k}$. By Claim 2, the triangle incident to $\overline{v_j v_k}$ from either above or below connects to a vertex v_l either to the left of v_j or to the right of v_k . Assume w.l.o.g. it is the triangle above, and that v_l lies to the right of v_k . Now consider edge $\overline{v_j v_l}$. Since the triangle below it has v_k as third vertex and $j < k < l$, there must be another vertex v_m that connects to $\overline{v_j v_l}$ and lies either to the left of j or to the right of l . This argument repeats, and we never reach v_{i+1} . Contradiction. \square

For any pair $i < j$, let V_{ij} denote the set of consecutive vertices v_i, v_{i+1}, \dots, v_j , and let $G_{ij} = (V_{ij}, E_{ij})$ be the subgraph of G induced by V_{ij} .

Since G is planar, we know that $|E| \leq 3|V| - 6$, and furthermore, that $|E_{ij}| \leq 3|V_{ij}| - 6$ for all subgraphs induced by groups of 3 or more consecutive vertices.

In the remainder of the proof we will apply a sequence of operations on G that may create multiple edges and edge crossings. Hence, we consider G as an abstract multigraph. However, the operations will maintain the invariant that $|E_{ij}| \leq 3|V_{ij}| - 6$ for all $i < j$.

Let $i < j < k$ be a triple of indices such that $\overline{v_i v_j}, \overline{v_i v_k} \in E$. The operation $\text{shift}(i, j, k)$ removes the

edge $\overline{v_i v_k}$ from E , and inserts the edge $\overline{v_j v_k}$ into E (see Fig. 10). Note that the new edge may already have been present, in this case we insert a new copy of this edge (i.e., we increment its multiplicity by one).

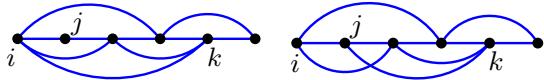


Figure 10: The operation $\text{shift}(i, j, k)$.

Claim 4 *The operation $\text{shift}(i, j, k)$ does not decrease the number of x -monotone paths in G .*

Proof. Clearly, any path that used $\overline{v_i v_k}$ can be replaced by a path that uses $\overline{v_i v_j}$ and (the new copy of) $\overline{v_j v_k}$. \square

Now, we apply the following algorithm to the input graph G . We process the vertices from left to right, and whenever we encounter a vertex v_i with outdegree 4 or higher, we identify the smallest index j such that v_i has an edge to v_j and the largest index k such that v_i has an edge to v_k ; and then apply $\text{shift}(i, j, k)$. We repeat until there are no more vertices with outdegree larger than 3.

Claim 5 *The algorithm terminates and maintains the invariants that (1) all edges $\overline{v_i v_{i+1}}$ are present in G with multiplicity one; and (2) $|E_{ij}| \leq 3|V_{ij}| - 6$ for all subgraphs induced by V_{ij} , $i < j$.*

Proof. Initially, invariant (1) holds by Lemma 3, and (2) by planarity. Suppose, to the contrary, that there is an operation that increases the number of edges of an induced subgraph above the threshold. Let $\text{shift}(i, j, k)$ be the first such operation. Since the only new edge is $\overline{v_j v_k}$, any violator subgraph must contain both v_j and v_k ; and it cannot contain v_i since the only edge removed is $\overline{v_i v_k}$. Recall that v_j was the leftmost vertex that v_i is adjacent to; and by invariant (1), we know $j = i + 1$. Therefore, the violator subgraph is induced by $V_{jk'}$ for some $k' \geq k$, and we have $|E_{jk'}| > 3|V_{jk'}| - 6$ after the shift. Since v_k was the rightmost vertex adjacent to v_i before the shift, all outgoing edges of v_i went to vertices in $V_{jk'}$. The outdegree of v_i was at least 4 before the shift, hence $G_{ik'}$ had at least $3|V_{ik'}| - 4$ edges. Contradiction. \square

Now, after executing the algorithm, we are left with a multigraph where the outdegree of every vertex is at most 3, and no subgraph induced by $|V_{ij}| \geq 3$ consecutive vertices has more than $3|V_{ij}| - 6$ edges. This, combined with invariant (1), implies that the multiplicity of any edge $\overline{v_i v_{i+2}}$ is at most one. Thus, for every vertex v_i , the three outgoing edges go to vertices at distance at least 1, 2, and 3, respectively, from v_i . Denoting by $T(i)$

the number of x -monotone paths that start at v_{n-i+1} , we arrive at the recurrence

$$T(i) = T(i-1) + T(i-2) + T(i-3)$$

for $i \geq 4$, with initial values $T(1) = T(2) = 1$, $T(3) = 2$. The recurrence solves to $T(n) = O(\alpha^n)$ where $\alpha \approx 1.83929$ is the real root of $x^3 - x^2 - x - 1 = 0$. Therefore, every plane straight-line graph on n vertices admits at most $O(\alpha^n)$ x -monotone paths.

Since the edges of an n -vertex planar straight-line graph have $O(n)$ distinct directions, the number of monotone paths (in any direction) is bounded by $O(n\alpha^n)$.

Star-shaped Polygons. Given a plane straight-line graph G on n vertices, the lines passing through the $O(n)$ edges induce a line arrangement with $O(n^2)$ faces. Choose a face f of the arrangement, and a vertex p of G . We show that G contains $O(\alpha^n)$ star-shaped polygons with vertex v and a star center lying in f . Indeed, pick an arbitrary point $o \in f$. Each edge of G is oriented either clockwise or counterclockwise with respect to o (with the same orientation for any $o \in f$). Order the vertices of G by a rotational sweep around o starting from the ray \overrightarrow{ov} . Let G_v be the graph obtained from G by deleting all edges that cross the ray \overrightarrow{ov} . We can repeat the argument for x -monotone path for G_f , replacing the x -monotone order by the rotational sweep order about o , and conclude that G admits $O(\alpha^n)$ star-shaped polygons with vertex p and star center o .

Directed Simple Paths. Let $G = (V, E)$ be a directed planar graph. Denote by $\deg^-(v)$ the outdegree of vertex $v \in V$; let $V_0 = \{v_1, \dots, v_\ell\}$ be the set of vertices with outdegree at least 1, where $1 \leq \ell \leq n$. We show that for every $v_0 \in V_0$, there are $O(3^n)$ maximal² directed simple paths starting from v_0 . Each maximal directed simple path can be encoded in an ℓ -dimensional vector that contains the outgoing edge of each vertex $v \in V_0$ in the path (and an arbitrary outgoing edge if $v \in V_0$ is not part of the path). The number of such vectors is

$$\prod_{i=1}^{\ell} \deg^-(v_i) \leq \left(\frac{1}{\ell} \sum_{i=1}^{\ell} \deg^-(v_i) \right)^{\ell} < \left(\frac{3n}{\ell} \right)^{\ell} \leq 3^n,$$

where we have used the geometric-arithmetic mean inequality, Euler's formula $\sum_{i=1}^{\ell} \deg^-(v_i) \leq 3n - 6 < 3n$, and maximized the function $x \rightarrow (3n/x)^x$ over the interval $1 \leq x \leq n$. Since there are $O(n)$ choices for the starting vertex $v_0 \in V_0$, and a maximal simple path contains $O(n)$ nonmaximal paths starting from the same vertex, the total number of simple paths is $O(n^2 3^n)$.

²Maximal for containment.

4 Minimizing the number of configurations

In this section, we explore the *minimum* number of geometric configurations that a triangulation on n points in the plane can have. Our bounds are summarized in Table 3.

configurations	lower bound	upper bound
convex polygons	$\Omega(n)$	$O(n)$
star-shaped polygons	$\Omega(n)$	$O(n^2)$
monotone paths	$\Omega(n^2)$	$O(n^{3.39})$
directed paths	$\Omega(n)$	$O(n)$

Table 3: Bounds for the minimum number of configurations in a (directed) triangulation with n vertices.

Potatoes. Every n -vertex triangulation has $\Theta(n)$ convex faces, hence $\Omega(n)$ is a natural lower bound for the number of convex polygons. The triangulation in Fig. 11(left) contains $\Theta(n)$ convex polygons, which is the best possible apart from constant factors. The triangulation consists of the join of two paths $P_2 * P_{n-2}$, where path P_{n-2} is realized as a monotone zig-zag path. Every convex polygon is either a triangle or the union of two adjacent triangles that share a flippable edge [5].

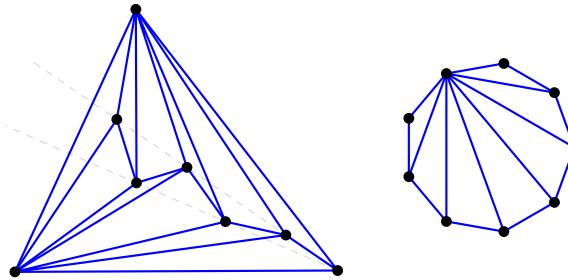


Figure 11: There are $\Theta(n)$ convex polygons and x -monotone paths in the triangulation on the left; it contains $\Theta(n^4)$ star-shaped polygons and monotone paths. There are $\Theta(n^2)$ star-shaped polygons and $\Theta(n^4)$ monotone paths in the triangulation on the right.

Carambolas. The sum of degree squares $\Omega(\sum_{v \in V} \deg^2(v)) = \Omega(n)$ is a natural lower bound for the number of star-shaped polygons, since the union of consecutive triangles incident to a vertex forms a star-shaped polygon. This might suggest that a triangulation that minimizes the number of star-shaped polygons should have bounded degree. Surprisingly, the best construction found so far is a triangulation with a vertex of degree $n - 1$ (Fig. 11, right), which admits $\Theta(n^2)$ star-shaped polygons.

Monotone paths. It is not difficult to see that between any two vertices, u and v , in a triangulation there is a

monotone path in direction \vec{uv} [3]. Hence every triangulation contains $\Omega(n^2)$ monotone paths. Two vertices, however, are not always connected by an x -monotone path: a trivial lower bound for x -monotone paths is $\Omega(n)$, since every edge is x -monotone.

The triangulation $P_2 * P_{n-2}$ in Fig. 11(left) is embedded such that P_{n-2} is x -monotone and lies to the right of P_2 . With this embedding, it contains $\Theta(n^2)$ x -monotone paths: every x -monotone path consists of a sequence of consecutive vertices of P_{n-2} , and 0, 1, or 2 vertices of P_2 . However, both triangulations in Fig. 11 admit $\Theta(n^4)$ monotone paths (in some direction).

Triangulations with a polynomial number of monotone paths are also provided by known constructions in which all monotone paths are “short.” Dumitrescu et al. [3] constructed full triangulations with maximum degree $O(\log n / \log \log n)$ such that every monotone path has $O(\log n / \log n \log n)$ edges. Furthermore, there are triangulations with bounded degree in which every monotone path has $O(\log n)$ edges. These constructions contain polynomially many, but $\omega(n^4)$, monotone paths.

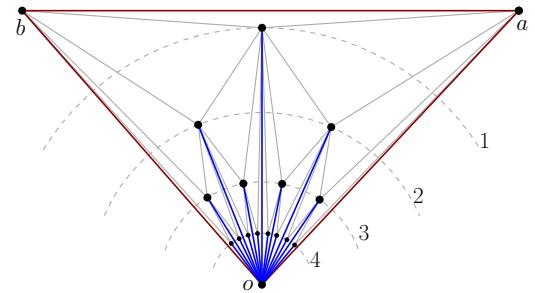


Figure 12: A triangulation from [3] in which every monotone path has $O(\log n)$ edges.

A triangulation that contains only $O(n^\beta \log^2 n)$ monotone paths, where $\beta = 2 + 2 \log_2((1 + \sqrt{5})/2) \approx 3.3885$ comes from [3]: It has maximum degree $n - 1$ and every monotone path has $O(\log n)$ edges. Refer to Fig. 12.

The number of vertices is $n = 2^\ell + 1$ for some $\ell \in \mathbb{N}$. The outer face is a regular triangle abo , where o is the origin. The interior vertices are arranged on $\ell - 1$ circles centered at the origin, with 2^i points on circle i , where the radii of the circles rapidly converge to 0. The vertices on circle i are drawn interspersed (in angular order) with the vertices of the previous layers. The origin is connected to all other vertices, and a vertex on circle i is connected to the two vertices of the previous layers that are closest in angular order. The radii of the circles are chosen recursively such that the edges that connect an interior vertex v to vertices on smaller circles are almost parallel—thus a monotone path can contain two such edges for at most one interior vertex v . It follows that every monotone path contains at most two vertices from each circle, hence the $O(\log n)$ bound on the number of edges [3].

Claim 6 For every $n \in \mathbb{N}$, there is an n -vertex triangulation that admit $O(n^\beta \log^2 n)$ monotone paths, where $\beta = 2 + 2 \log_2((1 + \sqrt{5})/2) \approx 3.3885$.

Proof. In the above construction, it is enough to count *maximal* monotone paths, since every monotone path can be extended to the outer triangle abo , and each maximal monotone path contains only $O(\log^2 n)$ subpaths. First, consider paths between a and b . For every path between a and b , we can record the layers of the vertices along the paths, where a and b are at level 0, and o is at level ℓ . This sequence must be unimodal for a monotone path (by construction). An a - b path that avoids the origin is uniquely determined by its modality (the vertex lying on the smallest circle), hence there are at most $O(n)$ such paths (all these paths happen to be monotone).

Consider now the paths incident to o . An a - b path that goes through o is counted as the combination of a path from o to a and one from o to b . By symmetry, it is enough to count monotone paths from o to a . Such a path also corresponds to a unimodal sequence (with o being the only modality). We have $n - 1$ choices for the first edge of incident to o , and the remainder of the path is restricted to an outerplanar graph with at most $\ell + 2 = 1 + \log(n - 1)$ vertices.

In an outerplanar graph with k vertices, any two vertices are connected by at most $F_k = \Theta((1 + \sqrt{5})/2)^k = O(1.62^k)$ paths, where F_k is the k th Fibonacci number. Therefore, the number of o - a paths is at most $n \cdot F_{\ell+2} = \Theta(n^{1+\log_2((1+\sqrt{5})/2)}) = O(n^{1.70})$. (In fact, all these paths are monotone.) Every path from a to b via o is the combination of two branches: a path from o to a and one from o to b . Hence the number of these paths is bounded by $O(n^\beta \log^2 n)$, where $\beta = 2 + 2 \log_2((1 + \sqrt{5})/2) \approx 3.3885$. \square

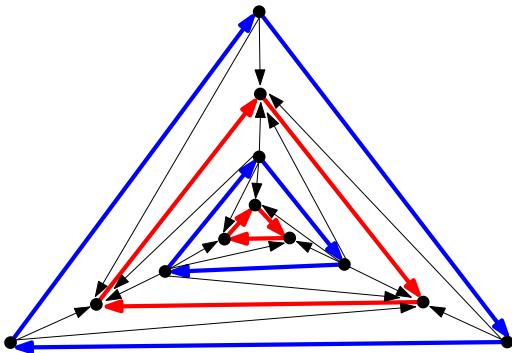


Figure 13: There are $\Theta(n)$ directed paths in this directed planar graph.

Directed paths. Every edge in a planar digraph is a directed path on its own, hence there are $\Omega(n)$ directed paths in every directed triangulation on n vertices. This

bound is tight, apart from constant factors. Directed triangulation with $O(n)$ paths consists of a sequence of $n/3$ triangles, and edges between consecutive triangles point either inward or outward alternately (Fig. 13).

5 Conclusion

We considered the maximum and minimum number of star-shaped polygons, monotone paths, and directed paths that a (directed) triangulation of n points in the plane can have. Our results are summarized in Tables 1 and 3. Closing or narrowing the gap between the upper and lower bounds is left for future research.

Acknowledgments

M. Löffler is supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123. Tóth is supported in part by NSERC (RGPIN 35586) and NSF (CCF-0830734). This work was initiated at the workshop “Counting and Enumerating of Plane Graphs”, which took place at Schloss Dagstuhl in March 2013.

References

- [1] K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel. On the number of cycles in planar graphs. In *Proc. 13th Annual International Conference on Computing and Combinatorics (COCOON)*, volume 4598 of *Lecture Notes in Computer Science*, pages 97–107. Springer, 2007.
- [2] K. Buchin and A. Schulz. On the number of spanning trees a planar graph can have. In M. de Berg and U. Meyer, editors, *Proc. 18th Annual European Symposium on Algorithms (ESA)*, volume 6346 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2010.
- [3] A. Dumitrescu, G. Rote, and C. D. Tóth. Monotone paths in planar convex subdivisions. In *Proc. 18th Annual International Conference on Computing and Combinatorics (COCOON)*, volume 7434 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2012.
- [4] M. Hoffmann, A. Schulz, M. Sharir, A. Sheffer, C. D. Tóth, and E. Welzl. Counting plane graphs: flippability and its applications. In J. Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 303–326. Springer, 2013.
- [5] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.
- [6] M. Sharir, A. Sheffer, and E. Welzl. Counting plane graphs: Perfect matchings, spanning cycles, and Kasteleyn’s technique. *J. Comb. Theory, Ser. A*, 120(4):777–794, 2013.
- [7] M. van Kreveld, M. Löffler, and J. Pach. How many potatoes are in a mesh? In *Proc. 23rd International Symposium on Algorithms and Computation (ISAAC)*, volume 7676 of *Lecture Notes in Computer Science*, pages 166–176. Springer, 2012.

Cell-Paths in Mono- and Bichromatic Line Arrangements in the Plane*

Oswin Aichholzer[†] Jean Cardinal[‡] Thomas Hackl[†] Ferran Hurtado[§] Matias Korman[§]
 Alexander Pilz[†] Rodrigo I. Silveira[§] Ryuhei Uehara[¶] Birgit Vogtenhuber[†] Emo Welzl^{||}

Abstract

We show that in every arrangement of n red and blue lines—in general position and not all of the same color—there is a path through a linear number of cells where red and blue lines are crossed alternatingly (and no cell is revisited). When all lines have the same color, and hence the preceding alternating constraint is dropped, we prove that the dual graph of the arrangement always contains a path of length $\Theta(n^2)$.

1 Introduction

Given an arrangement of n red and blue lines in the Euclidean plane, we consider sequences of cells of the arrangement such that consecutive cells share an edge and no cell appears more than once in the sequence. We refer to such sequences as *cell-paths*, or simply *paths*. A path is called *alternating* if the common edges of consecutive cells alternate in color. The *length* of a path is defined to be one less than the number of cells involved. Cell-paths can also be seen as paths in the dual graph of the arrangement, in which there is a node for every cell in the arrangement, and an edge connects two nodes

*OA and BV supported by the ESF EUROCORES programme EuroGIGA—CRP ‘ComPoSe’, Austrian Science Fund (FWF): I648-N18. JC supported by the ESF EUROCORES programme EuroGIGA—CRP ‘ComPoSe’ and F.R.S.-FNRS Grant R70.01.11F. TH supported by the Austrian Science Fund (FWF): P23629-N18 ‘Combinatorial Problems on Geometric Graphs’. FH, MK, and RS partially supported by projects MINECO MTM2012-30951, Gen. Cat. DGR2009SGR1040, and ESF EUROCORES programme EuroGIGA—CRP ‘ComPoSe’: MICINN Project EUI-EURO-2011-4306. MK received support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union. AP is a recipient of a DOC-fellowship of the Austrian Academy of Sciences at the Institute for Software Technology, Graz University of Technology, Austria. RS was funded by FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235. EW supported by EuroCores/EuroGiga/ComPoSe SNF 20GG21 134318/1.

[†]Graz University of Technology, Graz, Austria, [oaich|thackl|apilz|bvogt]@ist.tugraz.at

[‡]Université Libre de Bruxelles, Brussels, Belgium, jcardin@ulb.ac.be

[§]Universitat Politècnica de Catalunya, Barcelona, Spain, [ferran.hurtado|matias.korman|rodrigo.silveira]@upc.edu

[¶]Japan Advanced Institute of Science and Technology, Ishikawa, Japan, uehara@jaist.ac.jp

^{||}ETH Zürich, Zürich, Switzerland, emo@inf.ethz.ch

when the corresponding cells are adjacent.

We consider the following question: Is there a function $p(n)$ tending to infinity so that every arrangement of n blue and red lines in general position (i.e., no three lines share a point and no two lines parallel) and not all of the same color has an alternating path of length at least $p(n)$? In Section 2 we answer this question in the affirmative proving that $p(n) \geq n$ (and give an upper-bound example with $p(n) = 2n - O(1)$).

If the n lines in the arrangement have the same color—a monochromatic arrangement—we ask a similar question: Is there a growing function $f(n)$ so that every arrangement of n lines in general position has a cell-path of length at least $f(n)$? One would expect that dropping the requirement for the path to be alternating from the previous problem would lead to a stronger result. Indeed, in Section 3 we prove that $f(n) = \Theta(n^2)$.

Previous work. Arrangements of (uncolored) lines have been thoroughly studied for decades [7, 10, 13, 14, 15, 22]. For example, properties of monotone paths in the arrangement have been studied (see, e.g., [9]). Substantial emphasis has been put into studying degenerate arrangements in which, e.g., the number of vertices decreases dramatically. Further, the kind of cells one may obtain as well as their extremal number were investigated (for example how many triangles appear in any simple arrangement). In another direction one can study the graph having as nodes the intersection points of lines, which are adjacent if they are consecutive in one of the lines [6], and study its basic properties as a graph, such as edge-colorings or whether it can be decomposed into Hamiltonian cycles (in projective space) [11]. For later use recall that the cells in any arrangement can be 2-colored chessboard-like, i.e., no two cells with the same color are adjacent [23] (see also [18]).

Not many problems on colored arrangements of lines were considered in the early times, in contrast to the rich (and still growing) literature on combinatorial problems on red and blue points [7, 20]. The first publications considered bichromatic sets of lines and studied the number and distribution of the intersection points of lines with the same color [16, 17, 26]. There is a recent line of research on problems in which lines have to be colored to achieve some property, or are already colored and one looks at the kind of cells that appear, regarding the color of their sides [3, 4, 5]. Our problem

on alternating paths adds to this trend.

The presented work on bichromatic line arrangements was inspired by a well known (still open) problem on points: consider a set R of n red points and a set B of n blue points in convex position, what is the longest alternating path spanned by these points (a crossing-free alternating Hamiltonian path on $R \cup B$ does not always exist)? Erdős (see [21]) proposed to study the value $\ell(n)$ such that a plane alternating path of length at least $\ell(n)$ always exists for any such pair R and B . Independently, Akiyama and Urrutia [2] considered the same problem and gave a necessary and sufficient condition for the existence of an alternating Hamiltonian path and an $O(n^2)$ algorithm to find one, if it exists. Abellanas et al. [1], and independently Kynčl et al. [21], proved that $\ell(n) \leq \frac{4}{3}n + O(\sqrt{n})$. Cibulka et al. [8] proved that $\ell(n) \geq n + \Omega(\sqrt{n})$. The gap is still to be closed. Other variations and related problems appear in Mészáros' PhD Thesis [24].

Finally, our results are also related to a long-standing open problem about paths in planar graphs. In 1963, Moon and Moser [25] showed that there exist three-connected planar graphs with n vertices in which the longest simple path has length at most $cn^{\log 2/\log 3}$, where c is some constant. It is conjectured (see [19]) that this is a lower bound as well, hence that every three-connected planar graph contains a path of this length. Our result on $f(n)$ shows that considering dual graphs of arrangements instead, we always get a path of length $\Omega(n^2)$.

2 Long Alternating Paths in Bichromatic Arrangements

In this section we prove the existence of long alternating paths in bichromatic arrangements. First observe that general position is important to allow a positive answer: Assume $n \geq 2$ and take all the n lines in the arrangement to go through a common point, so that all of the red lines have slope between 0 and 1, and all of the blue lines have positive slope larger than 1. Now every alternating path has length at most two. This holds since each cell on an alternating path, except for the first and the last one, has to be *bichromatic*, i.e., has to have a red and a blue edge on its boundary, and the constructed arrangement has only four bichromatic cells, which do not share an edge among each other.

Further consider an arrangement of n lines, all blue except for exactly one of them red, in general position. Then the length of the longest alternating path is $2n - O(1)$: In the arrangement (as edge set) there are n red edges which can be used at most once in a path, so in an alternating path at most $2n+1$ edges can be used; hence the upper bound. For the lower bound we go through the $2n$ cells along the red line, crossing the red line in

every other step, and a blue edge for entering a red-line-incident cell not yet visited in the steps in between; hence a path of length at least $2n - 1$.

The following lemma directly implies our main result for the stated problem on bichromatic arrangements. For the sake of convenience we delay the proof of Lemma 1 to Section 2.1.

Lemma 1 *Any pair of bichromatic cells in an arrangement of red and blue lines in general position is connected by an alternating path.*

Consider two antipodal bichromatic infinite cells in an arrangement (“antipodal” means that the cells are separated by all n lines). As long as there is at least one line of each color, such a pair of cells has to exist and, by Lemma 1, is connected by an alternating path. Clearly, such a path has to cross every line at least once.

Theorem 2 *In a set of n blue and red lines — in general position and not all of the same color — there is an alternating path of length n .*

By the example with exactly one red line, the bound in the theorem is asymptotically tight. However, if we require the same number of red and blue lines, we do not know whether longer alternating paths always exist.

2.1 Proof of Lemma 1

The graph underlying our problem is the dual graph of the arrangement: the $\binom{n+1}{2} + 1$ cells are the nodes of the graph, two nodes are adjacent if their corresponding cells share an edge in the arrangement. In order to capture the ‘alternating’ property, we can orient the edges as follows. First, observe that the graph has a proper 2-coloring (choose the color of a cell according to the parity of the number of lines above the cell), for which we choose the colors “r-out” and “b-out”. Now direct the edges of the dual graph by directing red edges (i.e., edges dual to red edges in the arrangement) from color r-out to color b-out and blue edges from color b-out to color r-out. This is illustrated in Figure 1. It is an easy exercise to verify that every (undirected) alternating path in the arrangement can be directed in one way so that it appears as a directed path in this oriented version of the dual of the arrangement — and vice versa, every directed path is clearly alternating.

Let us fix an arbitrary bichromatic cell z in the arrangement and consider the set of all cells that can be reached by a directed path in the just defined directed graph. Note that the construction of this directed graph is not unique, since we can obtain a second one by changing directions of all edges. We arbitrarily choose one and proceed. Now consider the closure of the union, $\text{reach}(z)$, of all the cells that are reachable by a directed path starting from z in the oriented graph.

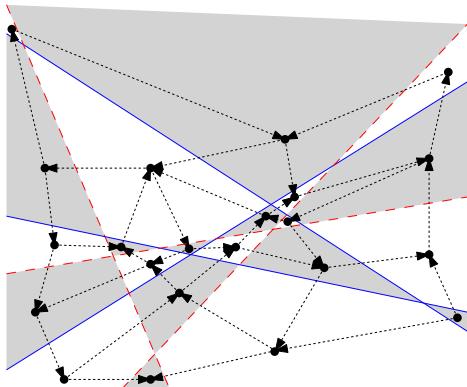


Figure 1: Orientation of the dual graph of a bichromatic line arrangement. Every bichromatic vertex of the arrangement yields a four-cycle in the oriented graph.

Here comes the crucial observation: Let us call a vertex in the arrangement *bichromatic*, if it is the intersection of a red and a blue line. Then the four cells incident to a bichromatic vertex form a directed cycle in the oriented graph and thus either all four of them are contained in $\text{reach}(z)$ or none of them is. (Here we implicitly use the fact that the existence of a directed walk—i.e., with repetitions of vertices allowed—from node x to node y implies the existence of a directed path from x to y .) Therefore, every bichromatic vertex is interior either to $\text{reach}(z)$ or to its complement.

We have now established the following fact.

Lemma 3 *Let E be the set of edges of the arrangement that separate cells in $\text{reach}(z)$ from its complement. No red edge in E shares a vertex with a blue edge in E .*

We will show that bichromatic vertices cannot be interior to the complement of $\text{reach}(z)$. Thus all of them have to be interior to $\text{reach}(z)$ and therefore, all bichromatic cells are in $\text{reach}(z)$ from which Lemma 1 follows. To this end consider a set X of separating edges that form a cycle or maximal path in the graph of separating edges. A maximal path has to start and end with an edge that extends to infinity. Consequently the union of edges in X separates the plane into two parts, one part of which contains the seed cell z . Suppose the edges in X are all blue (recall that they all have the same color). Then it is not possible that both sides contain a point on a red line, as otherwise, since we can travel on red lines between these two points, a red line had to cross the union of edges in X , which we know is not possible, since all of them are blue. Thus, the side that does not contain z must be completely monochromatic, i.e., all cells there are bounded by edges of the same color.

Since every point p in the complement of $\text{reach}(z)$ must have such a cycle or path X separating p from $\text{reach}(z)$, it follows that bichromatic vertices cannot be

interior to the complement of $\text{reach}(z)$, as claimed. As argued before, this implies Lemma 1.

2.2 Discussion

(1) As mentioned before, the linear bound on the length of the alternating path is probably not tight if an equal number of red and blue lines is required. However, abandoning the general position assumption, we have an example with the same number of red and blue lines, half of the lines vertical and half of them horizontal, so that the longest alternating path has only length $O(n)$.

(2) A closer inspection of the proof given shows that we have actually established the following.

Theorem 4 *Let C be a set of red and blue simple closed or biinfinite curves, each of which separates the plane into two parts. If the union of red curves is connected, the union of blue curves is connected, and no point is contained in more than two of the curves, then any pair of bichromatic cells in the arrangement is connected by an alternating path.*

(3) Similarly, Lemma 1 can be generalized to higher dimensions: Consider two antipodal bichromatic cells in a $(d+1)$ -dimensional arrangement. Intersect these two cells (and the arrangement) with a hyperplane H . The intersection of the arrangement with H gives a d -dimensional bichromatic arrangement, in which the antipodal cells are connected by induction.

3 Long Paths in (Monochromatic) Arrangements

We are given a set S of $n \geq 2$ lines in general position. Our aim is to find bounds on the length of the longest simple path in the dual graph of the arrangement. Let $A(S)$ be the arrangement associated with S , and let G be the dual graph of $A(S)$. Recall that the number of vertices of G is $N := \binom{n+1}{2} + 1$. We define $f(S)$ as the length of the longest simple path in G , and let $f(n) = \min_{|S|=n} f(S)$. In this section we show:

Theorem 5 $f(n) = \Theta(n^2)$.

This theorem is a direct consequence of Theorem 14 and Proposition 15 below. The lower bound actually holds for simple arrangements of *pseudolines* in the Euclidean plane. The main idea for its proof is to perform local transformations to G to make it four-connected, and then apply Tutte's Theorem, which states that every four-connected planar graph is Hamiltonian.

3.1 Lower Bound

G is a planar bipartite quasi-quadrangulation (i.e., every face of G has size four except for the unbounded one). It is easy to check that G is two-connected. We consider

the natural embedding of G given by S , in which every vertex of G is located in the corresponding face of $A(S)$, every edge of G intersects exactly one edge of $A(S)$, and the face corresponding to the unbounded cells of $A(S)$ is the outer face. Recall that a *vertex cutset* of a graph is a set of vertices whose removal disconnects the graph.

Let C be a simple cycle of G . By Jordan's Theorem, the removal of C from G decomposes the remaining vertices into two subsets, which we call *outer* and *inner*, where the outer one is the component that contains the unbounded cells of the arrangement. Given a vertex $v \in C$, we define its *inner degree* as the number of neighbors of v that belong to the inner component.

Lemma 6 *In any cycle C of G of length $2k$, the inner degree of any vertex of C is at most $k-2$ and the number of vertices in the inner part is at most $(k-1)(k-2)/2$.*

Proof. Consider the set $S(C) \subseteq S$ of lines associated with edges of C . Since C is a cycle, every line in $S(C)$ is intersected by C an even number of times, and at least twice. Hence, there are $|S(C)| \leq k$ such lines. As any vertex v of C is incident to two edges on C , there are exactly $|S(C)| - 2$ lines left that could correspond to edges incident to v and in the inner part of C . Further, the number of vertices in the inner part of C is at most the number of bounded cells of the arrangement formed by $S(C)$ and thus at most $(k-1)(k-2)/2$. \square

Let P be a simple path of G whose first and last vertex are incident to the outer face, while all other vertices are interior vertices of G . Then the removal of P splits the remaining vertices of G into two subsets as well. We refer to P as a *separating path* and to the induced subsets as *separated vertex sets*. The proof of the following lemma is similar to the one of Lemma 6.

Lemma 7 *For any separating path P of G with k vertices, one of the separated vertex sets of G has cardinality at most $(k-1)(k-2)/2$.*

Lemma 8 *Let C be a simple cycle of G , and let $I(C)$ be the set of vertices in its interior. If $I(C) \neq \emptyset$, there exists a simple cycle C' with the same set $I(C)$ in its interior such that no two consecutive vertices of C' have inner degree zero. For any separating path P which splits the remaining vertices of G into V_1 and V_2 , there exists a separating path P' such that (1) for each side of P' , at least one out of any two consecutive vertices of P' has an emanating edge to this side, (2) P' splits the remaining vertices of G into $V'_1 \supseteq V_1$ and $V'_2 \supseteq V_2$, and (3) the first and the last vertex of P' have emanating edges to both sides of P' .*

Proof. Suppose $I(C) \neq \emptyset$. As all faces of G in the interior of C have size four, at most two consecutive vertices

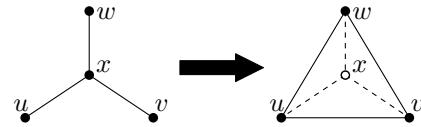


Figure 2: The Y-Δ transformation.

of C can have inner degree zero. Moreover, the only possibility of having two consecutive such vertices is that C uses three consecutive edges of a face in its interior. Iteratively replacing these edges by the fourth edge of this face wherever such a situation occurs, we obtain a simple cycle C' . C' has exactly $I(C)$ in its interior as well, and no two consecutive vertices of C' have inner degree zero. Similarly, in a separating path P , two consecutive vertices without emanating edges on one side can occur only if P uses three consecutive edges of a face on that side. Replacing all such occurrences on both sides gives the claimed properties. \square

Lemma 9 *All vertex cutsets of size two of G consist of the two neighbors of a degree-two vertex.*

Proof. Consider a cutset $C = \{c_1, c_2\}$ and the at least two resulting sets V_1, V_2 of the remaining vertices of G . First consider the case in which a component, say V_1 , contains only inner vertices of G . Then c_1 and c_2 are part of a cycle which has V_1 as its inner part. By Lemma 8 such a cycle with length at most four exists. Hence Lemma 6 implies that $V_1 = \emptyset$ and thus C is not a cutset. If both V_1 and V_2 contain vertices of the outer face, then by Lemma 8, c_1 and c_2 are the end points of a separating path with at most three vertices. Thus, by Lemma 7, $\min\{|V_1|, |V_2|\} \leq 1$, implying that c_1 and c_2 are the two neighbors of a degree-two vertex or C is not a cutset. \square

Lemma 10 *If C is a vertex cutset of size three of G where one of the separated sets does not contain any vertex of the outer face, then C consists of the three neighbors of a degree-three vertex.*

Proof. Consider a minimal cutset $C = \{c_1, c_2, c_3\}$ and let V_1 be a connected component that contains only interior vertices of G . By Lemma 8, C must be contained in a cycle of length at most six that has V_1 in its interior. Thus, Lemma 6 implies $|V_1| \leq 1$. \square

In order to construct a long path in G , we will use the following well-known result.

Theorem 11 (Tutte [27]) *Every four-connected planar graph is Hamiltonian.*

Given a degree-three vertex x in a graph, adjacent to u, v, w , the corresponding Y-Δ transformation consists of removing vertex x , adding edges uv , uw , and vw , and removing any parallel edges. This is illustrated in

Figure 2. We define a new graph G' by applying the following two transformations to G , in the given order (see Figure 3):

1. Add an extra vertex v_∞ , and make it adjacent to all vertices of G dual to the unbounded faces of $A(S)$.
2. For all vertices of degree three, perform a Y- Δ transformation. Note that after adding v_∞ , no two vertices of degree three are adjacent. Hence, the transformation is well-defined.

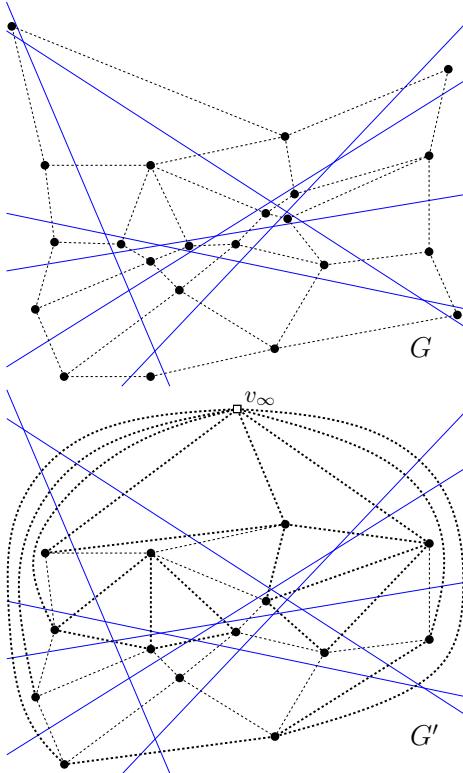


Figure 3: Transforming the dual graph G into G' .

Lemma 12 G' is a four-connected planar graph, hence it is Hamiltonian.

Proof. We need to show that G' does not have any cutset of size three or less. We first have to make the following observation: if C is a cutset of G' , then $C \setminus \{v_\infty\}$ is a cutset of G . To check this, we need to show that the vertices of degree three that are eliminated by the Y- Δ transformations do not reconnect the separated vertex sets in G' . Each such Y- Δ transformation involves three vertices u, v, w that are pairwise adjacent in G' . Hence the degree-three vertex of G that was eliminated can be assigned to the same side of the partition in G as the vertices $\{u, v, w\} \setminus C$.

Now, we rule out the existence of a cutset C of size two in G' , thereby showing that G' is at least three-connected. From our observation, $C \setminus \{v_\infty\}$ would be a

cutset of G . If v_∞ belongs to C then $C \setminus \{v_\infty\}$ has size one, which is impossible since G is known to be two-connected. Otherwise, we have a cutset of size two in G , which from Lemma 9 must be neighbors of a degree-two vertex in G . This vertex, however, after adding v_∞ became of degree three, thus it must have been eliminated by a Y- Δ transformation, hence again C cannot be a cutset in G' .

Now suppose that C is a cutset of size exactly three in G' , and let us first suppose that C does not contain v_∞ . From our observation, C is also a cutset of G , and from Lemma 10 it either (i) consists of the neighbors of a degree-three vertex of G , or (ii) is such that both separated sets contain vertices of the outer face. In case (i), since degree-three vertices of G are eliminated by the Y- Δ transformations, C cannot be a cutset of G' and we have a contradiction. In case (ii), since the vertex subsets separated in G are connected by v_∞ in G' , C cannot be a cutset in G' and we have a contradiction again. Hence we can assume that $v_\infty \in C$, and that $C \setminus \{v_\infty\}$ is a cutset of size two of G . But this case was already ruled out above, hence G' cannot have a cutset of size at most three, and therefore is four-connected. \square

Lemma 13 G' has at least $n^2/6 - 5n/6 + 2$ vertices.

Proof. Recall that G has exactly $N = \binom{n+1}{2} + 1$ vertices. Further, it is known that the maximum number of degree-three interior vertices in G , that is, of bounded triangular faces in the arrangement $A(S)$, is at most $n(n-2)/3$ [12]. Also, the number of degree-two vertices is at most $2n$. These are exactly the vertices that are eliminated by the Y- Δ transformation to obtain G' . Hence the number of vertices of G' is at least $n(n+1)/2 + 2 - (n(n-2)/3) - 2n = n^2/6 - 5n/6 + 2$. \square

Theorem 14 $f(n) \geq n^2/6 - 5n/6$.

Proof. Consider a Hamiltonian cycle in G' . This cycle can be transformed into a simple path of length at least $n^2/6 - 5n/6$ in G , by eliminating v_∞ and replacing every portion of the cycle using one or two edges of a triangle obtained from a Y- Δ transformation by two edges incident to the degree-three vertex (see Figure 4). \square

3.2 Upper Bound

We show that the previous lower bound on the length of the longest simple path in G is within a factor two of the optimum.

Proposition 15 $f(n) \leq n^2/3 + O(n)$.

Proof. It is well-known that the cells of any line arrangement can be properly two-colored, hence that G is a bipartite graph. We will refer to the colors as black and white. Füredi and Palásti [13] give an example of

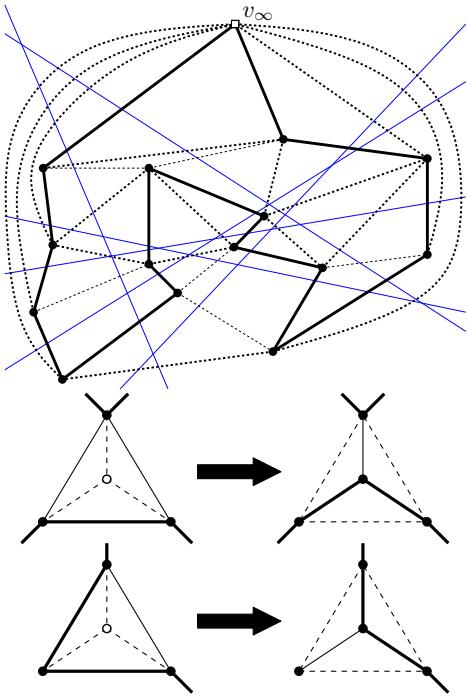


Figure 4: Obtaining a long path in G (operations shown at the bottom) from a Hamiltonian cycle in G' (top).

an arrangement of n lines in which there are $n^2/3 + O(1)$ black cells and $n^2/6 + O(n)$ white cells. Hence, there are roughly twice as many black cells as white cells, which is known to be asymptotically tight [18]. Now observe that any simple path or cycle in G will alternatingly traverse white and black cells, hence cannot have length greater than $n^2/3 + O(n)$. \square

Acknowledgments

This work was initiated during the ComPoSe-Meeting “Workshop on combinatorics of colored point sets” held on Feb. 4–8, 2013 at the University of Seville, Spain. We thank the other participants — J.M. Díaz-Báñez, P. Kamousi, D. Orden, P. Ramos, M. Saumell, W. Steiger, and I. Ventura — for helpful comments.

References

- [1] M. Abellanas, A. García, F. Hurtado, and J. Tejel. Caminos alternantes. *Actas X Encuentros de Geometría Computacional* (in Spanish), Sevilla, 2003, 7–12.
- [2] J. Akiyama, J. Urrutia. Simple alternating path problem. *Discrete Math.* 84:101–103, 1990.
- [3] E. Ackerman, R. Pinchasi. A note on coloring line arrangements. <http://arxiv.org/pdf/1207.0080.pdf>.
- [4] S. Bereg, F. Hurtado, M. Kano, M. Korman, D. Lara, C. Seara, R.I. Silveira, J. Urrutia and K. Verbeek. Balanced partitions of 3-colored geometric sets in the plane Manuscript, 2012. Abstract in 29th EuroCG, 2013.
- [5] P. Bose, J. Cardinal, S. Collette, F. Hurtado, S. Langerman, M. Korman, and P. Taslakian. Coloring and guarding arrangements. Abstract in 28th EuroCG, 2012.
- [6] P. Bose, H. Everett, and S. Wismath. Properties of Arrangements. *Int. J. Comput. Geom.*, 13:447–462, 2003.
- [7] P. Brass, W. Moser and J. Pach. *Research Problems in Discrete Geometry*. Vieweg Verlag, 2004.
- [8] J. Cibulka, J. Kynčl, V. Mészáros, R. Stolař, and P. Valtr. Hamiltonian alternating paths on bicolored double-chains. In: *Graph Drawing 2008, LNCS*, vol. 5417 (2009) 181–192.
- [9] A. Dumitrescu. On some monotone path problems in line arrangements. *Comput. Geom.* 32(1):13–25, 2005.
- [10] S. Felsner. *Geometric Graphs and Arrangements*. Springer, Berlin, 2005.
- [11] S. Felsner, F. Hurtado, M. Noy and I. Streinu. Hamiltonicity and colorings of arrangement graphs. *Discrete Appl. Math.*, 154:2470–2483, 2006.
- [12] S. Felsner and K. Kriegel. Triangles in Euclidean Arrangements. *Discrete Comput. Geom.*, 22:429–438, 1999.
- [13] Z. Füredi and I. Palásti. Arrangements of lines with a large number of triangles. *Proc. Amer. Math. Soc.*, 92:561–566, 1984.
- [14] B. Grünbaum. *Arrangements and Spreads*. Regional Conf. Ser. Math., Amer. Math. Soc., 1972.
- [15] B. Grünbaum. How many triangles? *Geombinatorics*, 8:154–159, 1998.
- [16] B. Grünbaum. Arrangements of colored lines. Abstract 720-50-5, *Notices Amer. Math. Soc.*, 22(1975), A-200.
- [17] B. Grünbaum. Monochromatic intersection points in families of colored lines. *Geombinatorics*, 9:3–9, 1999.
- [18] B. Grünbaum. Two-coloring the faces of arrangements. *Period. Math. Hungar.*, 11(3):181–185, 1980.
- [19] B. Grünbaum and H. Walther. Shortness exponents of families of graphs. *J. Comb. Theory A*, 14:364–385, 1973.
- [20] 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*, pp. 551–570, 2003.
- [21] J. Kynčl, J. Pach and G. Tóth. Long alternating paths in bicolored point sets. *Discrete Math.* 308:4315–4322, 2008.
- [22] J. Leaños, M. Lomelí, C. Merino, G. Salazar, and J. Urrutia. Simple Euclidean arrangements with no (≥ 5)-gons. *Discrete Comput. Geom.*, 38:595–603, 2007.
- [23] E. Lucas. *Récréations mathématiques IV*. Paris, 1894
- [24] V. Mészáros. Extremal problems on planar point sets. PhD Thesis, University of Szeged, 2011.
- [25] J.W. Moon and L. Moser. Simple paths on polyhedra. *Pacific J. Math.*, 13:629–631, 1963.
- [26] T. S. Motzkin. Nonmixed connecting lines. Abstract 67T 605, *Notices Amer. Math. Soc.*, 14(1967), p. 837.
- [27] W. T. Tutte. A theorem on planar graphs. *Trans. Amer. Math. Soc.*, 82:99–116, 1956.

Optimal Data Structures for Farthest-Point Queries in Cactus Networks*

Prosenjit Bose[†]

Jean-Lou De Carufel[†]

Carsten Grimm^{†‡}

Anil Maheshwari†

Michiel Smid†

Abstract

Consider the continuum of points on the edges of a network, i.e., a connected, undirected graph with positive edge weights. We measure the distance between these points in terms of the weighted shortest path distance, called the *network distance*. Within this metric space, we study farthest points and farthest distances. We introduce optimal data structures supporting queries for the farthest distance and the farthest points on trees, cycles, uni-cyclic networks and cactus networks.

1 Introduction

1.1 Problem Definition

We call a simple, finite, undirected graph with positive edge weights a *network*. Unless stated otherwise, we consider only connected networks. Let $G = (V, E)$ be a network with n vertices and m edges, where V is the set of vertices and E is the set of edges. We write uv to denote an edge with endpoints $u, v \in V$ and we write w_{uv} to denote its weight. A point p on edge uv subdivides uv into two sub-edges up and pv with $w_{up} = \lambda w_{uv}$ and $w_{pv} = (1 - \lambda)w_{uv}$, where λ is the real number in $[0, 1]$ for which $p = \lambda u + (1 - \lambda)v$. We write $p \in uv$ when p is on edge uv and $p \in G$ when p is on some edge of G .

As shown in Fig. 1, we measure the distance between points $p, q \in G$ in terms of the weighted length of a shortest path from p to q in G , denoted by $d_G(p, q)$. We say that p and q have *network distance* $d_G(p, q)$. The points on G and the network distance form a metric space. Within this metric space, we study farthest points and farthest distances. We call the largest network distance from some point p on G the *eccentricity* of p and denote it by $\text{ecc}_G(p)$, i.e., $\text{ecc}_G(p) = \max_{q \in G} d_G(p, q)$. A point \bar{p} on G is farthest from p if and only if $d_G(p, \bar{p}) = \text{ecc}_G(p)$. We omit the subscript G whenever the underlying network is understood.

We aim to construct data structures for a fixed network G supporting the following queries. Given a point p on G , what is the eccentricity of p ? What is the set of farthest points from p in G ? We refer to the former as an *eccentricity query* and to the latter as a *farthest-*

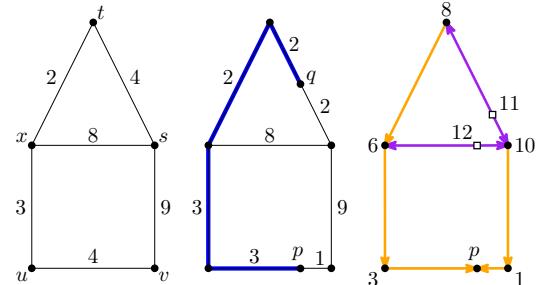


Figure 1: From left to right: (a) a network G (b) the network distance from $p = \frac{1}{4}u + \frac{3}{4}v$ to $q = \frac{1}{2}s + \frac{1}{2}t$ is $d_G(p, q) = 10$ (c) a shortest path tree rooted at p (orange¹) and its extension (orange + purple). We have $\text{ecc}(p) = 12$ and the farthest point from p is on xs .

point query. Both queries consist of the query point p represented by the edge uv containing p and the value $\lambda \in [0, 1]$ such that $p = \lambda u + (1 - \lambda)v$. We study trees, cycles, uni-cyclic networks and cactus networks. A *uni-cyclic network* is a network with exactly one simple cycle and a *cactus network* is a network where no two simple cycles share an edge.

1.2 Related Work

The problem of determining farthest points has been encountered [1, 2] when studying farthest-point Voronoi diagrams on networks. Specifically, when all of the infinitely many points on a network are considered sites. This point of view leads to a data structure with construction time $O(m^2 \log n)$ and size $O(m^2)$ supporting eccentricity queries and farthest point queries on arbitrary networks in optimal time [1, 2].

This work has connections to center problems [12, 11]. In a tree network, the set of farthest points changes only at its *absolute center* [4]. An *absolute center* is a point c on a network $G = (V, E)$ whose farthest vertices are as close as possible, i.e., $\max_{v \in V} d(c, v) = \min_{q \in G} \max_{v \in V} d(q, v)$. There are linear time algorithms for finding an absolute center in trees [6], unicyclic networks [5], and cactus networks [10]. The algorithm by Hämäläinen [5] plays an important role when we study uni-cyclic networks. We use the decomposition of a network into its tree structure like many works about center problems [9]. Tansel [12] and Kincaid [9] provide comprehensive surveys about center problems.

*Research supported in part by FQRNT and NSERC.

[†]School of Computer Science, Carleton University

[†]School of Computer Science, Carleton University
[‡]Institut für Simulation und Graphik, Fakultät für Informatik,
Otto-von-Guericke-Universität Magdeburg

¹Due to the limitations of the printed proceedings, please refer to the online version for colors in figures.

1.3 Our Contributions

We introduce optimal data structures supporting eccentricity queries and farthest-points queries for trees, cycles, uni-cyclic networks and cactus networks. The query times are summarized in Tab. 1. All of the presented data structures have linear construction time and, thus, require only linear space.

Type	Eccentricity	Farthest-Points
Tree	$O(1)$	$O(k)$
Cycle	$O(\log n)$	$O(\log n)$
Uni-Cyclic	$O(\log n)$	$O(k + \log n)$
Cactus	$O(\log n)$	$O(k + \log n)$

Table 1: The query times for different types of networks, where k is the number of reported farthest points.

In Section 2, we introduce data structures for trees, cycles and uni-cyclic networks. In Section 3, we construct data structures supporting eccentricity queries and farthest-point queries on cactus networks. Our approach is to reduce a cactus network to smaller networks having a sufficiently simple structure such that the query algorithms of Section 2 can be applied.

2 Trees, Cycles, and Uni-Cyclic Networks

2.1 Trees

Let T be a tree network. We call a point c on T whose farthest points are closest, a center of T , i.e., $\text{ecc}(c) = \min_{x \in T} \text{ecc}(x)$. A tree has exactly one center and we can find this center in linear time [6].

Lemma 1 *Let T be a tree, and let p be a point on T . All farthest points from p are leaves and any path from p to a farthest leaf contains the center of T .*

Corollary 2 *Let T be a tree with center c . For all points p on T we have $\text{ecc}(p) = d(p, c) + \text{ecc}(c)$.*

Splitting a tree T at its center c yields sub-trees with common farthest points, as shown in Fig. 2. When c is on edge uv with $u \neq c \neq v$, we split T into two sub-trees: the sub-tree T_u , containing the sub-edge uc , and the sub-tree T_v containing the sub-edge cv . The points on T_u (except for c) have all farthest points in T_v . The farthest points in c are those points that are farthest from T_u in T_v and those farthest from T_v in T_u .

Lemma 3 *Let T be a tree with center c , and let T' be one of the sub-trees obtained by splitting T at c . Leaf $l \in T'$ is farthest from $p \in T \setminus T'$ if and only if l is farthest from c , i.e., $\text{ecc}_T(p) = d_T(l, p) \iff \text{ecc}_T(c) = d_T(l, c)$.*

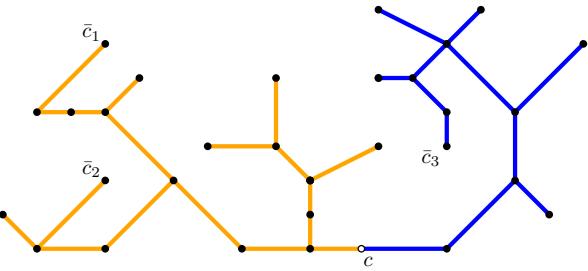


Figure 2: A tree T with geometric edge weights. The center c splits T into two sub-trees. For every point on the left sub-tree (orange) \bar{c}_3 is farthest and for every point on the right sub-tree (blue) \bar{c}_1 and \bar{c}_2 are farthest.

Using Corollary 2 and Lemma 3, we support eccentricity queries and farthest point queries in tree networks: Let T be a tree network with center c . We compute the position of c and the distances $d(c, v)$ for each vertex v of T . The maximum encountered distance is the eccentricity of c . Let T_1, T_2, \dots, T_r be the sub-trees obtained by splitting T at c . For each sub-tree, we store the set of farthest leaves from c in T_i , denoted by L_i , i.e., $L_i = \{l \in T_i \mid d(l, c) = \text{ecc}(c)\}$. For an eccentricity query from point p on edge uv of T with $d(u, c) < d(v, c)$, we have $\text{ecc}(p) = w_{up} + d(u, c) + \text{ecc}(c)$. For a farthest-point query from p with $p \neq c$ and $p \in T_i$, we report all leaves in each L_j with $j \neq i$; for a farthest-point query from c we report the union of all L_i .

Theorem 4 *Let T be a tree network with n vertices. There is a data structure with construction time $O(n)$ supporting eccentricity queries on T in constant time and farthest-point queries on T in $O(k)$ time, where k is the number of reported farthest points.*

2.2 Cycles

Let C be a cycle network and let w_C be the sum of all edge weights of C . Each point p on C has exactly one farthest point \bar{p} located on the opposite side of C with $\text{ecc}(p) = d(p, \bar{p}) = w_C/2$. Supporting eccentricity queries on C amounts to calculating and storing $w_C/2$.

To support farthest-point queries, we compute the farthest-point \bar{v} of each vertex v , subdivide the edge st containing \bar{v} at \bar{v} , and introduce pointers between v and \bar{v} . We perform this computation as illustrated in Fig. 3: First, we compute the farthest point \bar{v} for some initial vertex v by walking a distance of $w_C/2$ from v along C . Then, we sweep a point p from position $p = v$ to position $p = \bar{v}$ along C while maintaining the farthest point \bar{p} . During this sweep we subdivide C at p whenever \bar{p} hits a vertex and at \bar{p} whenever p hits a vertex. We store the distance from v to any other vertex, which enables us to compute the distance of any pair of vertices in constant time. The entire sweep takes linear time, thus, the resulting data structure occupies linear space.

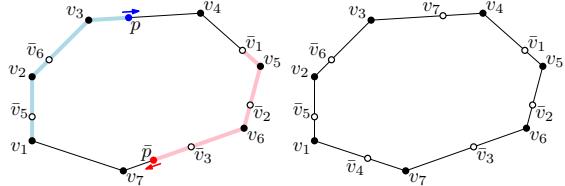


Figure 3: From left to right: (a) a sweep along cycle C starting from $p = v_1$ and (b) the resulting subdivision of C . The farthest point from any point on sub-edge $v_5\bar{v}_2$ is located on the sub-edge v_5v_2 .

With the subdivided network, we can answer farthest-point queries in constant time, provided we know the sub-edge containing the query point p : When p is located on sub-edge $\bar{x}\bar{y}$ with $p = \mu\bar{x} + (1 - \mu)\bar{y}$ for some $\mu \in [0, 1]$ then \bar{p} is located on xy with $\bar{p} = \mu x + (1 - \mu)y$. The query point p is represented by the edge uv containing p and the value $\lambda \in [0, 1]$ such that $p = \lambda u + (1 - \lambda)v$. Using a binary search, we determine the sub-edge containing p and the value μ . This takes $O(\log n)$ time, since we subdivide each edge at most n times.

Lemma 5 *Given a cycle network C with n vertices. There is a data structure with construction time $O(n)$ supporting eccentricity queries on C in constant time and farthest-point queries on C in $O(\log n)$ time.*

2.3 Uni-cyclic Networks

As shown in Fig. 4, a *uni-cyclic* network U consists of a cycle C and trees T_1, T_2, \dots, T_r , called the *branches*, attached to C at vertices v_1, v_2, \dots, v_r , respectively.

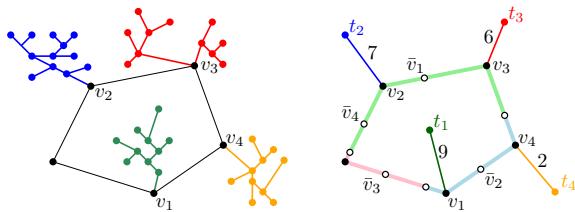


Figure 4: From left to right: (a) A unicyclic network with four branches (coloured) attached to its cycle. (b) The same network with compressed branches. The colouring of the cycle indicates the farthest branch.

Our data structure for uni-cyclic networks consists of three components: a data structure for queries on the cycle C that yields the farthest point among the points on C , a data structure for queries on C that yields the branches containing farthest points, and data structures for queries on the branches. The first component is the data structure from Section 2.2 supporting queries on C . The second component is a data structure supporting *farthest-branch queries*, i.e., queries for the branches containing farthest points from a query point on C . The second component uses the following simplification of U .

We replace each branch T_i of U with a vertex t_i and an edge t_iv_i , where v_i is the vertex connecting T_i to C . The weight of t_iv_i is the farthest distance from v_i in T_i , i.e., $w_{t_iv_i} = \text{ecc}_{T_i}(v_i)$. In the resulting network S , vertex t_i is farthest from p if and only if T_i contains farthest points from p with respect to U , i.e.,

$$d_S(p, t_i) = \text{ecc}_S(p) \iff \exists q \in T_i: d_U(p, q) = \text{ecc}_U(p).$$

We call a vertex t_i *relevant* if there exists a point p on C who has t_i as a farthest vertex among t_1, t_2, \dots, t_r , i.e., $d_S(t_i, p) = \max_{j=1}^r d_S(t_j, p)$. Recall that \bar{v}_i denotes the farthest point from v_i among all points on C .

Lemma 6 *Vertex t_i is relevant if and only if t_i is farthest from \bar{v}_i among t_1, t_2, \dots, t_r .*

Lemma 6 yields a certificate for irrelevance. We say that t_j *dominates* t_i , and write $t_i \prec t_j$, if $d_S(t_i, \bar{v}_i) < d_S(t_j, \bar{v}_i)$. When t_j dominates t_i , all points on C are closer to t_i than to t_j and, thus, t_i cannot be relevant. Conversely, a vertex is relevant if and only if there is no other vertex dominating it. For the following, assume we have a circular list storing t_1, t_2, \dots, t_r ordered as v_1, v_2, \dots, v_r appear along the cycle.

Lemma 7 *Let t_a be the first relevant vertex after t_i and let t_b be the first relevant vertex before t_i . Vertex t_i is relevant if and only if neither t_a nor t_b dominate t_i , i.e., if and only if $t_i \not\prec t_a$ and $t_i \not\prec t_b$.*

Algorithm 1 computes the relevant vertices in $O(r)$ time using Lemma 7. We begin with a circular list containing all vertices t_1, t_2, \dots, t_r . We remove irrelevant vertices from this list until no vertex in the list is dominated by its predecessor or successor. In each iteration of the while-loop we either delete some vertex or we mark the current t as processed ensuring that it will never assume the role of t again. Thus, the claim about the running time follows. Hämäläinen [5] uses a variant of Algorithm 1 in his linear time algorithm for finding the absolute center of a uni-cyclic network.

Algorithm 1: Determining the relevant vertices

```

input : A circular list  $L$  containing  $t_1, t_2, \dots, t_r$ .
output: A sub-list of  $L$  containing only the relevant
         vertices among  $t_1, t_2, \dots, t_r$ .
1 Mark each  $t_1, t_2, \dots, t_r$  as unprocessed;
2  $t \leftarrow t_1$ ;
3 while  $t$  is unprocessed do
4   if  $t \prec \text{pred}(t)$  or  $t \prec \text{succ}(t)$  then
5      $t \leftarrow \text{succ}(t)$ ;
6      $\text{delete}(\text{pred}(t))$ ;
7   else if  $\text{pred}(t) \prec t$  then  $\text{delete}(\text{pred}(t))$ ;
8   else if  $\text{succ}(t) \prec t$  then  $\text{delete}(\text{succ}(t))$ ;
9   else ( $t \not\prec \text{pred}(t)$  and  $t \not\prec \text{succ}(t)$ )
10    | Mark  $t$  as processed;
11    |  $t \leftarrow \text{succ}(t)$ ;
12  end
13 end

```

The relevant vertices induce a subdivision of C into regions with a common farthest vertex among t_1, t_2, \dots, t_r . When walking along C , we encounter these regions in the same order as the corresponding relevant points. Given the relevant vertices, we can compute the subdivision in linear time. Storing the relevant vertex with each sub-edge reduces a query for the farthest vertices among t_1, t_2, \dots, t_r to a binary search. We query for branches containing farthest points using the subdivision and our data structure for the cycle C .

Lemma 8 *Let U be a uni-cyclic network with n vertices and cycle C . There is a data structure with construction time $O(n)$ supporting farthest-branch queries in U from points on the cycle C in time $O(b + \log n)$, where b is the number of reported branches.*

Lemma 8 concludes the description of the second component of our data structure for uni-cyclic networks.

The third component is a data structure supporting queries on branches. Consider a branch T that is attached to the cycle C at vertex v . We extend T by a vertex v' and an edge vv' whose weight is the farthest distance from v to any point outside of T , i.e., $w_{vv'} = \text{ecc}_{U \setminus T}(v)$. The resulting tree T' , preserves farthest distances with respect to U , i.e., we have $\text{ecc}_U(p) = \text{ecc}_{T'}(p)$ for all $p \in T$. Thus, we can use the data structure from Section 2.1 to support eccentricity queries in U from points on T . Furthermore, if a point q outside of T has a farthest point \bar{q} on T , then \bar{q} is also farthest from v' in T' . When a farthest-branch query from q returns T , we report the farthest points from q in T with a farthest-point query from v' in T' .

A farthest point query in T' from a point $p \in T$ yields the farthest points from p on T and the vertex v' when p has farthest points outside of T . If v' is reported as a farthest point, we check whether \bar{v} , the farthest point from v on C is farthest from p . We determine the branches containing farthest points from p with a farthest-branch query at v and then report the farthest points from p in these branches as described above.

The above procedure for farthest point queries from T works correctly, unless the farthest branch query from v returns only T itself. This situation occurs for at most one branch of U , because it implies that all points on C have T as their only farthest branch. We resolve this issue by removing T from U and computing the farthest branches from v in the resulting network.

Theorem 9 *Let U be a uni-cyclic network with n vertices. There is a data structure with construction time $O(n)$ supporting eccentricity queries on U in $O(\log n)$ time and farthest-point queries on U in $O(k + \log n)$ time, where k is the number of reported farthest points.*

3 Cactus Networks

In this section, we construct a data structure supporting eccentricity queries and farthest-point queries on cactus networks. Recall that a cactus networks is a network where no two simple cycles share an edge. A *cut-vertex* is a vertex whose removal increases the number of connected components and a *bi-connected component* is a maximal connected sub-network without cut-vertices.

In linear time [8], we can decompose any network G into connected sub-networks B_1, B_2, \dots, B_b such that

- each edge of G is contained in exactly one B_i
- each B_i is a bi-connected component of G or the union of bi-connected components of G ,
- and each vertex contained in more than one sub-network is a cut-vertex of G .

We call this a *block decomposition* of G into *blocks* B_1, B_2, \dots, B_b . We call a cut-vertex contained in more than one block a *hinge vertex* [3]. For cactus networks, we consider the block decomposition where each block is a simple cycle or one of the (non-trivial) trees that remain when removing all cycles, as shown in Fig. 5.

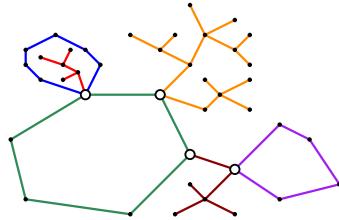


Figure 5: A cactus network decomposed into six blocks (coloured) with four hinge vertices (empty discs).

The following terms describe how we subdivide a network G with respect to a block decomposition; examples are shown in Fig. 6. For a sub-network S of G , we write $G - S$ to denote the network resulting from removing all edges of S from G (without removing any vertices). For a block B and a hinge vertex $h \in B$, we call the connected component of $G - B$ containing h the *block-cut* of B at h , denoted by $\text{bcut}(B, h)$. We call the connected component of $G - \text{bcut}(B, h)$ containing h the *co-block-cut* of B at h , denoted by $\text{co-bcut}(B, h)$.

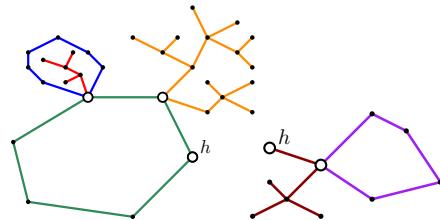


Figure 6: For the network from Fig. 5, from left to right: (a) the block-cut of the brown block at hinge vertex h and (b) the corresponding co-block-cut.

3.1 Eccentricity Queries

Consider a block B of a network G . To support eccentricity queries on B , we compress the (non-trivial) connected components of $G - B$ like we compress the branches of uni-cyclic networks. For each hinge vertex $h \in B$ we replace $\text{bcut}(B, h)$ with a vertex \hat{h} and an edge $h\hat{h}$ whose weight is the largest distance from h to any point in $\text{bcut}(B, h)$, i.e., $w_{h\hat{h}} = \text{ecc}_{\text{bcut}(B, h)}(h)$. We refer to the resulting network as the *locus* of B , denoted by $\text{loc}(B)$. The locus of block B preserves farthest distances of G , i.e., $\text{ecc}_{\text{loc}(B)}(p) = \text{ecc}_G(p)$ for all p on B .

We begin at some block B^* of a cactus network. For each hinge $h^* \in B^*$, we compute $\text{ecc}_{\text{bcut}(B^*, h^*)}(h^*)$ with a modified breadth-first-search in linear time. This breadth-first-search also yields the farthest distances along paths leading away from B^* , i.e., we obtain $\text{ecc}_{\text{bcut}(B, h)}(h)$ for any $\text{bcut}(B, h) \subseteq \text{bcut}(B^*, h^*)$.

Let B' be a block neighboring B^* at hinge vertex h , as shown in Fig. 7. To construct $\text{loc}(B')$, we only lack the farthest distance from h in $\text{co-bcut}(B^*, h)$. We obtain this value with an eccentricity query in $\text{loc}(B^*)$ via

$$\text{ecc}_{\text{co-bcut}(B^*, h)}(h) = \text{ecc}_{\text{loc}(B^*)}(\hat{h}) - w_{h\hat{h}},$$

where \hat{h} represents $\text{bcut}(B^*, h)$ in $\text{loc}(B^*)$. This way we obtain the loci of all neighbors of B^* , then all loci of the neighbors of all neighbors of B^* and so forth.

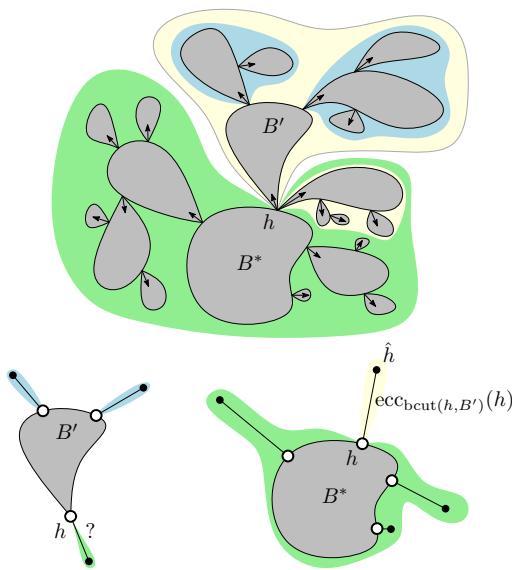


Figure 7: Top down and left to right: (a) An abstraction of the block structure of a network. The arrows indicate shortest path trees emanating from block B^* . (b) When constructing the locus of block B' we lack the distance from $\text{bcut}(B', h)$ (green) whereas the distances from all other block cuts (blue) are known. (c) We obtain the missing distance with an eccentricity query in $\text{loc}(B^*)$.

Constructing a data structure supporting eccentricity queries on a locus takes linear time in the size of the locus. Recall that each locus of a cactus network is either a tree or a uni-cyclic network. The eccentricity queries in a neighboring block take constant time, due to Theorem 9. Therefore, our data structure for eccentricity queries in cactus networks has construction time $O(n)$ and inherits the query times from uni-cyclic networks.

Lemma 10 *Let G be a cactus network with n vertices. There is a data structure with construction time $O(n)$ supporting eccentricity queries on G in $O(\log n)$ time.*

3.2 Farthest-Point Queries

To answer a farthest-point query from a point p in block B , we perform a farthest-point query in the locus $\text{loc}(B)$ and then cascade the query into the neighboring blocks. If the query from p in $\text{loc}(B)$ returns vertex \hat{h} representing $\text{bcut}(B, h)$, then $\text{bcut}(B, h)$ contains farthest points from p . From the construction of $\text{loc}(B)$, we know which blocks neighboring B at h lie on a path to from p to one of its farthest points. We continue with a farthest-point query from \hat{h} in the loci of these blocks. This takes $O(n)$ time, since we might cascade through $O(n)$ blocks until we reach one containing farthest points from p . We improve the query time by using shortcuts to skip long chains of blocks without farthest points.

We define the *tree structure* [7] of a block decomposition of a network G , denoted by T_G , as the following graph. The set of vertices of T_G consists of the blocks of G and the hinge vertices of G . The edges of T_G connect a hinge vertex h and a block B whenever $h \in B$. Since the tree structure is indeed a tree [7] and since there are at most n blocks and at most n cut-vertices in a network with n vertices, T_G has at most $2n - 1$ edges.

The blocks and the hinge vertices visited during a cascading farthest-point query form a sub-tree T_{query} of the tree structure T_G . All farthest points from the query point are located in blocks that occur as vertices of T_{query} . Next, we use path compression to obtain a version of T_{query} whose size is linear in the number of blocks containing farthest points from the query point.

Consider an edge $\{h, B\}$ in T_G and the paths from h to blocks containing farthest points from h with respect to $\text{co-bcut}(B, h)$. We store a shortcut from $\{h, B\}$ to the first edge $\{\hat{h}, B'\}$ along these paths, where B' contains farthest points or two paths split at B' . Fig. 8 shows a farthest-point query using one of these shortcuts. There are $O(n)$ shortcuts in total, since we add at most one shortcut per edge of T_G and since T_G has $O(n)$ edges.

We obtain the shortcuts leading away from B^* as a byproduct of the breadth-first-search used in the construction of the locus of B^* . For the remaining shortcuts, we rely on a similar strategy as used to obtain the loci of all blocks B with $B \neq B^*$. Let B be a block neighboring B^* at h . We introduce no shortcut

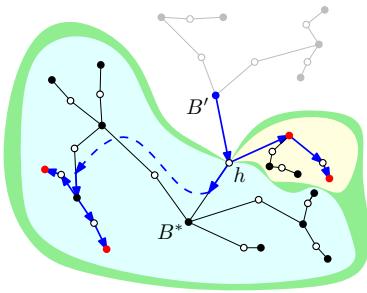


Figure 8: A farthest point query (blue) from block B' reporting the farthest points in $\text{bcut}(B', h)$ (green) using a shortcut (dashed). Blocks containing farthest points are indicated in red. An arc from a block B to a hinge vertex h indicates that we continue reporting farthest points in $\text{bcut}(B, h)$. An arc from h to B indicates that we continue reporting farthest points in $\text{co-bcut}(B, h)$.

when B^* contains farthest points from \hat{h} or when two paths to farthest points from h in $\text{co-bcut}(B^*, h)$ split in B^* or at some hinge vertex of B^* . Otherwise, B^* has one neighboring block B' at hinge vertex h' such that $\text{co-bcut}(B', h')$ contains all farthest points from h in $\text{co-bcut}(B^*, h)$. In this case, we add a shortcut from $\text{co-bcut}(B^*, h)$ to the destination of the shortcut from $\text{co-bcut}(B', h')$. Since we conduct farthest point queries only on pendant edges of the loci, it takes constant time to determine which case applies and the overall construction time for cactus networks is $O(n)$.

Let p be a point in block B with k farthest points. During a farthest-point query from p , we report all farthest points from p in B and all block-cuts containing farthest points with a query in $\text{loc}(B)$. This takes $O(k + \log n)$ time due to Theorem 9. We follow the shortcuts associated to the reported block-cuts and obtain all other blocks containing farthest points in $O(k)$ time. For each reported block B' we perform a farthest-point query from a pendant vertex of $\text{loc}(B')$. By Theorem 4, this takes linear time in the number of farthest points in B' . The overall query time is $O(k + \log n)$.

Theorem 11 *Let G be a cactus network with n vertices. There is a data structure with construction time $O(n)$ supporting eccentricity queries on G in $O(\log n)$ time and farthest-point queries in $O(k + \log n)$ time, where k is the the number of reported farthest points.*

4 Conclusions and Future Work

In previous work [1, 2], we introduce a data structure with optimal query times for eccentricity and farthest-point queries and construction time $O(m^2 \log n)$ for any network with n vertices and m edges. In this work, we improve the construction time to $O(n)$ for certain classes of networks without sacrificing query time. In future work, we aim to achieve $o(m^2 \log n)$ construction time for more classes of networks.

References

- [1] P. Bose, J.-L. D. Carufel, C. Grimm, A. Maheshwari, and M. Smid. On farthest-point information in networks. In *Proceedings of the 24th Canadian Conference on Computational Geometry*, pages 199–204, 2012.
- [2] P. Bose, K. Dannies, J.-L. De Carufel, C. Doell, C. Grimm, A. Maheshwari, S. Schirra, and M. Smid. Network Farthest-Point Diagrams. *ArXiv*, Apr. 2013.
- [3] R. E. Burkard and J. Krarup. A linear algorithm for the pos/neg-weighted 1-median problem on a cactus. *Computing*, 60(3):193–215, 1998.
- [4] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964.
- [5] P. Hämäläinen. The absolute center of a unicyclic network. *Discrete Appl Math*, 25(3):311 – 315, 1989.
- [6] G. Y. Handler. Minimax location of a facility in an undirected tree graph. *Trans. Sci.*, 7(3):287–293, 1973.
- [7] F. Harary and G. Prins. The block-cutpoint-tree of a graph. *Publ. Math. Debrecen*, 13:103–107, 1966.
- [8] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372–378, 6 1973.
- [9] R. K. Kincaid. Exploiting structure: Location problems on trees and treelike graphs. In *Foundations of Location Analysis*, pages 315–334. Springer US, 2011.
- [10] Y.-F. Lan, Y.-L. Wang, and H. Suzuki. A linear-time algorithm for solving the center problem on weighted cactus graphs. *IPL*, 71(5–6):205–212, 1999.
- [11] Q. Shi. *Efficient algorithms for network center/covering location optimization problems*. PhD thesis, School of Computing Science-Simon Fraser University, 2008.
- [12] B. Ç. Tansel. Discrete center problems. In *Foundations of Location Analysis*, pages 79–106. Springer US, 2011.

Cole’s Parametric Search Technique Made Practical

Michael T. Goodrich

Dept. of Computer Science
University of California, Irvine

Pawel Pszona

Dept. of Computer Science
University of California, Irvine

Abstract

Parametric search has been widely used in geometric algorithms. Cole’s improvement provides a way of saving a logarithmic factor in the running time over what is achievable using the standard method. Unfortunately, this improvement comes at the expense of making an already complicated algorithm even more complex; hence, this technique has been mostly of theoretical interest. In this paper, we provide an algorithm engineering framework that allows for the same asymptotic complexity to be achieved probabilistically in a way that is both simple and practical (i.e., suitable for actual implementation). The main idea of our approach is to show that a variant of quicksort, known as *boxsort*, can be used to drive comparisons, instead of using a sorting network, like the complicated AKS network, or an EREW parallel sorting algorithm, like the fairly intricate parallel mergesort algorithm. This results in a randomized optimization algorithm with a running time matching that of using Cole’s method, with high probability, while also being practical. We show how this results in practical implementations of some geometric algorithms utilizing parametric searching and provide experimental results that prove practicality of the method.

1 Introduction

Parametric search [24] has proven to be a useful technique in design of efficient algorithms for many geometric and combinatorial optimization problems (e.g., see [2, 3, 28]). Example applications include ray shooting [1], slope selection [13], computing the Fréchet distance between two polygonal curves [6, 8], matching drawings of planar graphs [5], labeling planar maps with rectangles [22], and various other matching and approximation problems (e.g., see [15, 16, 17]).

Although it has been superseded in some applications by Chan’s randomized optimization technique [9, 10], for many problems asymptotically best known results still depend on parametric searching.

The technique is applied to a decision problem, B , whose solution depends on a real parameter, λ , in a monotonic way, so that B is true on some interval $(-\infty, \lambda^*)$. The goal is to determine the value of λ^* ,

the maximum for which B is true. To achieve this goal, the parametric search approach utilizes two algorithms. The first algorithm, \mathcal{C} , is a sequential *decision* algorithm for B that can determine if a given λ is less than, equal to, or greater than λ^* . The second algorithm, \mathcal{A} , is a *generic* parallel algorithm whose inner workings are driven by “comparisons,” which are either independent of λ or depend on the signs of low-degree polynomials in λ . Because \mathcal{A} works in parallel, its comparisons come in batches, so there are several independent such comparisons that occur at the same time. The idea, then, is to run \mathcal{A} on the input that depends on the unknown value λ^* , which will result in actually finding that value as a kind of by-product (even though we do not know λ^* , \mathcal{C} can be used to resolve comparisons that appear during the execution of \mathcal{A}). The next step is to simulate an execution of \mathcal{A} sequentially. To resolve comparisons that occur in a single step of this simulation, we can use the algorithm \mathcal{C} to perform binary search among the (ordered) roots of the polynomials in λ for these comparisons, which allows us to determine signs of all these polynomials, hence, allows us to continue the simulation. When the simulation completes, we will have determined the value of λ^* . Moreover, the running time for performing this simulation is $O(P(n)T(n) + C(n)T(n) \log P(n))$, where $C(n)$ is the (sequential) running time of \mathcal{C} , $T(n)$ is the (parallel) running time of \mathcal{A} , and $P(n)$ is the number of processors used by \mathcal{A} .

Cole [11] shows how to improve the asymptotic performance of the parametric search technique when sorting is the problem solved by \mathcal{A} . His improvement comes from an observation that performing a separate binary search for each step of the algorithm \mathcal{A} will often “waste” calls to \mathcal{C} to resolve a relatively small number of comparisons. Rather than resolve all the comparisons of a single step of \mathcal{A} , he instead assumes that \mathcal{A} is implemented as the AKS sorting network [4] or an optimal EREW parallel sorting algorithm [12, 18], which allows for comparisons on multiple steps of \mathcal{A} to be considered at the same time (so long as their preceding comparisons have been resolved). This improvement results in a running time for the optimization problem that is $O(P(n)T(n) + C(n)(T(n) + \log P(n)))$.

From an algorithm engineering perspective, the “clas-

“parametric” parametric search technique (utilizing a parallel algorithm) is admittedly difficult to implement, although some implementations do exist [29, 30, 31]. Cole’s improvement is even more complex, however, and we are not familiar with any implementations of his parametric search optimization.

Even without Cole’s improvement, a challenge for implementing the parametric search technique is the simulation of a parallel algorithm on a sequential machine. This difficulty has motivated some researchers to abandon the use of parametric searching entirely and instead use other paradigms, such as expander graphs [21], geometric random sampling [23], and ϵ -cuttings [7] (see also [2]).

Interestingly, van Oostrum and Veltkamp [31] show that, for sorting-based parametric search applications, one can use the well-known **quicksort** algorithm to drive comparisons instead of a parallel sorting algorithm. Unfortunately, as van Oostrum and Veltkamp note in their paper, Cole’s improvement cannot be applied in this case. The main difficulty is that, when viewed as a kind of parallel algorithm, comparisons to be done at one level of **quicksort** become known only after all the comparisons on the level above have been resolved. Thus, comparisons cannot be pipelined in the way required by Cole’s optimization when using this approach. The result, of course, is that this sets up an unfortunate tension between theory and practice, forcing algorithm designers to choose between a practical, but asymptotically inferior, implementation or an impractical algorithm whose running time is asymptotically better by a logarithmic factor.

1.1 Our Results

We show that it is, in fact, possible to implement Cole’s parametric search technique in a manner that is efficient and practical (i.e., fast and easy to implement). The main idea is to use a variant of quicksort, known as **boxsort** [26], to drive comparisons (instead of sorting networks, like the complicated AKS network or an EREW parallel sorting algorithm). We apply a potential function to comparisons in the **boxsort** algorithm, which, together with a weighted-median-finding algorithm, allows us to schedule these comparisons in a pipelined fashion and achieve, with high probability, the same asymptotic running time as Cole’s method, while also being practical. Moreover, we provide experimental results that give empirical evidence supporting these claims for the “median-of-lines” problem [24] and the geometric optimization problems of matching planar drawings [5] and labeling planar maps with rectangles [22].

2 Parametric Search Explained

In this section, we provide a more in-depth description of the parametric search technique. Recall that B is a problem that we want to solve. Furthermore, we restrict ourselves to the case where the generic algorithm \mathcal{A} is a sorting algorithm. We require of B the following.

1. There is a *decision algorithm*, \mathcal{C} , which, for any value λ , resolves a comparison $\lambda < \lambda^*$ in time $C(n)$ without actually knowing λ^* (note that $C(n)$ is a function of the size of input to B). Typically, $C(n)$ is at least $\Omega(n)$, as opposed to $O(1)$ comparison time which is usual for classical sorting algorithms.
2. There is an efficient way of generating values x_i (with each x_i being either a real value or a real-valued function of λ) from an input to problem B . Ideally, it produces $O(n)$ such values.
3. For each $x_i < x_j$ comparison, the answer is determined by the sign of a low-degree polynomial in λ at $\lambda = \lambda^*$ (polynomials for different comparisons may differ).
4. *Critical values* (values λ that, based on combinatorial properties of B , have the potential of being equal to λ^*) form a subset of the set of roots of the polynomials determining answers to every possible comparison $x_i < x_j$.

Then, as a by-product of sorting values x_i , we get (directly or indirectly) the answers to all comparisons $\lambda < \lambda^*$, where λ ’s are roots of all comparisons $x_i < x_j$. Therefore, we are able to find λ^* .

We can solve B in the following way: generate x_i ’s, sort them using algorithm \mathcal{A} and recover λ^* from the answer. If \mathcal{A} sorts n items in $T(n)$ comparisons and each comparison is resolved in time $O(C(n))$ (it requires determining whether $\lambda < \lambda^*$ for a constant number of roots λ), solving B this way takes time $T(n)C(n)$.

It is important to note that if there are k comparisons $x_i < x_j$, we can avoid calling \mathcal{C} on every single root of their polynomials, and still resolve them all. This is because resolving $\lambda < \lambda^*$ automatically resolves comparisons for values $\lambda' \leq \lambda$ (if the result was YES) or $\lambda'' > \lambda^*$ (if the result was NO). Therefore, we can solve k comparisons in only $O(\log k)$ calls to \mathcal{C} , if in every iteration we use a standard median-finding algorithm (e.g., see [14]) to find the median root λ , and then resolve it by a call to \mathcal{C} (each iteration halves the number of unresolved comparisons).

The above observation lies at the heart of the original parametric search, as introduced by Megiddo [24]. Note that we can group the comparisons in such a way only if they are *independent* of each other. To assure this, one chooses \mathcal{A} to be a *parallel* sorting algorithm, running in

time $T(n)$ on $P(n)$ processors. At every step of \mathcal{A} , there are $O(P(n))$ independent comparisons, and they can be resolved in time $O(P(n) + \log(P(n)) \cdot C(n))$ according to the previous observation. Resolving comparisons at all $T(n)$ steps of \mathcal{A} takes time $O(T(n) \cdot P(n) + T(n) \cdot \log(P(n)) \cdot C(n))$. Simulating \mathcal{A} on a sequential machine takes time $O(T(n)P(n))$. Therefore, parametric search, as originally introduced, helps solve B in time $O(T(n) \cdot P(n) + T(n) \cdot \log(P(n)) \cdot C(n))$.

2.1 Cole's Improvement

Cole [11] was able to improve on Megiddo's result by using a sorting network or an EREW parallel sorting algorithm as \mathcal{A} , and changing the order of comparison resolution by assigning weights to comparisons and resolving the *median weighted comparison* at each step.

In the case of a sorting network, a straightforward notion of *active* comparisons and *active* wires was introduced. Initially, all input wires (and no others) are *active*. A comparison is said to be *active* if it is not resolved and both its input wires are *active*. When active comparison gets resolved, its output wires now become *active*, possibly activating subsequent comparisons. Informally, *active* comparisons have not been resolved yet, but both of their inputs are already determined.

Weight is assigned to every comparison, being equal to 4^{-j} for a comparison at depth j . The *active weight* is defined as the weight of all *active* comparisons. The weighted median comparison can be found in $O(n)$ time [27], and resolving it automatically resolves a weighted half of the comparisons.

It is shown that for a sorting network of width $P(n)$ and depth $T(n)$, or an EREW sorting algorithm with $P(n)$ processors and time $T(n)$, the method of resolving weighted median comparison requires only $O(T(n) + \log(P(n)))$ direct calls to \mathcal{C} . Including simulation overhead, we solve B in time $O(P(n) \cdot T(n) + (T(n) + \log(P(n))) \cdot C(n))$.

This is completely impractical, however, as the bounds for the AKS network have huge constant factors. In a subsequent work [12], Cole shows that one can substitute an EREW parallel sorting algorithm for the AKS network, which makes using his optimization more implementable, but arguably still not practical, since the existing optimal EREW parallel sorting algorithms [12, 18] are still fairly intricate.

2.2 Applying quicksort to Parametric Search

Van Oostrum and Veltkamp [31] have shown that the `quicksort` algorithm [20] can be used as \mathcal{A} . Recall that in the randomized version of this algorithm we sort a set of elements by picking one of them (called the *pivot*) at random, and recursively sorting elements smaller than the pivot and greater than the pivot. A

key observation here is that all the comparisons with the pivot(s) at a given level of recursion are independent of each other. It leads to a practical algorithm, running in $O(n \log n + \log^2 n \cdot C(n))$ expected-time, for solving B (it becomes $O(n \log n + \log n \cdot C(n))$ under additional assumption about distribution of the roots of polynomials). Comparisons are resolved by resolving the median comparison among unresolved comparisons at the current level. As `quicksort` is expected to have $O(\log n)$ levels of recursion, and $O(n)$ comparisons at each level can be resolved in time $O(n + \log n \cdot C(n))$, time bound follows.

Cole's improvement cannot be applied in this case, because all comparisons at one level have to be resolved before we even know what comparisons have to be done at the next level (that is, we don't know the splits around pivots until the very last comparison is resolved).

3 Our Practical Version of Cole's Technique

In this section, we describe our algorithm engineering framework for making Cole's parametric search technique practical. Our approach results in a randomized parametric search algorithm with a running time of $O(n \log n + \log n \cdot C(n))$, with high probability, which makes no assumptions about the input. Our framework involves resolving median-weight comparison, according to a potential function based on Cole-style weights assigned to comparisons of a fairly obscure sorting algorithm, which we review next.

3.1 The boxsort Algorithm

We use the `boxsort` algorithm due to Reischuk [26] (see also [25]) as \mathcal{A} . This algorithm is based on an extension of the main idea behind randomized `quicksort`, namely splitting elements around pivots and recursing into subproblems. While `quicksort` randomly selects a single pivot and recurses into two subproblems, `boxsort` randomly selects \sqrt{n} pivots and recurses into $\sqrt{n} + 1$ subproblems in a single stage. We think of it as a parallel algorithm, in the sense that the recursive calls on the same level are independent of each other. The pseudocode is shown in Algorithm 1.

Sorting in lines 3 and 6 is done in a brute-force manner, by comparing all pairs of items, in time $O(n^2)$ in line 3, and $O(n)$ in line 6 (note that since all these comparisons are independent, they can all be realized in a single parallel step).

Once the *marked* items are sorted in line 6, splitting in line 7 is simply $n - \sqrt{n}$ independent binary searches through the *marked* items (to determine, for each unmarked element, the subproblem where it lands). It takes $O(n \log \sqrt{n})$ time (when realized in a sequential way). Equivalently, we think of the sorted set of *marked*

```

// N - original number of items
proc boxsort( $A[i \dots j]$ )
1:  $n \leftarrow (j - i + 1)$ 
2: if  $n < \log N$  then // base case
3:   sort  $A[i \dots j]$ 
4: else
5:   randomly mark  $\sqrt{n}$  items
6:   sort the marked items
7:   use the marked items to split  $A[i \dots j]$  into sub-
   problems  $A_1, A_2, \dots, A_{\sqrt{n}+1}$ 
8:   for all  $i \leftarrow 1 \dots \sqrt{n} + 1$  do
9:     boxsort( $A_i$ )
10:    end for
11:   end if

```

Algorithm 1: boxsort

items as forming a perfectly balanced binary search tree. Locating a destination subproblem for an item is then done by *routing* the item through this tree. The tree has $\log \sqrt{n}$ levels, and all routing comparisons are independent between different unmarked items. Therefore, *routing* can be realized in $\log \sqrt{n}$ parallel steps.

3.2 Weighting Scheme

Motivated by Cole’s approach, we assign weight to every *active* comparison, and resolve the weighted median comparison in a single step. For simplicity, we identify each comparison $x_i < x_j$ with a single comparison against the optimum value, i.e., $\lambda_{ij} < \lambda^*$ for real λ_{ij} (in essence, we assume that comparison polynomials have degree 1). It is straightforward to extend the scheme for the case of higher degrees of comparison polynomials.

It makes sense here to think of **boxsort** in a network-like fashion, in order to understand how the weights are assigned to comparisons. Here, nodes represent comparisons, and directed edges represent dependence on previous comparisons. Furthermore, we imagine the network with edges directed downward, and refer to edge sources as *parents*, and destinations as *children*. Comparison becomes *active* as soon as all its dependencies become resolved (and stops when it gets resolved).

Our “network” also contains nodes for *virtual comparisons*. These are not real comparisons, and don’t appear during actual execution of the algorithm. Their sole purpose is to make it easy to assign weights to *real* comparisons once they become *active* (we will later see that, in fact, they are not necessary even for that; but they make it easy to understand how the weights are computed). When a *virtual* comparison becomes *active*, it is automatically resolved (reflecting the fact that there is no *real* work assigned to a virtual comparison).

Contrary to Cole’s weighting scheme for sorting networks, our scheme does not rely only on comparison’s depth when assigning weights. In fact, different compar-

isons at the same level of the network may have different weights. Weights are assigned to comparisons (virtual or not) according to the following *weight rule*:

When comparison C of weight w gets resolved and causes m comparisons C_1, \dots, C_m to become *active*, each of these comparisons gets weight $w/2m$.

Informally, resolved comparison distributes half of its weight among its newly activated children. Each comparison gets its weight only once, from its last resolved parent (the scheme guarantees that all parents of a comparison have equal weight).

3.3 The Algorithm

Simulating a single recursive call of **boxsort** (including the *virtual* parts) consists of the following steps.

1. Randomly mark \sqrt{n} items.
2. Create $\sqrt{n} \cdot (\sqrt{n} - 1)/2 = O(n)$ comparisons for sorting *marked* items.
3. Construct a complete binary tree of virtual comparisons (comparisons from Step 2 are leaves).
4. Create *routing* trees from section 3.1 for routing unmarked elements; make the root of each such tree depend on the root of the tree from Step 3.
5. Route items through the tree of *marked* items;
6. Construct a binary tree of virtual comparisons (leaves are last comparisons from *routing* trees).
7. Split items into boxes
8. Assign weights for comparisons in the next level of recursion (after the items are split into boxes) by making them children of the root from Step 6.
9. Recurse into subproblems (simultaneously).

Blue steps (3, 6) deal with trees of virtual comparisons, while *red* steps (4, 8) represent relationships that make *real* comparisons depend on *virtual* ones. The idea behind *blue* steps is to ensure synchronization (that is, guarantee that all *real* comparisons on the levels above have been resolved), and *red* steps are there to ensure proper assignment of weights. For simplicity, we present heights/weights as if there were exactly n (instead of $\sqrt{n} \cdot (\sqrt{n} - 1)/2$) comparisons between *marked* items, and exactly n (instead of $n - \sqrt{n}$) unmarked items to be routed. This assumption also applies to the following.

Steps 1 and 7 do not involve any comparisons, and they do not affect weights. Comparisons from Step 2 start with weight w . The tree from Step 3 has height

$\log n$, so its root, according to the *weight rule* gets weight $w/(2^{\log n}) = w/n$. Dependencies introduced in **Step 4** between that root and roots of the *routing trees* cause their weight to be $w/2n^2$ (weight w/n divided among n comparisons). *Routing trees* have height $\log \sqrt{n}$, so the comparisons at their bottom have weight $w/2n^{2.5}$ ($w/2n^2$ divided by $2^{\log \sqrt{n}}$, because, as the routing progresses, the *routing trees* get whittled down to paths, and resolving a routing comparison *activates* at most one new routing comparison. **Step 6** is essentially the same as **Step 3**, so the root of the second *virtual tree* gets weight $w/2n^{3.5}$. All initial comparisons in the subsequent recursive calls (sorting of new *marked* items and/or sorting in the base case) depend on this root (**Step 8**), and they are given weight $w/4n^{4.5}$ (much like in **Step 4**). The height of the dependence network is $O(\log n)$, and at any given moment the number of currently *active* comparisons does not exceed n .

From now on, comparisons are independent across different subproblems. For subsequent subproblems, n from the above discussion gets substituted by \hat{n} , the size of the subproblem. Since subproblem sizes may differ, comparisons on the same level of the network (general level, for the entire algorithm) are no longer guaranteed to have same weights (weights of comparisons belonging to the same subproblem are however equal).

The above discussion shows that, as advertised, we don't really need *virtual* comparisons in order to assign weights to *real* comparisons, as these depend only on n , the size of the subproblem. Therefore, the actual algorithm only consists of steps 1, 2, 5, 7, and 9 and is the following.

1. Randomly mark \sqrt{n} items
2. Sort *marked* items by comparing every pair in $O(n)$ comparisons, each of weight w .
3. When the last comparison finishes, *activate* comparisons for routing through the tree of *marked* items, each of weight $w/2n^2$.
4. Route items through the trees, following the *weight rule* when a comparison gets resolved.
5. When the destination for the last item is determined, split items into boxes (no additional comparisons resolved here).
6. Assign weight $w/4n^{4.5}$ to initial comparisons in new subproblems.
7. Recurse into subproblems (simultaneously).

3.4 Analysis

Assume that initially all comparisons at the highest level were given weight 1. Here, we also include *virtual* com-

parisons. Motivated by Cole's analysis [11], we get the following (for details, refer to the full version [19]).

Lemma 1 $O(f(n) + \log n)$ rounds of resolving the median-weight comparison suffice to resolve every comparison, where $f(n)$ is the height of **boxsort**'s network.

We also have the following fact about **boxsort**.

Lemma 2 (Theorem 12.2 of [25]) There is a constant $b > 0$ such that **boxsort** terminates in $O(\log n)$ parallel steps with probability at least $1 - \exp(-\log^b n)$.

Originally, **boxsort** requires $O(\log n)$ parallel steps to execute a single recursive call for a problem of size n . We noted that the dependence network for a single recursive call in our simulation has height $O(\log n)$ for a problem of size n as well. This means that Lemma 2 applies here and proves that, with high probability, the dependence network for the entire simulation has height $O(\log n)$.

Combining that with Lemma 1 and the observation that any level in the dependence network contains $O(n)$ comparisons, we get the following.

Theorem 3 With high probability, the presented algorithm requires $O(\log n)$ calls to \mathcal{C} , yielding an $O(n \log n + \log n \cdot C(n))$ time parametric search solution to problem B.

4 Conclusion

We have introduced a practical version of Cole's optimization of the parametric search technique. Our method results in a randomized algorithm whose running time matches that of using Cole's technique, with high probability, while being easily implementable. We have implemented it and, based on experimentation performed on some geometric problems (details in the full paper [19]), showed that our approach is competitive with the previous practical parametric search technique of van Oostrum and Veltkamp [31], while having superior asymptotic performance guarantees.

References

- [1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.
- [2] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998.
- [3] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17(3):292–318, 1994.

- [4] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, January 1983.
- [5] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.
- [6] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.
- [7] H. Brönnimann and B. Chazelle. Optimal slope selection via cuttings. *Computational Geometry: Theory and Applications*, 10(1):23–29, 1998.
- [8] E. W. Chambers, E. Colin de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Walking your dog in the woods in polynomial time. In *24th ACM Symp. on Computational Geometry*, pages 101–109, 2008.
- [9] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999.
- [10] T. M. Chan. An optimal randomized algorithm for maximum tukey depth. In J. I. Munro, editor, *SODA*, pages 430–436. SIAM, 2004.
- [11] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.
- [12] R. Cole. Parallel merge sort. *SIAM J. Comput.*, 17:770–785, August 1988.
- [13] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, 1989.
- [14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- [15] C. A. Duncan, M. T. Goodrich, and E. A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *8th ACM-SIAM Symp. on Discrete algorithms (SODA)*, pages 121–130, 1997.
- [16] H. Fournier and A. Vigneron. A deterministic algorithm for fitting a step function to a weighted point-set. *CoRR (arXiv ePrint)*, abs/1109.1152, 2011.
- [17] M. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry*, 14:445–462, 1995.
- [18] M. T. Goodrich and S. R. Kosaraju. Sorting on a parallel pointer machine with applications to set expression evaluation. *J. ACM*, 43:331–361, March 1996.
- [19] M. T. Goodrich and P. Pszona. Cole’s parametric search technique made practical. *CoRR (arXiv ePrint)*, abs/1306.3000, 2013.
- [20] C. A. R. Hoare. Algorithm 64: Quicksort. *Commun. ACM*, 4:321–, July 1961.
- [21] M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Information Processing Letters*, 47(3):115–122, 1993.
- [22] A. Koike, S.-I. Nakano, T. Nishizeki, T. Tokuyama, and S. Watanabe. Labeling points with rectangles of various shapes. *International Journal of Computational Geometry and Applications*, 12(6):511–528, 2002.
- [23] J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters*, 39(4):183–187, 1991.
- [24] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.
- [25] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.
- [26] R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM J. Comput.*, 14(2):396–409, 1985.
- [27] A. Reiser. A linear selection algorithm for sets of elements with weights. *Inf. Process. Lett.*, 7(3):159–162, 1978.
- [28] J. S. Salowe. Parametric search. In J. E. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 969–982. Chapman & Hall/CRC Press, Inc., 2004.
- [29] J. Schwerdt, M. H. M. Smid, and S. Schirra. Computing the minimum diameter for moving points: An exact implementation using parametric search. In *ACM Symp. on Computational Geometry*, pages 466–468, 1997.
- [30] S. Toledo. *Extremal Polygon Containment Problems and Other Issues in Parametric Searching*. MS Thesis, Dept. Comput. Sci., Tel Aviv Univ., Tel Aviv, 1991.
- [31] R. van Oostrum and R. C. Veltkamp. Parametric search made practical. *Computational Geometry: Theory and Applications*, 28(2-3):75–88, 2004.

Polynomial Time Algorithms for Label Size Maximization on Rotating Maps

Yusuke Yokosuka*

Keiko Imai†

Abstract

Map labeling is a problem of placing labels at corresponding graphical features on a map. There are two optimization problems: the label number maximization problem and the label size maximization problem. In general, both problems are NP-hard for static maps. Recently, the widespread use of several applications, such as personal mapping systems, has increased the importance of dynamic maps and the label number maximization problem for dynamic cases has been studied. In this paper, we consider the label size maximization problem for points on rotating maps. Our model is as follows. For each label, a point is chosen inside the label or on its boundary as an anchor point. Each label is placed such that the anchor point coincides with the corresponding point on the map. Furthermore, while the map fully rotates from 0 to 2π , the labels are placed horizontally according to the angle of the map. Our problem consists of finding the maximum scale factor for the labels such that the labels do not intersect, and deciding the place of the anchor points. We propose an $O(n \log n)$ -time and $O(n)$ -space algorithm for the case where each anchor point is inside the label. Moreover, if the labels are of unit-height (or unit-width) and the anchor points are on the boundary, we also present an $O(n \log n)$ -time and $O(n)$ -space algorithm.

1 Introduction

Map labeling is the problem of placing text or symbol labels corresponding to graphical features on input maps such that the labels are pairwise disjoint. This problem is important in several areas, such as geographic information system (GIS), cartography, and graph drawing. On maps, labels of regions, rivers, stations, etc., are placed in appropriate positions so that the corresponding features in the map can be understood. In map labeling, points, polylines, and polygons are considered as graphical features. In this paper, we consider map labeling for points.

A lot of map labeling research has been presented [15]. There are two optimization problems in map labeling.

*Department of Information and System Engineering, Graduate School of Science and Engineering, Chuo University, yoyu@imai-lab2.ise.chuo-u.ac.jp

†Department of Information and System Engineering, Chuo University, imai@ise.chuo-u.ac.jp

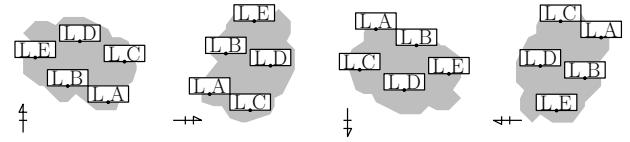


Figure 1: Example of label size maximization problem for rotating maps.

One is the *label number maximization problem* of finding the placement of a maximum cardinality subset of labels with fixed size. The other is the *label size maximization problem* of placing all labels such that the sizes of the labels are maximized under a global scale factor. Most research has considered static maps.

Recently, the importance of dynamic maps has increased due to several applications such as personal mapping systems. There are a lot of dynamic cases, for example, panning, rotating, and zooming maps, translating points, moving points with different velocity. In this context, research on map labeling for dynamic cases has been presented [2, 3, 9, 10]. Mainly, the dynamic label number maximization problem was investigated in their research. In contrast to this, it is a natural direction to consider label size maximization problems for dynamic maps.

In this paper, we consider rotating maps. Since commercial GIS applications (e.g., navigation) often rotate maps dynamically according to the direction in which the user is facing, we assume that labels are placed horizontally according to the angle of the map. We consider the problem of maximizing the label size such that the labels are pairwise disjoint over all rotations $\theta \in [0, 2\pi)$ (Figure 1).

1.1 Problem Definition and Our Results

Let M be a map that includes a set of points $P = \{p_1, \dots, p_n\}$ in the plane with a set of labels $L = \{\ell_1, \dots, \ell_n\}$. In this paper, the labels are considered to be open axis-aligned rectangles of different sizes. Each initial size of $\ell_i \in L$ is expressed by its width $w_i > 0$ and height $h_i > 0$. When the *scale factor* is σ , the label size of ℓ_i is $w_i\sigma \times h_i\sigma$.

Each label is placed such that a point called an *anchor point* coincides with the corresponding point p_i (Figure 2 (a)). The anchor point is inside the label ℓ_i or on its boundary. When the label ℓ_i is fixed in place,

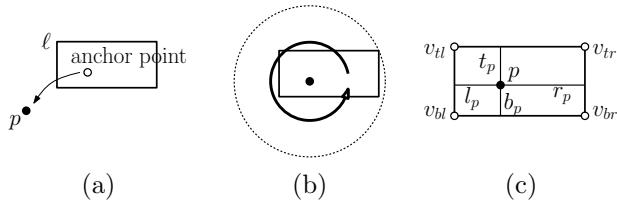


Figure 2: Definitions.

Table 1: Our results (running time). All solvable problems can be computed in $O(n)$ space.

Rectangle shape	MSR	MSBR
Unit squares	$O(n \log n)$ (Theorem 4)	$O(n \log n)$ (Corollary 5)
Squares		-
Unit-height rectangles	$O(n \log n)$ (Theorem 6)	$O(n \log n)$ (Corollary 9)
Rectangles		-

we say that it is *anchored* at p_i . While M fully rotates from 0 to 2π with the anchor points touching the corresponding points, the labels are placed horizontally, and should not intersect each other. Our problem is finding the *maximum scale factor* σ^* such that the labels do not intersect, and deciding the place of the anchor points. In ordinary map labeling, each label is placed such that the anchor point is on the boundary of its label. However, we also consider the case that the point is inside the label. We call the former problem the *maximization problem of the size of labels with boundary anchor points on rotating maps (MSBR)*, and the latter problem the *maximization problem of the size of labels on rotating maps (MSR)*. This formulation on dynamic maps is a natural extension of the label size maximization problem on static maps.

Our results are summarized in Table 1. We address several rectangular label shapes (e.g., unit squares and unit-height rectangles). Although static label size maximization is NP-hard [8], MSR and MSBR can be solved in polynomial time, which is surprising.

In the following, we treat the clockwise rotation of M as the counterclockwise rotation of labels around their anchor points (Figure 2 (b)), as did Gemsa et al. [9]. Both rotations are equivalent and yield exactly the same results.

1.2 Related Work

In map labeling, two models have been considered w.r.t. the number of label candidates for each point: the *fixed-position model* [8] and the *slider model* [14]. In both models, each label is placed such that the corresponding point is on the boundary of the label. The fixed-position model has a finite number of label candidates (e.g., the

2-position and 4-position model). The label candidates of the slider model are the specified sides of the labels (e.g., in the 2-slider model, two sides of the label serve as a set of label candidates).

It is known that the static label size maximization problems, except for the 1-position and 2-position model, are APX-hard, even for unit square labels [8]. A lot of constant-factor approximation algorithms have been proposed for several axis-parallel rectangles [8, 11]. Doddi et al. [6] dealt with unit square labels with different orientations, and Zhu and Qin [16] considered the case that all the square labels have the same orientation. Furthermore, the static label number maximization problems in several models are known to be NP-hard (e.g., [8, 14]). Therefore, many approximation algorithms have already been presented (e.g., [1, 14]).

In dynamic map labeling, Been et al. [2] proposed consistency desiderata for dynamic map labeling, which are that labels should not pop and jump during panning and zooming. Been et al. [3] treated the problems of maximizing the lengths of active ranges, where the active range of a label ℓ is a contiguous range of map scales at which ℓ is displayed. Moreover, the problem satisfies that the labels are pairwise disjoint at any scale and satisfy the consistency desiderata. They proved that the problems for points in the plane are NP-hard, and proposed several exact and approximation algorithms for points in 1D and 2D. Gemsa et al. [10] extended the above problems to the slider model, and also dealt with selecting the slider positions. Moreover, Gemsa et al. [9] considered similar dynamic map labeling for rotating maps. They also proved that the problem is NP-hard, and proposed approximation algorithms.

In the circular labeling problem [13], the corresponding point in the plane is on the boundary of the circular label. However, in MSR and MSBR, during the rotation, the point is inside the label or on the boundary, and it may not be on the circle obtained by rotation of the label.

2 Properties

In this section, first, we investigate locations of anchor points such that the scale factor is maximized. Next, for the locations, we calculate the maximum scale factor.

Let ℓ_p be a label anchored at a point p with the initial width w_p and the initial height h_p . Further, the top-left, top-right, bottom-left, and bottom-right point of ℓ rotated by angle 0 are denoted by v_{tl} , v_{tr} , v_{bl} , and v_{br} , respectively. Draw the segments passing through p in parallel with the edges of ℓ_p . We assume that p divides the horizontal segment and vertical segment internally in the ratio $l_p : r_p$ (where $r_p = 1 - l_p$) and $t_p : b_p$ (where $b_p = 1 - t_p$), respectively (Figure 2 (c)). We define each parameter for a point p' in the same way. Thus, $\ell_{p'}$ is

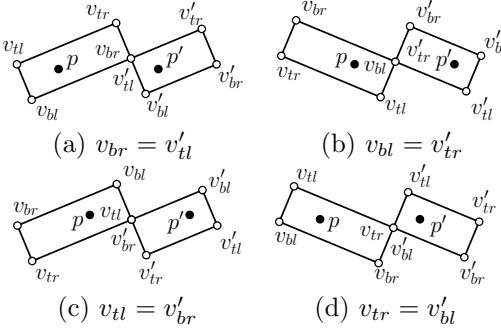


Figure 3: Four possible cases that the corner points of two labels ℓ_p and $\ell_{p'}$ intersect.

the label of p' . $w_{p'}$ and $h_{p'}$ are the initial width and initial height of $\ell_{p'}$, respectively. Moreover, the above parameters of $\ell_{p'}$ are defined as v'_{tl} , v'_{tr} , v'_{bl} , v'_{br} , $l_{p'}$, $r_{p'}$, $t_{p'}$ and $b_{p'}$. Finally, let d be the distance between p and p' , and $\sigma_{pp'}$ be the maximum scale factor for p and p' .

Lemma 1 *Let ℓ_p and $\ell_{p'}$ be two labels, which are anchored at points p and p' , respectively. ℓ_p and $\ell_{p'}$ can be placed with the maximum scale factor $\sigma_{pp'}$ if and only if the anchor points of ℓ_p and $\ell_{p'}$ satisfy $(1 - 2l_p)w_p = (1 - 2l_{p'})w_{p'}$ and $(1 - 2t_p)h_p = (1 - 2t_{p'})h_{p'}$.*

Proof. Without loss of generality, we assume that p and p' lie on a horizontal line. Let σ be the scale factor for p and p' . Note that ℓ_p and $\ell_{p'}$ touch at their corner points. Otherwise, if ℓ_p and $\ell_{p'}$ touch on their boundary segments, they overlap by slight rotation. Moreover, ℓ_p and $\ell_{p'}$ are parallel. Therefore, there are only the following four possible cases: $v_{br} = v'_{tl}$, $v_{bl} = v'_{tr}$, $v_{tl} = v'_{br}$, and $v_{tr} = v'_{bl}$ (Figure 3).

We consider the case that $v_{tl} = v'_{br}$ (Figure 4). Since ℓ_p and $\ell_{p'}$ are parallel, we have $(l_p w_p \sigma + (1 - l_{p'}) w_{p'} \sigma)^2 + (t_p h_p \sigma + (1 - t_{p'}) h_{p'} \sigma)^2 \leq d^2$. Therefore,

$$\sigma \leq d / \sqrt{(l_p w_p + (1 - l_{p'}) w_{p'})^2 + (t_p h_p + (1 - t_{p'}) h_{p'})^2}. \quad (1)$$

In the same way, if $v_{tr} = v'_{bl}$,

$$\sigma \leq d / \sqrt{((1 - l_p) w_p + l_{p'} w_{p'})^2 + (t_p h_p + (1 - t_{p'}) h_{p'})^2}, \quad (2)$$

if $v_{br} = v'_{tl}$,

$$\sigma \leq d / \sqrt{((1 - l_p) w_p + l_{p'} w_{p'})^2 + ((1 - t_p) h_p + t_{p'} h_{p'})^2}, \quad (3)$$

and if $v_{bl} = v'_{tr}$,

$$\sigma \leq d / \sqrt{(l_p w_p + (1 - l_{p'}) w_{p'})^2 + ((1 - t_p) h_p + t_{p'} h_{p'})^2}. \quad (4)$$

First, we focus on the inequalities (1) and (2) (or, (3) and (4)). As the denominators of the right-hand sides

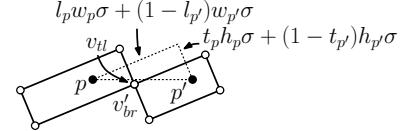


Figure 4: The case that $v_{tl} = v'_{br}$.

of (1) and (2) become smaller, the maximum possible σ becomes greater. $(t_p h_p + (1 - t_{p'}) h_{p'})^2$ is appeared in both right-hand sides of (1) and (2). The smaller $(l_p w_p + (1 - l_{p'}) w_{p'})^2$ becomes, the greater $((1 - l_p) w_p + l_{p'} w_{p'})^2$ becomes. Therefore, if $(l_p w_p + (1 - l_{p'}) w_{p'})^2 = ((1 - l_p) w_p + l_{p'} w_{p'})^2$, σ is maximized among values satisfying (1) and (2). This condition is equivalent to the equation $(1 - 2l_p)w_p = (1 - 2l_{p'})w_{p'}$. Similarly, in case that the inequalities (1) and (4) (or, (2) and (3)), if $(1 - 2t_p)h_p = (1 - 2t_{p'})h_{p'}$, σ is maximized among values satisfying (1) and (4). The above two equations are satisfied simultaneously. Therefore, if $(1 - 2l_p)w_p = (1 - 2l_{p'})w_{p'}$ and $(1 - 2t_p)h_p = (1 - 2t_{p'})h_{p'}$, σ is the maximum scale factor $\sigma_{pp'}$.

The converse is also true. \square

From Lemma 1, we can obtain the following lemma.

Lemma 2 *For given points p and p' with labels ℓ_p and $\ell_{p'}$, if the anchor point of each label is the intersection of two diagonals of the label, ℓ_p and $\ell_{p'}$ can be placed with the maximum scale factor $\sigma_{pp'}$.*

Proof. Let p and p' be two points. In the case that the anchor points lie in the label centers, $l_p = t_p = l_{p'} = t_{p'} = 1/2$. Therefore, $(1 - 2 \times 1/2)w_p = (1 - 2 \times 1/2)w_{p'} = 0$ and $(1 - 2 \times 1/2)h_p = (1 - 2 \times 1/2)h_{p'} = 0$. The conditions in Lemma 1 are satisfied, and hence, the scale factor $\sigma_{pp'}$ is maximized. \square

From Lemma 2, the maximum scale factor $\sigma_{pp'}$ of MSR for p and p' is $2d / \sqrt{(w_p + w_{p'})^2 + (h_p + h_{p'})^2}$. Therefore, we can solve MSR for more than two points by computing the maximum scale factor σ_{ij} for all point pairs p_i and p_j , and choosing the minimum among those. This naive algorithm runs in $\Theta(n^2)$ time. Moreover, if all heights (or widths) of labels are equal to each other, we obtain the following proposition.

Proposition 3 *MSBR for unit-height (or unit-width) rectangular labels can be computed in $\Theta(n^2)$ time.*

Proof. The naive algorithm of MSR gives the maximum scale factor σ^* for the unit-height rectangular labels. We consider points obtained by translating the anchor points placed at the center of rectangles in MSR to the top or bottom (or, left or right) boundary. Those points satisfy the equations (1)–(4) in Lemma 1. Therefore, the points are the anchor points in MSBR and σ^* is also the maximum scale factor in MSBR. \square

In the following sections, we will improve the time complexity of these algorithms for MSR and MSBR to $O(n \log n)$.

3 Square Labels

When the labels of all points are squares, the problem has a strong connection to the *weighted closest pair problem* [7]: The input is a set of disks. Each disk has a point in P as its center, a weight W , and a radius $W\sigma$, where σ is a scale factor. The goal is to find the maximum scale factor σ^* such that the disks are pairwise disjoint.

Theorem 4 *MSR for square labels can be computed in $O(n \log n)$ time and $O(n)$ space.*

Proof. Let p and p' be two points that have square labels, and lie on a horizontal line. Let $\sigma_{pp'}$ be the maximum scale factor for p and p' . Since the labels are square, we have $w_p = h_p$ and $w_{p'} = h_{p'}$. When the labels are anchored at p and p' , and their anchor points are their centers by Lemma 2, the distance between p and p' is $\frac{\sqrt{2}}{2}(w + w')\sigma_{pp'}$. Then, $\sigma_{pp'}$ is determined by the angles $\pi/4$, $3\pi/4$, $5\pi/4$, and $7\pi/4$. We consider the disks drawn by fully rotating the square labels around the points p and p' . The maximum scale factor $\sigma_{pp'}$ is obtained by maximizing the size of the disks such that they are pairwise disjoint.

Therefore, MSR for square labels is considered as the weighted closest pair problem with weight $W = \frac{\sqrt{2}}{2}w_p$ for each point p . For the weighted closest pair problem, Formann [7] proposed an $O(n \log n)$ -time and $O(n)$ -space algorithm based on a plane sweep. Therefore, this completes the proof. \square

Corollary 5 *MSBR for unit square labels can be computed in $O(n \log n)$ time and $O(n)$ space.*

4 Rectangular Labels

For the rectangular labels, the algorithm of square labels does not work directly because the disks obtained by sweeping the rectangular labels around their anchor points can intersect when the scale factor is maximized. However, Formann's idea [7] used in weighted closest pair problem can be modified to MSR and MSBR for rectangular labels. Our modified algorithm overestimates the maximum scale factor, and then fixes the maximum value using the intersection graph of disks drawn by fully rotation of the labels. In the algorithm, we use the *Delaunay triangulation* [5, 12] of P , $DT(P)$, which is a triangulation with the *empty circle property*: for any triangle T in $DT(P)$, the circumcircle of T contains no points of P in its interior. We call a triangle of $DT(P)$ a *Delaunay triangle*. When points p and q are

vertices of a Delaunay triangle in $DT(P)$, q is called a *neighbor* of p .

Our algorithm can be described as Algorithm 1.

Algorithm 1 Algorithm for MSR.

- 1: Compute $DT(P)$ for P .
 - 2: For each point p , calculate the maximum scale factor σ_p with all the neighbors in $DT(P)$. Take the minimum scale factor $\sigma_{\text{pre}} = \min_{p \in P} \sigma_p$ of all the scale factors.
 - 3: For each point $p \in P$, draw a closed disk with center p and radius $\frac{\sigma_{\text{pre}}}{2} \sqrt{w_p^2 + h_p^2}$. Enumerate all intersections of disks using the standard intersection detecting algorithm of Bentley and Ottmann [4].
 - 4: Calculate the maximum scale factor for all intersections of disks, and take the minimum value among them as σ^* .
-

The following theorem shows the correctness of Algorithm 1 and its complexity. In the following, let D_p be the disk centered at p in Step 3 of Algorithm 1, and let R_p be its radius $\frac{\sigma_{\text{pre}}}{2} \sqrt{w_p^2 + h_p^2}$.

Theorem 6 *MSR can be computed in $O(n \log n)$ time and $O(n)$ space.*

In order to prove Theorem 6, we present some lemmas.

Lemma 7 *Each disk obtained after Step 3 of Algorithm 1 contains no points in P other than its center point.*

Proof. For each $p \in P$, let D_p be the disk with center p and radius $R_p = \frac{\sigma_{\text{pre}}}{2} \sqrt{w_p^2 + h_p^2}$. From the definition of σ_{pre} , the labels of p and q do not intersect during rotation for a neighbor q . Therefore, D_p cannot contain neighbors of p . In the Delaunay triangulation, the nearest point q of p is a neighbor of p in $DT(P)$. Therefore, the radius of D_p is less than $|pq|$. From this, D_p cannot contain points that are not the neighbors of p in $DT(P)$. \square

Lemma 8 *The number of intersecting pairs in the set of disks obtained at Step 3 of Algorithm 1 is at most $3n - 6$.*

Proof. First, we draw straight line segments between the points whose closed disks intersect at Step 3 of Algorithm 1. We will show that the straight line graph G having the line segments as edges is planar. We consider the case that two closed disks D_p and $D_{p'}$ intersect. In G , p and p' are connected by a straight line edge. If there is no other disk D_q centered at a point $q \neq p, p'$

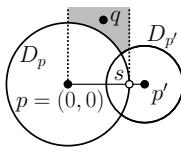


Figure 5: Assumption of Lemma 8.

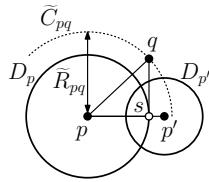


Figure 6: Case 1.

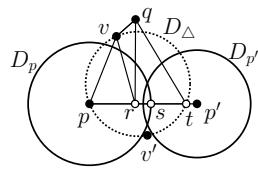


Figure 7: Case 2-1.

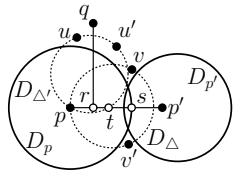


Figure 8: Case 2-2a.

which intersects the line segment $\overline{pp'}$, no two line segments in G intersect without endpoints. This shows that the graph G is planar.

Without loss of generality, we assume that p and p' lie on a horizontal line and the x -coordinate of p' is greater than that of p (Figure 5). We denote the x - and y -coordinates of p and p' by x_p , y_p , $x_{p'}$, and $y_{p'}$, respectively. We denote the x - and y -coordinates of the other points in the same way. Let s be the intersection of the boundary of D_p and $\overline{pp'}$. In the following, we assume $0 = x_p \leq x_q \leq x_s$ and $0 = y_p = y_{p'} \leq y_q$. q is in the shaded area in Figure 5. When $x_q < 0$ or $x_{p'} < x_q$, D_p cannot intersect $\overline{pp'}$, by Lemma 7. Moreover, when $y_q < 0 = y_p = y_{p'}$ or $x_s < x_q \leq x_{p'}$, these cases can be proved in the same way.

We consider separately the cases that q is or is not a neighbor of p in $\text{DT}(P)$.

Case 1: q is a neighbor of p in $\text{DT}(P)$.

We denote $\frac{\sigma_{\text{pre}}}{2}\sqrt{(w_p + w_q)^2 + (h_p + h_q)^2}$ by \tilde{R}_{pq} . In this case, by the definition of σ_{pre} , we have $|\overline{pq}| \geq \tilde{R}_{pq}$. Moreover, since $w_q, h_q > 0$, \tilde{R}_{pq} is greater than R_p . Let \tilde{C}_{pq} be a circle centered at p with radius \tilde{R}_{pq} . \tilde{C}_{pq} is shown as a dotted circle in Figure 6. Note that the vertical distance between q and $\overline{pp'}$ is greater than or equal to the length of a vertical straight segment from s to \tilde{C}_{pq} . Then, we consider the case that $x_q = x_s$. Because $|\overline{pq}| \geq \tilde{R}_{pq}$ and $|\overline{ps}| = R_p = \frac{\sigma_{\text{pre}}}{2}\sqrt{w_p^2 + h_p^2}$, we have that

$$\begin{aligned} |\overline{sq}|^2 &= |\overline{pq}|^2 - |\overline{ps}|^2 \\ &\geq \left(\frac{\sigma_{\text{pre}}}{2}\right)^2 ((w_p + w_q)^2 + (h_p + h_q)^2) \\ &\quad - \left(\frac{\sigma_{\text{pre}}}{2}\right)^2 (w_p^2 + h_p^2) \\ &= \left(\frac{\sigma_{\text{pre}}}{2}\right)^2 (w_q^2 + h_q^2 + 2w_p w_q + 2h_p h_q). \end{aligned}$$

Since $R_q = \frac{\sigma_{\text{pre}}}{2}\sqrt{w_q^2 + h_q^2}$ and $w_p, h_p, w_q, h_q > 0$, we have that $|\overline{sq}|^2 - R_q^2 \geq \left(\frac{\sigma_{\text{pre}}}{2}\right)^2 (2w_p w_q + 2h_p h_q) > 0$. Therefore, D_q cannot intersect $\overline{pp'}$.

Case 2: q is not a neighbor of p in $\text{DT}(P)$.

In this case, we can show that there is a Delaunay triangle with p whose circumcircle contains $\overline{pp'} \cap D_p$ in the following way. If p' is a neighbor of p , the circumcircle

of a Delaunay triangle that has the edge $\overline{pp'}$ completely contains $\overline{pp'} \cap D_p$. If p' is not a neighbor of p , there is a Delaunay triangle $\triangle pvv'$ that has an edge that intersects the inside of $\overline{pp'}$. Since $v, v' \notin D_p$ by Lemma 7, the circumcircle of $\triangle pvv'$ completely contains $\overline{pp'} \cap D_p$. Therefore, we consider such a Delaunay triangle $\triangle pvv'$. In this case, v' might be p' . Let v and v' be points such that $y_v > 0$ and $y_{v'} \leq 0$, respectively. Further, let t be the intersection point of the circumcircle of $\triangle pvv'$ and $\overline{pp'}$. Since the circumcircle contains $\overline{pp'} \cap D_p$ regardless of whether p' is a neighbor of p , we have $x_q \leq x_s < x_t$. In the case that $x_v = x_q$, since $y_v < y_q$ and by Lemma 7, D_q cannot intersect $\overline{pp'}$. Therefore, we consider two cases: $x_v < x_q \leq x_s$ and $x_q < x_v < x_s$. In the following, we denote the closed disk whose boundary is the circumcircle of $\triangle pvv'$ by D_{Δ} .

Case 2-1: $x_v < x_q \leq x_s$.

Let t be the intersection of $\overline{pp'}$ and the boundary of D_{Δ} , and r be the intersection of $\overline{pp'}$ and a perpendicular from q to $\overline{pp'}$ (Figure 7). Since D_{Δ} completely contains $\overline{pp'} \cap D_p$, $x_r \leq x_s < x_t$. Because $\angle prq = \pi/2$ and $x_t - x_s > 0$, $\angle ptq < \pi/2$. Moreover, since $q \notin D_{\Delta}$ by the empty circle property, $\angle pvq + \angle ptq \geq \pi$. Therefore, we have $\angle pvq \geq \pi - \angle ptq > \pi/2$. Because $r \in D_p$ and $v \notin D_p$, $|\overline{pr}| < |\overline{pv}|$ and $\angle pvr \leq \angle prv$. Then, we have $\angle qvr = \angle pvq - \angle pvr > \pi/2 - \angle prv = \angle qrv$. Therefore, we have $|\overline{qv}| < |\overline{qr}|$. Since $R_q < |\overline{qv}|$ by Lemma 7, D_q cannot contain r . Therefore, D_q cannot intersect $\overline{pp'}$.

Case 2-2: $x_q < x_v < x_s$.

First, we show that there is a Delaunay triangle $\triangle puv'$ such that $x_u \leq x_q \leq x_{u'}$. Since q is not a neighbor of p in $\text{DT}(P)$, there is a Delaunay triangle $\triangle pzz'$ that has an edge $\overline{zz'}$ that intersects \overline{pq} at an interior point of \overline{pq} . Moreover, v is a vertex of a Delaunay triangle that has p as its vertex. Since p is in the polygon consisting of Delaunay triangles that have p as their vertex, when we visit the Delaunay triangles clockwise from $\triangle pzz'$ to $\triangle pvv'$, there is a Delaunay triangle $\triangle puv'$ that satisfies the condition. In the following, we denote the closed disk whose boundary is the circumcircle of $\triangle puv'$ by $D_{\Delta'}$, and the boundary of $D_{\Delta'}$ by $C_{\Delta'}$. We consider the cases that $r \in D_{\Delta'} \setminus C_{\Delta'}$ and $r \notin D_{\Delta'} \setminus C_{\Delta'}$.

Case 2-2a: $r \in D_{\Delta'} \setminus C_{\Delta'}$.

Let t be the intersection of $C_{\Delta'}$ and the inside of $\overline{pp'}$ (Figure 8). Since $r \in D_{\Delta'} \setminus C_{\Delta'}$, we have $x_r < x_t$. Therefore, we can use the same proof as for Case 2-1 by

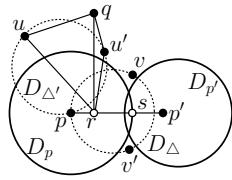


Figure 9: Case 2-2b.

replacing u with v . It is shown that D_q cannot intersect $\overline{pp'}$.

Case 2-2b: $r \notin D_{\Delta'} \setminus C_{\Delta'}$.

In this case, we consider the quadrilateral $uqu'r$ (Figure 9). By Lemma 7, u is not contained inside D_q . Therefore, if D_q intersects $\overline{pp'}$, $|qr| \leq R_q < |qu|$ or $|qr| \leq R_q < |qu'|$. First, we consider the case $|qr| \leq R_q < |qu|$. Since $\angle qur < \angle qru$ in $\triangle qur$, we have $\angle qur < \pi/2$. Since q and r are outside $D_{\Delta'}$, we have $\angle qu'r + \angle qur \geq \pi$. Therefore, we have $\angle qu'r > \pi/2$ and $\angle qru' < \pi/2$. This means that $|qu'| < |qr| \leq R_q$ and u' is inside D_q . This contradicts Lemma 7. The case $|qr| \leq R_q < |qu'|$ can be proven in the same way. Therefore, D_q cannot intersect $\overline{pp'}$. \square

Proof of Theorem 6. First, we show the correctness. From the definition of σ_{pre} in Step 2 of Algorithm 1, two labels whose corresponding points are neighbors in $\text{DT}(P)$ do not intersect. In Step 4, the disks can be drawn by fully rotating the labels from 0 to 2π . Each label has the anchor point at its center, and is scaled by σ_{pre} . Moreover, since $\sigma_{\text{pre}} \geq \sigma^*$, we can obtain σ^* by checking intersecting disks.

Next, we show the complexity. Step 1 can be computed in $O(n \log n)$ time and $O(n)$ space [5, 12]. Step 2 calculates the maximum scale factor between neighbors. Since the number of edges in Delaunay triangulation is $O(n)$, Step 2 can be computed in $O(n)$ time and $O(1)$ space. In Step 3, the algorithm of Bentley and Ottmann [4] can be computed in $O((n+K) \log n)$ time and $O(n+K)$ space where K is the number of intersecting pairs. Moreover, Step 4 can be computed in $O(K)$ time and $O(1)$ space. Since $K \leq 3n - 6$ by Lemma 8, this completes the proof. \square

Corollary 9 *MSBR for unit-height (or unit-width) rectangular labels can be computed in $O(n \log n)$ time and $O(n)$ space.*

From Theorem 4, MSR and MSBR are generalizations of the closest pair problem. The time complexity of this problem is lower-bounded by $\Omega(n \log n)$ [12], which may also apply to our problems.

5 Conclusion

We considered the label size maximization problem for rotating maps. In general, label size maximization

problems for static maps are APX-hard. However, we showed that the problem for rotating maps can be solved in polynomial time, and we presented efficient algorithms for finding the maximum scale factor.

Acknowledgments.

The work of the second author was supported in part by Grant-in-Aid for Scientific Research of Japan Society for the Promotion of Science.

References

- [1] P. K. Agarwal, M. J. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998.
- [2] K. Been, E. Daiches, and C.-K. Yap. Dynamic map labeling. *IEEE Trans. Vis. Comput. Graph.*, 12(5):773–780, 2006.
- [3] K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom.*, 43(3):312–328, 2010.
- [4] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computers*, 28(9):643–647, 1979.
- [5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.
- [6] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu. Map labeling and its generalizations. In *SODA*, pages 148–157, 1997.
- [7] M. Formann. Weighted closest pairs. In *STACS*, pages 270–281, 1993.
- [8] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *ACM Symposium on Computational Geometry*, pages 281–288, 1991.
- [9] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In *WADS*, pages 451–462, 2011.
- [10] A. Gemsa, M. Nöllenburg, and I. Rutter. Sliding labels for dynamic point labeling. In *CCCG*, pages 205–210, 2011.
- [11] J.-W. Jung and K.-Y. Chwa. Labeling points with given rectangles. *Inf. Process. Lett.*, 89(3):115–121, 2004.
- [12] F. P. Preparata and M. I. Shamos. *Computational Geometry - An Introduction*. Springer, 1985.
- [13] T. Strijk and A. Wolff. Labeling points with circles. *Int. J. Comput. Geometry Appl.*, 11(2):181–195, 2001.
- [14] M. J. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Comput. Geom.*, 13(1):21–47, 1999.
- [15] A. Wolff and T. Strijk. The map-labeling bibliography, 2009.
- [16] B. Zhu and Z. Qin. New approximation algorithms for map labeling with sliding labels. *J. Comb. Optim.*, 6(1):99–110, 2002.

Drawing some 4-regular planar graphs with integer edge lengths

Timothy Sun*

Abstract

A classic result of Fáry states that every planar graph can be drawn in the plane without crossings using only straight line segments. Harborth *et al.* conjecture that every planar graph has such a drawing where every edge length is integral. Biedl proves that every planar graph of maximum degree 4 that is not 4-regular has such a straight-line embedding, but the techniques are insufficient for 4-regular graphs. We further develop the rigidity-theoretic methods of the author and examine an incomplete construction of Kemnitz and Harborth to exhibit integral drawings of families of 4-regular graphs.

1 Introduction

All graphs in this paper are simple and finite. Let $G = (V, E)$ be a planar graph. A *Fáry embedding* $\phi : V \rightarrow \mathbb{R}^2$ of a planar graph is an embedding such that the straight-line drawing induced by ϕ has no crossing edges. Fáry [3] proved that all planar graphs have such an embedding. A natural extension of Fáry's theorem is to require that every edge has integral length, but it is not known if every planar graph has such an embedding, which we call an *integral Fáry embedding*.

Conjecture 1 (Harborth *et al.* [7]) All planar graphs have an integral Fáry embedding.

Analogously, we call a Fáry embedding with rational edge lengths a *rational Fáry embedding*. For the remainder of the paper, we consider only rational Fáry embeddings, since an appropriate scaling yields an integral one.

Kemnitz and Harborth [8] show that every planar 3-tree has a rational Fáry embedding. However, their solution for the analogous operation for 4-valent vertices does not always work. Geelen *et al.* [4] use a technical theorem of Berry [1] to prove Conjecture 1 for cubic planar graphs. Biedl [2] notes that their proof extends to even more graphs. One family of interest are the *almost 4-regular graphs*, namely the connected graphs of maximum degree 4 that are not 4-regular. These results actually yield rational Fáry embeddings with rational coordinates, and we call such embeddings *fully-rational*.

Biedl strongly conjectures that all 4-regular graphs have rational Fáry embeddings. The aforementioned methods all rely on inductively adding vertices of degree at most 3 into a rational Fáry embedding, but unfortunately, there are no known general methods for adding vertices of degree 4. It is even unknown whether or not there is a point in the interior of a unit square at rational length from each of the four vertices [6].

A previous paper by the author [11] details a construction of rational Fáry embeddings of graphs using elementary results from rigidity theory. We use these rigidity-theoretic techniques for drawing planar graphs with small edge cuts and synthesize the aforementioned results to prove the existence of rational Fáry embeddings for two families of 4-regular planar graphs, namely those that are not 4-edge-connected and those with a diamond subgraph.

2 Berry's Theorem and 3-Eliminable Graphs

Perhaps the most general technique known for constructing rational Fáry embeddings is the following result of Berry [1].

Theorem 1 (Berry [1]) *Let A , B , and C be points in the plane such that AB , $(BC)^2$, and $(AC)^2$ are rational. Then the set of points P at rational distance with all three points is dense in the plane.*

Geelen *et al.* [4] show that this leads to an inductive method for finding rational Fáry embeddings of a certain family of graphs. If G is a graph on n vertices, a sequence of those vertices v_1, v_2, \dots, v_n is a *3-elimination order* [2] if

1. G is the graph on one vertex, or
2. v_n has degree at most 2 and v_1, \dots, v_{n-1} is a 3-elimination order for $G - v_n$, or
3. v_n has degree 3 and v_1, \dots, v_{n-1} is a 3-elimination order for some graph $(G - v_n) \cup uw$, where u and w are two of the neighbors of v_n .

A graph is said to be *3-eliminable* if it has a 3-elimination order. For any two maps $p, p' : V \rightarrow \mathbb{R}^2$, let $d(p, p')$ be the Euclidean distance between p and p' , interpreted as points in $\mathbb{R}^{2|V|}$. We say that a Fáry embedding ϕ can be *approximated* by a type of Fáry embedding (e.g. rational Fáry embedding) if for all $\epsilon > 0$,

*Department of Computer Science, Columbia University,
ts2578@columbia.edu

there exists a Fary embedding ϕ' of that type such that $d(\phi, \phi') < \epsilon$. Geelen *et al.* essentially prove the following:

Theorem 2 (Geelen *et al.* [4], Biedl [2]) *Any Fary embedding ϕ of a 3-eliminable graph can be approximated by a fully-rational Fary embedding.*

We do not have a non-inductive characterization of 3-eliminable graphs, but some partial results are known. A graph $G = (V, E)$ is called (k, l) -sparse if for every subset of vertices V' of size at least k , the induced subgraph of G on V' has at most $k|V'| - l$ edges.

Theorem 3 (Biedl [2]) *Every $(2, 1)$ -sparse graph is 3-eliminable, and hence any Fary embedding of a $(2, 1)$ -sparse planar graph can be approximated by a fully-rational Fary embedding.*

The family of $(2, 1)$ -sparse graphs contains many other interesting classes of graphs, some of which can be found in [2]. Our interest lies mostly in the following corollary, as it is used in our constructions of rational Fary embeddings for both families of 4-regular planar graphs.

Corollary 4 (Biedl [2]) *Any Fary embedding of an almost 4-regular planar graph can be approximated by a fully-rational Fary embedding.*

3 Rigidity Theory and Graphs With Small Edge Cuts

A *framework* is a pair (G, p) where G is equipped with a *configuration* $p : V(G) \rightarrow \mathbb{R}^d$ which sends vertices to points in d -dimensional Euclidean space. A *generic configuration* is one where its $|V|d$ coordinates are independent over the rational numbers, and a *generic framework* is one with a generic configuration. A framework is *flexible* if there is a continuous motion of the vertices preserving edge lengths that does not extend to a Euclidean motion of \mathbb{R}^d , and it is said to be *rigid* otherwise.

The rigidity of a framework can be tested by examining its rigidity matrix. Let G be a graph on n vertices and m edges, and fix an ordering of the edges e_1, \dots, e_m . Define $f_G : \mathbb{R}^{nd} \rightarrow \mathbb{R}^m$ to be the function that takes a configuration p to a vector $(\|p(e_1)\|^2, \dots, \|p(e_m)\|^2)$ consisting of the squares of the edge lengths. The *rigidity matrix* of (G, p) is defined to be $\frac{1}{2}df_G(p)$, where d is the Jacobian, and its dimensions are $m \times nd$. Then, the kernel of the rigidity matrix corresponds to so-called “infinitesimal motions” of the framework. A *regular point* is a configuration that maximizes the rank of the rigidity matrix over all possible configurations. It is easy to see that generic configurations are all regular points. We say that a Fary embedding is *regular* if it is a regular point.

An edge is *independent* if the corresponding row in the rigidity matrix is linearly independent from the other rows. Otherwise, it is said to be *redundant*, since deleting it does not change the space of infinitesimal motions. For $d > 2$, it is a long-standing open problem to find a combinatorial characterization of graphs with all independent edges. However, a complete characterization is known in two dimensions.

Theorem 5 (e.g. Graver *et al.* [5]) *A generic framework of a graph in \mathbb{R}^2 has all independent edges if and only if it is $(2, 3)$ -sparse.*

A framework is *minimally rigid* if it is rigid and deleting any edge makes it flexible. Perhaps the most well-known restatement of this result is known as Laman’s theorem.

Corollary 6 (Laman [9]) *A generic framework of a graph $G = (V, E)$ is minimally rigid in \mathbb{R}^2 if and only if it is $(2, 3)$ -sparse and has $2|V| - 3$ edges.*

One consequence of this result is that all planar $(2, 3)$ -sparse graphs have rational Fary embeddings, as proved by the author in [11], but we are more concerned with how rigidity theory allows us to draw graphs with small edge cuts. An *edge cut* of a connected graph $G = (V, E)$ is a subset of E whose deletion disconnects the graph. A *minimal edge cut* has no edge cuts as proper subsets. For example, consider a Fary embedding of a planar graph with a bridge uv that splits the graph into subgraphs G_1 and G_2 . Deleting uv yields a new flex, namely the one that allows us to translate G_1 or G_2 in a direction parallel to uv , and we move along this flex until the distance between uv is rational and replace the edge. Such a technique can be generalized to cuts of up to three edges, using the main trick from [11].

Lemma 7 (Sun [11]) *Let ϕ be a regular Fary embedding of G , and let uv be an independent edge. Then, ϕ can be approximated by a regular Fary embedding ϕ' such that $\|\phi'(u) - \phi'(v)\|$ is rational, and all other edge lengths remain the same.*

Theorem 8 *Let $G = (V, E)$ be a graph with a minimal edge cut $\{e_1, e_2, e_3\}$ which separates G into G_1 and G_2 . Furthermore, suppose that e_1, e_2 , and e_3 are not all incident with the same vertex. Then, each e_i is independent in a generic framework.*

Proof. Assume without loss of generality that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are minimally rigid graphs. If G is also minimally rigid, then each of the edges in the cut must be independent. Furthermore, assume that V_1 and V_2 are just the vertices incident with the e_i ’s, in which case G_1 and G_2 are one of the complete graphs K_2 or K_3 . We can make this assumption because any flex

on G induces a rigid motion on G_1 and G_2 , so replacing each graph with K_2 or K_3 still results in a flexible graph. There are just three graphs under this assumption, which are depicted in Figure 1. By Corollary 6, all three are rigid. \square

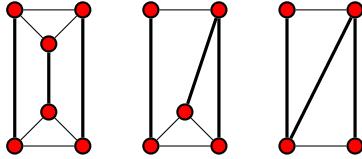


Figure 1: The three minimally rigid graphs in Theorem 8. The edge cuts are thickened.

The previous theorem is the best possible in terms of the number of edges in the cut, since for an edge cut of size 4, there are $|E_1|+|E_2|+4 \geq (2|V_1|-3)+(2|V_2|-3)+4 > 2|V|-3$ edges (we have strict inequality when $|V_1|$ or $|V_2|$ is 1), so one of the edges in the cut is redundant. Furthermore, we require that the e_i 's not meet at the same vertex for the same reason.

Using this result and those of the previous section, we obtain our first result for 4-regular graphs.

Theorem 9 *All connected 4-regular planar graphs that are not 4-edge-connected have rational Fáry embeddings.*

Proof. By a degree-counting argument, a 4-regular graph cannot have a minimal edge cut of size 3, so the edge cut must consist of two edges. Let G be a 4-regular planar graph that is not 4-edge-connected, and let ϕ be a Fáry embedding of G . We can perturb ϕ to a generic (and hence regular) Fáry embedding ϕ' . In the framework (G, ϕ') , the edges of the cut are independent by Theorem 8. There exists an open neighborhood around ϕ' consisting of only regular Fáry embeddings, so if our perturbations of ϕ' are suitably small, every edge of the cut stays independent.

Deleting the edge cut yields two almost 4-regular graphs G_1 and G_2 . By Corollary 4, each G_i can be approximated by a rational Fáry embedding. Combining these two approximations yields a Fáry embedding of G such that the only edges that are possibly not rational are those in the cut. By applying Lemma 7 on each edge of the cut, we obtain a rational Fáry embedding of G . \square

4 An Operation of Kemnitz and Harborth

The inductive step in proving Fáry's theorem possibly deletes edges and inserts a new k -valent vertex into the interior of the resulting polygon, as in Figure 2. We call this operation a k -addition if we start and end with

rational Fáry embeddings. Kemnitz and Harborth [8] tried to find k -additions for $k = 3, 4, 5$, following the proof of Fáry's theorem. Geelen *et al.* [4] remarked that Theorem 1 suffices for the case $k = 3$. For adding a vertex of degree 4 into a quadrilateral Q , Kemnitz and Harborth chose to place the new vertex on the diagonal so that one of the constraints is eliminated.

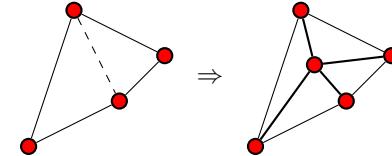


Figure 2: Adding a vertex of degree 4 after deleting an edge.

Consider a quadrilateral Q with a diagonal D of length f , as in Figure 3. Kemnitz and Harborth attempt to find a point P on D such that for rational lengths a, b, c, d , and f , the lengths x, y , and z are rational as well. They do not accomplish this for all quadrilaterals, though they always find a point on the line containing D . We briefly review their solution of the associated Diophantine equations.

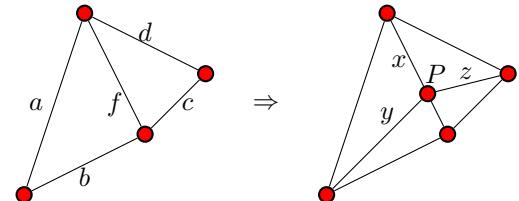


Figure 3: Variables for the Diophantine equations of Kemnitz and Harborth.

Let $s = \frac{y-a}{x}$ and $t = \frac{z-d}{x}$. Note that for nondegenerate Q , s and t cannot be ± 1 . We can express x as

$$x = \frac{2afs + a^2 + f^2 - b^2}{f(1-s^2)} = \frac{2df t + d^2 + f^2 - c^2}{f(1-t^2)}.$$

It suffices to find suitable values of s and t such that the second equality holds. Let

$$K = a^2 + f^2 - b^2$$

$$L = 2af$$

$$M = d^2 + f^2 - c^2$$

$$N = 2df.$$

t and s are related by

$$t = \frac{1}{2(K+Ls)}(N(s^2 - 1) \pm \sqrt{R})$$

where

$$R = N^2 s^4 + 4LM^3 + 2(2KM + 2L^2 - N^2)s^2 + 4L(2K - M)s + 4K(K - M)^2 + N^2.$$

We wish for \sqrt{R} to be rational, so if we let

$$\begin{aligned} q &= \sqrt{R}, \\ S &= 4LM, \\ T &= 2(2KM + 2L^2 - N^2), \\ U &= 4L(2K - M), \\ V &= 4K(K - M) + N^2, \end{aligned}$$

we obtain the Diophantine equation

$$N^2s^4 + Ss^3 + Ts^2 + Us + V = q^2,$$

which has already been solved in Mordell [10]. The solution of this equation gives one for the original Diophantine equation via substitution.

Theorem 10 (Kemnitz and Harborth [8]) *If $4N^2ST - S^3 - 8N^4U \neq 0$, then there exists a solution for P where x , y , and z are all rational that satisfies*

$$\begin{aligned} s &= \frac{64N^6V - 4(N^2T - S^2)^2}{8N^2(4N^2ST - S^3 - 8N^4U)} \\ q &= \frac{4N^2(2N^2s^2 + Ss + T) - S^2}{8N^3}. \end{aligned}$$

Kemnitz and Harborth analyze the case where the denominator of s vanishes, but for our purposes, Theorem 10 is sufficient. Unfortunately the solution to the Diophantine equation does not guarantee that P lies inside the quadrilateral, so it cannot be used as a general operation on rational Fary embeddings. For drawing 4-regular graphs with diamond subgraphs, we make use of *permissible* quadrilaterals, namely those where P does land on the diagonal. All we need is the existence of just one permissible quadrilateral.

Proposition 11 *The quadrilateral with lengths*

$$a = b = 3, \quad c = d = 4, \quad f = 5$$

is permissible.

Proof. Tracing through Theorem 10 yields a value of $x = 282240/357599$, which yields a point inside the quadrilateral. \square

Directly using Theorem 10 requires a 5-vertex wheel subgraph, but it turns out that we can relax the conditions slightly.

Proposition 12 *Theorem 10 still produces rational solutions even under the weaker condition that only b^2 and c^2 have to be rational.*

When we only require that b^2 and c^2 are rational when adding the new vertex, we call the operation a *generalized 4-addition*. For a fully-rational Fary embedding, the square of the distance between any two vertices is always rational, so Proposition 12 can be used when the corresponding edges are missing. Ultimately, we perform the generalized 4-addition on a slightly perturbed quadrilateral, so we need an additional result for quadrilaterals nearby.

Proposition 13 *Let Q be a permissible quadrilateral. There exists $\epsilon > 0$ such that every Q' satisfying $d(Q, Q') < \epsilon$ is also permissible.*

Proof. The solution for x is a continuous function of the edge lengths of Q , and hence a continuous function of the coordinates of the vertices. \square

5 4-Regular Graphs With Diamonds

We use the results of the previous section to find a rational Fary embedding of a 4-regular graph with a diamond subgraph. The *diamond graph* is the simple graph on four vertices and five edges, and the name comes from the common visualization as two triangles sharing an edge. For a 4-regular graph G with a diamond subgraph, label the vertices of that subgraph and the other neighbor of one of the 3-valent vertices as in the left-most graph in Figure 4. Let G' be the graph formed by deleting P from G and adding the edge v_2v_4 , and let G_T be the graph formed by deleting P and adding the edges v_1v_4 and v_3v_4 .

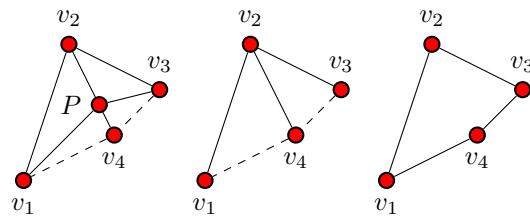


Figure 4: Local drawings of our graphs G , G' , G_T . Dashed lines are possibly missing edges.

The main idea of our construction is to perform a generalized 4-addition on a fully-rational Fary embedding of G' to get one of G , but some preliminary results are needed to ensure that the quadrilateral formed by the v_i 's is permissible and that adding P does not create any crossing edges. We want to show that the quadrilateral $Q = v_1v_2v_3v_4$ is a face in some planar embedding of G_T , and by using a modification of Tutte's spring theorem, we can devise a Fary embedding where Q is permissible and empty in the interior.

The figure suggests that v_1v_2P and v_2v_3P are faces in the planar embedding. Luckily for the graphs we consider, this is true for any planar embedding.

Proposition 14 *Let G be a planar 4-edge-connected 4-regular graph. Then, every K_3 subgraph bounds a face in any planar embedding of G .*

Proof. Consider any K_3 subgraph with vertices w_1, w_2 , and w_3 . The edges of the subgraph form a simple cycle C , so consider the remaining six edges incident with the w_i 's. We assert that the neighbors of w_1, w_2 , and w_3 , besides each other, are either all inside C or all outside C . If this is not the case, then there are $g > 0$ and $h > 0$ edges that are incident with vertices inside and outside of C , respectively. Since $g + h = 6$, either g or h is less than 4. However, this implies that one of those sets of edges is a cut of size less than 4, which is a contradiction. Thus, one of g or h must be 0, so C is a face. \square

Corollary 15 *Q is a face of some planar embedding of G_T .*

Proof. For any planar embedding of G , Proposition 14 implies that the cyclic rotation of vertices around P is v_1, v_2, v_3, v_4 or its mirror, so we may add the edges v_1v_4 and v_3v_4 into this embedding without violating planarity. Deleting P yields a planar embedding of G_T with Q as a face. \square

Now that we know that Q is a face of G_T , we need to draw it in the shape of a permissible quadrilateral. A well-known result in graph theory, sometimes referred to as Tutte's spring theorem, states that for a 3-connected plane graph G with exterior face F , we can make F whatever convex polygon we desire and obtain a Fary embedding with all convex interior faces. If the 3-connectedness condition is dropped, we can add vertices and edges into the graph to make the graph 3-connected, but the faces induced on the original graph might not be convex.

Theorem 16 (Tutte [12]) *Let G be a plane graph with a simple face F and a prescribed convex embedding $\phi_F : V(F) \rightarrow \mathbb{R}^2$ of F . Then, there exists a Fary embedding ϕ of G such that ϕ restricted on the vertices of F is equal to ϕ_F and F is the exterior face.*

Corollary 17 *Theorem 16 can be modified so that in ϕ , F is an interior face.*

Proof. Let $P : \mathbb{R}^2 \rightarrow S^2$ be the Riemann stereographic projection, where we view S^2 as the unit sphere centered at the origin of \mathbb{R}^3 and \mathbb{R}^2 as the hyperplane in \mathbb{R}^3 that is zero in the last coordinate. Define $r : S^2 \rightarrow S^2$ to be the reflection of the sphere across the plane \mathbb{R}^2 . Let U be the map $P^{-1}rP$, which is defined for all non-origin points in the plane. Intuitively, U “inverts” a Fary embedding, since any face containing the north pole in the sphere will now contain the south pole, and hence, that face goes from being exterior to interior.

Translate ϕ_F so that the origin lies inside the face. The embedding $\phi'_F = U\phi_F$ is an embedding of an inverted face F . Using Theorem 16 on ϕ'_F gives a Fary embedding ϕ' with the inverted F , so $\phi = U\phi'$ restricted to the vertices of F is ϕ_F . Furthermore, F is now an interior face of ϕ . \square

We now prove Conjecture 1 for the 4-regular graphs with diamonds.

Theorem 18 *Let G be a connected 4-regular planar graph with a diamond subgraph. Then, G has a rational Fary embedding.*

Proof. If G is not 4-edge-connected, the result follows from Theorem 9. Otherwise, Q is a face of G_T by Corollary 15. By using Corollary 17, we can construct a Fary embedding ϕ_T of G_T such that Q is empty and has the edge lengths prescribed in Proposition 11. ϕ_T is also a valid Fary embedding for G' since Q was drawn as a convex quadrilateral.

The vertex v_1 is 3-valent in G' , so G' is almost 4-regular. By Corollary 4, ϕ_T can be approximated by a fully-rational Fary embedding ϕ' . Q is still permissible in ϕ' by Proposition 13, and since ϕ' is fully-rational, Proposition 12 enables us to perform a generalized 4-addition on ϕ' , yielding a rational Fary embedding ϕ of G . \square

6 Conclusion

In this paper, we construct integral Fary embeddings of some 4-regular planar graphs, making progress on a conjecture of Biedl [2]. Perhaps surprisingly, one of the families we prove Conjecture 1 for has triangles close together, which seemingly make finding integral Fary embeddings difficult. The proof unfortunately does not extend to 4-regular graphs where the triangles are far apart. For every 4-regular planar graph, we can add an edge so that a diamond subgraph is formed, but undoing the generalized 4-addition operation does not always yield a 3-eliminable graph. Nonetheless, we believe that the techniques presented here can be extended to cover all 4-regular planar graphs.

References

- [1] T. Berry, *Points at rational distance from the vertices of a triangle*, Acta Arith. 62 (1992) No. 4, 391–398.
- [2] T. Biedl, *Drawing some planar graphs with integer edge lengths*, Proc. 23rd Canad. Conf. Comp. Geom. (2011), 291–296.
- [3] I. Fáry, *On straight-line representation of planar graphs*, Acta Sci. Math. 11 (1948), 229–233.

- [4] J. Geelen, A. Guo, D. McKinnon, *Straight line embeddings of cubic planar graphs with integer edge lengths*, J. Graph Theory 58 (2008) No. 3, 270-274.
- [5] J. Graver, B. Servatius, H. Servatius, *Combinatorial rigidity*, Grad. Stud. Math., Vol. 2, Amer. Math. Soc. (1993).
- [6] R. Guy, *Unsolved Problems In Number Theory*, 2nd Ed. Springer, New York (1994).
- [7] H. Harborth, A. Kemnitz, M. Möller, A. Sussenbach, *Ganzzahlige planare Darstellungen der platonischen Körper*, Elem. Math. 42 (1987), 118-122.
- [8] A. Kemnitz, H. Harborth, *Plane Integral Drawings of Planar Graphs*, Discrete Math. 236 (2001), 191-195.
- [9] G. Laman, *On graphs and rigidity of plane skeletal structures*, J. Eng. Math. 4 (1970), 331-340.
- [10] L. Mordell, *Diophantine Equations*, Academic Press, London (1969).
- [11] T. Sun, *Rigidity-theoretic constructions of integral Fary embeddings*, Proc. 23rd Canad. Conf. Comp. Geom. (2011), 287-290.
- [12] W. Tutte, *How to draw a graph*, Proc. London Math. Soc. 13 (1963), 743-767.

Hyperbanana Graphs

Christopher Clement*

Audrey Lee-St.John†

Jessica Sidman‡§

Abstract

A *bar-and-joint* framework is a finite set of points together with specified distances between selected pairs. In rigidity theory we seek to understand when the remaining pairwise distances are also fixed. If there exists a pair of points which move relative to one another while maintaining the given distance constraints, the framework is *flexible*; otherwise, it is *rigid*.

Counting conditions due to Maxwell give a necessary combinatorial criterion for generic minimal bar-and-joint rigidity in all dimensions. Laman showed that these conditions are also sufficient for frameworks in \mathbb{R}^2 . However, the flexible “double banana” shows that Maxwell’s conditions are not sufficient to guarantee rigidity in \mathbb{R}^3 . We present a generalization of the double banana to a family of *hyperbananas*. In dimensions 3 and higher, these are (infinitesimally) flexible, providing counterexamples to the natural generalization of Laman’s theorem.

1 Introduction

A *bar-and-joint framework* is composed of universal joints whose relative positions are constrained by fixed-length bars. An *embedding* of such a framework in \mathbb{R}^d associates a point in \mathbb{R}^d to each joint with the property that the distance between joints connected by a bar is satisfied by the embedding. Bar-and-joint frameworks can be used to model structures arising in many applications, including sensor networks, proteins, and Computer Aided Design (CAD) systems. In combinatorial rigidity theory we seek an understanding of the structural properties of such a framework, and ask whether it is flexible (i.e., admits an internal motion that respects the constraints) or rigid.

In this paper, we assume that we are given an embedding of a bar-and-joint framework from which the lengths of bars can be inferred.

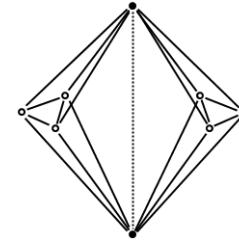


Figure 1: The double banana is a Maxwell graph in \mathbb{R}^3 , but is flexible. Each “banana” can rotate about the *implied hinge* (dotted).

Definition 1 A *bar-and-joint framework* $F = (G, \mathbf{p})$ embedded in \mathbb{R}^d is composed of a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ and an embedding $\mathbf{p} : V \rightarrow \mathbb{R}^d$, which assigns a position vector \mathbf{p}_i to each vertex v_i .

We only concern ourselves with *generic embeddings* of these frameworks, which can be thought of as embeddings with the properties we would expect if we chose an embedding at random. To formally define genericity we require the notion of a **rigidity matrix**, which encodes the infinitesimal behavior of the framework.

Definition 2 For a framework $F = (G, \mathbf{p})$ embedded in \mathbb{R}^d we define a rigidity matrix M_F to be an $m \times dn$ matrix in which the columns are grouped into n sets of d coordinates for each vertex. Each row of the rigidity matrix corresponds to an edge ij and has the following pattern.

$$ij \quad \begin{matrix} v_1 & \dots & v_i & \dots & v_j & \dots & v_n \\ [0 & \dots & 0 & \dots & \mathbf{p}_i - \mathbf{p}_j & \dots & 0 & \dots & \mathbf{p}_j - \mathbf{p}_i & \dots & 0 & \dots & 0] \end{matrix}$$

If F is a framework, M_F determines if it is *infinitesimally* flexible or rigid; for brevity, we omit “infinitesimally” for the remainder of this paper. We say that F is *rigid* if the insertion of any new bar between vertices does not change the rank of M_F ; otherwise it is *flexible*. A rigid framework is *minimally rigid* if the rows of M_F are independent.

The *infinitesimal motions* of F can be encoded by assigning a velocity vector $\mathbf{p}'_i \in \mathbb{R}^d$ to each vertex v_i so that $(\mathbf{p}'_1, \dots, \mathbf{p}'_n)$ is nonzero and is in the null space of M_F (intuitively, these are instantaneous velocities that do not shrink or stretch the bar constraints). There is always a set of *trivial motions* corresponding to rigid body motions of \mathbb{R}^d ; the space of rigid motions of \mathbb{R}^d has dimension $\binom{d+1}{2}$ and is generated by rotations about $(d-2)$ -dimensional affine linear subspaces

*Department of Mathematics, University of Michigan, cclement@umich.edu

†Department of Computer Science, Mount Holyoke College, astjohn@mtholyoke.edu. Research partially supported by the Clare Boothe Luce Foundation.

‡Department of Mathematics & Statistics, Mount Holyoke College, jsidman@mtholyoke.edu

§All three authors were partially supported by NSF grant DMS-0849637.

and translations. In general, then, a framework on at least d vertices is minimally rigid if and only if its rigidity matrix has nullity $\binom{d+1}{2}$. However, if a framework F is contained in an affine subspace $H \subset \mathbb{R}^d$ where $\dim H \leq d-2$, then there is a rigid motion of \mathbb{R}^d that fixes F ; hence, the null space of M_F has dimension less than $\binom{d+1}{2}$.

Combinatorial counting conditions, first observed by Maxwell [5], give a necessary condition for minimal bar-and-joint rigidity. Throughout this paper, we will use the convention that, if V' is a subset of the vertices of a graph G and \mathcal{E} is a subset of the edges of G , then $\mathcal{E}(V')$ is the set of edges in \mathcal{E} induced by the vertices in V' .

Definition 3 A Maxwell graph $G = (V, E)$ in dimension d satisfies

1. $|E| = d|V| - \binom{d+1}{2}$
2. $|E(V')| \leq d|V'| - \binom{d+1}{2}$, for all $V' \subseteq V$ where $|V'| \geq d$.

For almost all frameworks $F = (G, \mathbf{p})$ on a fixed graph G , the rank of M_F is constant, as the set of special embeddings for which M_F drops rank is parameterized by a closed subset of \mathbb{R}^{dn} . We formally define genericity as follows.

Definition 4 A framework (G, \mathbf{p}) is generic if its rigidity matrix achieves the maximum rank over all frameworks (G, \mathbf{q}) .

We call a framework *generically minimally rigid* if there exists a generic framework with the same underlying graph that is minimally rigid. We analyze the generic behavior of a framework purely by the combinatorial structure of the graph. Therefore, from here on we will write M_G to denote the rigidity matrix associated to a generic embedding of G .

In \mathbb{R}^2 , Laman proved that the Maxwell conditions are sufficient for generic minimal rigidity.

Theorem 5 (Laman [3]) A bar-and-joint framework, with underlying graph $G = (V, E)$, embedded in \mathbb{R}^2 is generically minimally rigid if and only if it satisfies the following conditions:

1. $|E| = 2|V| - 3$
2. $|E(V')| \leq 2|V'| - 3$, for all $V' \subseteq V$ where $|V'| \geq 2$

However, the sufficiency of the Maxwell counting conditions for rigidity does not generalize to higher dimensions. In \mathbb{R}^3 , the well-known “double banana” is a Maxwell graph that is flexible [2]. This structure is composed of two “bananas” joined on a pair of vertices (refer to Figure 1) and exhibits a hinge motion about

the dotted line. This denotes the existence of an *implied edge* between two vertices that are not incident to each other, yet whose distance is fixed as a consequence of the other constraints. Since a rotation is allowed about the edge, it is called an *implied hinge*.

Counterexamples like the double banana can provide insight into the challenges presented in dimension 3 and higher for which no combinatorial characterization of bar-and-joint rigidity is known.

Contributions. In this paper, we describe a class of graphs called *hyperbananas* that generalize the double banana to higher dimensions. We present hyperbananas that are Maxwell graphs and show these to be (infinitesimally) flexible. To the best of our knowledge, this is the first family of counterexamples to the sufficiency of the Maxwell conditions for minimal bar-and-joint rigidity addressing all dimensions of 3 and higher.

Related work. Other generalizations of the double banana include the banana spider graphs of Mantler and Snoeyink [4]. These were developed to address an attempt at classifying 3D bar-and-joint rigidity by vertex connectivity, as it was conjectured that all graphs with implied hinges must be 2-connected (like the double banana). The banana spider graphs provide examples with higher vertex connectivity, answering this conjecture in the negative. The key idea was to add “spider” components to the double banana, increasing vertex connectivity while maintaining flexibility about the implied hinge.

Another class of counterexamples to Maxwell’s conditions in 3D was developed by Cheng et al. [1]. These “ring of roofs” frameworks, first described by Tay [7], provide examples of flexible Maxwell graphs that admit no non-trivial rigid subgraphs, i.e., rigid subgraphs larger than a tetrahedron. This countered an earlier attempt by Sitharam and Zhou [6] to characterize 3D bar-and-joint rigidity by detecting rigid components and adding the resulting implied edges.

2 Maxwell hyperbananas

We now present a family of graphs called hyperbanana graphs; under certain conditions, hyperbananas are Maxwell graphs. We generalize the double banana, which consists of two minimally rigid “bananas” glued together on a pair of vertices. Each banana can be built using the following inductive construction.

Definition 6 Fix a positive integer d . A d -Henneberg 0-extension on a graph G results in a new graph by adding a single vertex and connecting it to d distinct vertices in G .

When a d -Henneberg 0-extension is applied to a minimally rigid framework in \mathbb{R}^d , minimal rigidity is preserved, and hence so are the Maxwell conditions [8]. In

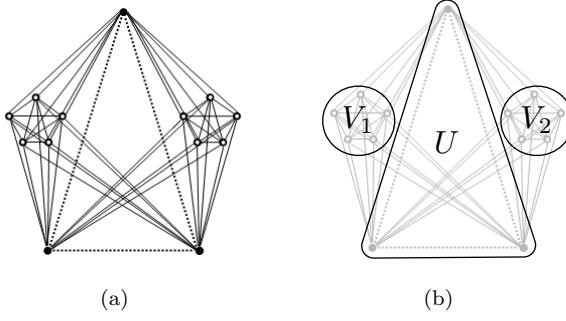


Figure 2: The hyperbanana $H_{5,3}$ is a flexible Maxwell graph \mathbb{R}^5 ; there are 3 implied edges (dotted) among the vertices in U .

the double banana, each individual banana is created by two 3-Henneberg 0-extensions on a triangle (which is minimally rigid in \mathbb{R}^3), connecting each new vertex to the 3 vertices of the triangle.

Before generalizing the banana construction, we give some additional notation. If U and W are finite sets, let K_U denote the complete graph with vertex set U and $K_{U,W}$ be the complete bipartite graph on the two disjoint sets U and W .

Definition 7 A banana bunch is a graph $B_{d,b}$ obtained by performing b d -Henneberg 0-extensions on a K_d . The b vertices added by the Henneberg extensions are called banana vertices.

Since K_d embedded in \mathbb{R}^d is minimally rigid for any d , $B_{d,b}$ is generically minimally rigid in dimension d .

Hyperbananas are composed of two banana bunches glued together along the banana vertices.

Definition 8 For $i = 1, 2$, let G_i be a copy of $B_{d,b}$ with vertex set partitioned into $V_i \cup U_i$, where the K_d has vertex set V_i and the set U_i consists of banana vertices. We define the hyperbanana $H_{d,b}$ to be $G_1 \cup G_2 / \sim$, where \sim identifies banana vertices based on some fixed bijection from U_1 to U_2 . The vertex set of $H_{d,b}$ is the set $V = V_1 \cup V_2 \cup U$, where U is the set of banana vertices.

The double banana is simply $H_{3,2}$. An example of a higher dimensional hyperbanana, $H_{5,3}$, is pictured in Figure 2. While this is a Maxwell graph, not all choices of b and d satisfy the counting conditions. For example, simply checking the counts on the total number of edges for the hyperbanana $H_{4,3}$ confirms that this graph has too many edges to be Maxwell. In fact, it is rigid in \mathbb{R}^4 , but overconstrained. Therefore, it is not minimally rigid as its rigidity matrix contains dependencies. Checking the counts on the total number of edges for the hyperbanana $H_{6,3}$ shows that it is underconstrained and therefore flexible in \mathbb{R}^6 .

2.1 Odd-dimensional hyperbananas

When d is odd and equal to $2b - 1$, we obtain hyperbananas that are Maxwell graphs. We begin with a more general lemma that will be used in proving the counting conditions. In the proofs that follow, we define $V'_i = V' \cap V_i$ and $U' = V' \cap U$ for a subset V' of the vertex set of $H_{d,b}$,

Lemma 9 If $H_{d,b} = (V, E)$ and $V' \subseteq V$, and $|V'_i \cup U'| \geq d$ for $i = 1, 2$, then

$$|E(V')| \leq d|V'| - 2\binom{d+1}{2} + d|U'|.$$

Proof. As each banana bunch is minimally rigid we have

$$|E(V'_i \cup U')| \leq d|V'_i \cup U'| - \binom{d+1}{2} \quad (1)$$

for each i . Adding the inequalities yields

$$\begin{aligned} |E(V')| &\leq d(|V'_1| + |V'_2| + 2|U'|) - 2\binom{d+1}{2} \\ &= d(|V'_1| + |V'_2| + |U'|) - 2\binom{d+1}{2} + d|U'| \\ &= d|V'| - 2\binom{d+1}{2} + d|U'|. \end{aligned} \quad (2)$$

□

We can now show that the specific class of hyperbananas in odd-dimensional spaces are Maxwell graphs.

Theorem 10 The hyperbanana $H_{d,b}$ embedded in \mathbb{R}^d with $d = 2b - 1$ is a Maxwell graph.

Proof. We check condition 1 of Definition 3 by vertex and edge counts. The graph $H_{d,b}$ has d vertices from each complete K_d graph and b banana vertices, totaling $2d + b$ vertices. Since $d = 2b - 1$, there are $\frac{5d+1}{2}$ vertices. Each K_d has $\binom{d}{2}$ edges, and each banana vertex is incident to $2d$ edges. This sums to an edge count of $2\binom{d}{2} + 2d\binom{d+1}{2}$. Simplifying, we can verify that the edge count is $|E| = 2d^2$. Substituting the vertex count $|V| = \frac{5d+1}{2}$, we see that Maxwell condition 1 is satisfied:

$$d|V| - \binom{d+1}{2} = d\left(\frac{5d+1}{2}\right) - \binom{d+1}{2} = |E|$$

Now we check Maxwell condition 2. If V' is contained within a single banana bunch, the condition is satisfied as $B_{d,b}$ is minimally rigid and therefore Maxwell. If V' intersects both banana bunches non-trivially, then there are three cases which depend on whether the intersection with each banana bunch contains at least d vertices.

If $|V'_i \cup U'| \geq d$ for both i , then combining $|U'| \leq b = \frac{d+1}{2}$ with Lemma 9 gives the result.

Now suppose, without loss of generality, that $|V'_1 \cup U'| \geq d$, but $|V'_2 \cup U'| < d$. We know that

$$|E(V'_2 \cup U')| = \binom{|V'_2|}{2} + |U'||V'_2| \quad (3)$$

$$= \frac{(|V'_2| - 1)|V'_2|}{2} + |U'||V'_2| \quad (4)$$

$$\leq \frac{(d-2)|V'_2|}{2} + b|V'_2| \quad (5)$$

Since $b = \frac{d+1}{2}$, we obtain $|E(V'_2 \cup U')| \leq d|V'_2|$. Combining this with Inequality 1 gives the desired inequality in the second case.

Finally, suppose that both $|V'_i \cup U'| < d$ and $|V'_1| \geq |V'_2|$. As $|V'_1 \cup V'_2 \cup U'| \geq d$, there exists a subset $W \subseteq V'_2$ so that $|V'_1 \cup W \cup U'| = d$. Let $W' = V'_2 \setminus W$. The set $|E(V'_2)|$ consists of the edges of K_W , the edges of $K_{W'}$ and the edges of $K_{W,W'}$.

Now suppose we had a set V''_1 satisfying $V_1 \supseteq V''_1 \supset V'_1$ and $|V''_1 \cup U'| = d$. Then

$$\begin{aligned} |E(V'_1 \cup W \cup U')| + |E(K_{V'_1, W})| &= |E(V''_1 \cup U')|, \\ \text{and by Inequality 1,} \\ |E(V'_1 \cup W \cup U')| + |E(K_{V'_1, W})| &\leq d|V'_1 \cup W \cup U'| - \binom{d+1}{2}. \end{aligned} \quad (6)$$

Applying the argument in the second case to the set $W' \cup U'$ and adding the inequality to 6, gives the result in this final case as $|E(K_{W,W'})| < |E(K_{V'_1, W})|$. \square

2.2 Even-dimensional hyperbananas

We observed earlier that hyperbananas may be either overconstrained or underconstrained in even-dimensional spaces and are not Maxwell graphs. However, by making a small modification to our definition, we obtain Maxwell graphs for even-dimensional spaces.

Definition 11 For even d , we define the even hyperbanana to be a graph $H_{d,b}^+$ consisting of a hyperbanana $H_{d,b}$ together with an additional $\frac{d}{2}$ edges connecting distinct vertices of the complete graphs in the two banana bunches.

This addition of $\frac{d}{2}$ edges between the complete graphs in $H_{d,b}$ results in $H_{d,b}^+$ being a Maxwell graph for the even-dimensional spaces for certain values of d relative to b . One example of an even hyperbanana, $H_{4,2}^+$, is shown in Figure 3. Note that $H_{d,b}^+ = (V, F)$, is built from $H_{d,b} = (V, E)$; let E^+ be the additional $\frac{d}{2}$ edges so that $F = E \cup E^+$. In Figure 3, for example, E^+ is composed of the 2 dashed edges.

Theorem 12 The even hyperbanana $H_{d,b}^+ = (V, F)$ embedded in \mathbb{R}^d with $d = 2b$ is a Maxwell graph.

Proof. Since $d = 2b$, the number of vertices in $H_{d,b}^+$ is $|V| = 2d + b = \frac{5}{2}d$, as there are two K_d graphs and b banana vertices. There are 2 complete graphs with $\binom{d}{2}$ edges, b banana vertices connecting to the $2d$ complete graph vertices, and $\frac{d}{2}$ edges between the complete graphs, resulting in $|F| = 2d^2 - \frac{d}{2}$. By substituting the vertex count, we can verify Maxwell condition 1.

$$d|V| - \binom{d+1}{2} = 2d^2 - \frac{d}{2} = |F|.$$

Now let $V' \subseteq V$ with $|V'| \geq d$. If V' is completely contained in a banana bunch, Maxwell condition 2 is

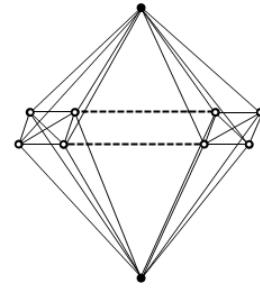


Figure 3: The even hyperbanana $H_{4,2}^+$ is a flexible Maxwell graph; it is built from the hyperbanana $H_{4,2}$ by an additional 2 (dashed) edges.

satisfied as $B_{d,b}$ is Maxwell. Assume, then, that V' non-trivially intersects both vertex sets V_1 and V_2 .

If $|V'_i \cup U'| \geq d$ for both $i = 1, 2$, then by Lemma 9,

$$|E(V')| \leq d|V'| - 2\binom{d+1}{2} + d|U'|.$$

The number of banana vertices is $b = \frac{d}{2}$, so $|U'| \leq \frac{d}{2}$. Therefore,

$$\begin{aligned} |E(V')| &\leq d|V'| - 2\binom{d+1}{2} + \frac{d^2}{2} \\ &= d|V'| - \binom{d+1}{2} - \frac{d^2+d}{2} + \frac{d^2}{2} \\ &= d|V'| - \binom{d+1}{2} - \frac{d}{2}. \end{aligned}$$

Since $F = E \cup E^+$, $|F(V')| = |E(V')| + |E^+(V')|$. By adding $|E^+(V')|$ to both sides of the previous inequality we obtain

$$|F(V')| \leq d|V'| - \binom{d+1}{2} - \frac{d}{2} + |E^+(V')|.$$

By definition, $|E^+| = \frac{d}{2}$, implying $|E^+(V')| \leq \frac{d}{2}$. Therefore, we can conclude that Maxwell condition 2,

$$|F(V')| \leq d(|V'|) - \binom{d+1}{2},$$

holds in this case.

Now suppose, without loss of generality, that $|V'_1 \cup U'| \geq d$, but $|V'_2 \cup U'| < d$. Since $b = \frac{d}{2}$, Inequality 5 implies

$$|E(V'_2 \cup U')| \leq (d-1)|V'_2|. \quad (7)$$

We can combine this with

$$|E(V'_1 \cup U')| \leq d|V'_1 \cup U'| - \binom{d+1}{2}$$

and the edges in $E^+(V')$ to obtain

$$\begin{aligned} |F(V')| &\leq d|V'_1 \cup U'| - \binom{d+1}{2} + (d-1)|V'_2| + |E^+(V')| \\ &= d|V'| - \binom{d+1}{2} - |V'_2| + |E^+(V')| \\ &\leq d|V'| - \binom{d+1}{2} \end{aligned}$$

as $|E^+(V')| \leq |V'_2|$.

Finally, suppose that both $|V'_i \cup U'| < d$. Assume that $|V'_1| \geq |V'_2|$ and define W and W' as in the proof of Theorem 10. Adding Inequalities 6 and 7 (with W'

replacing V'_2 ,

$$\begin{aligned} & |E(V'_1 \cup W \cup U')| + |E(K_{V'_1, W})| + |E(W' \cup U')| \\ & \leq d|V'_1 \cup W \cup U'| - \binom{d+1}{2} + (d-1)|W'| \\ & = d|V'| - \binom{d+1}{2} - |W'|, \end{aligned}$$

and hence

$$\begin{aligned} & |E(V'_1 \cup W \cup U')| + |E(K_{V'_1, W})| + |E(W' \cup U')| + |W'| \\ & \leq d|V'| - \binom{d+1}{2}. \end{aligned}$$

Since $|F(V')|$ is equal to

$|E(V'_1 \cup W \cup U')| + |E(K_{W, W'})| + |E(W' \cup U')| + |E^+(V')|$, it will suffice to show that

$$|E(K_{W, W'})| + |E^+(V')| \leq |E(K_{V'_1, W})| + |W'|,$$

or that

$$|W| \cdot |W'| + |E^+(V')| \leq |V'_1| \cdot |W| + |W'| \quad (8)$$

Now let $t = |V'_1| - |W'|$. Since $|V'_1| \geq |V'_2|$ and $|V'_1| < d$, $|W| > 0$, which implies that $|V'_1| > |W'|$ and hence that $t \geq 1$. Setting $|W'| = |V'_1| - t$, we have

$$\begin{aligned} & |W| \cdot |W'| + |E^+(V')| \\ & = |W| \cdot (|V'_1| - t) + |E^+(V')| \\ & = |V'_1| \cdot |W| - t|W| + |E^+(V')| \\ & \leq |V'_1| \cdot |W| - |W| + |E^+(V')|, \end{aligned}$$

as $t \geq 1$. Then

$$|V'_1| \cdot |W| - |W| + |E^+(V')| \leq |V'_1| \cdot |W| + |W'|$$

if and only if

$$|V'_1| \cdot |W| + |E^+(V')| \leq |V'_1| \cdot |W| + |W'| + |W|.$$

Indeed, since $|W'| + |W| = |V'_2| \geq |E^+(V')|$, this inequality holds, completing the proof. \square

3 Flexible hyperbananas

In this section, we prove that the Maxwell hyperbananas are flexible.

We begin by considering the rigidity matrix $M_{B_{d,b}}$ for a generic framework on the banana bunch $B_{d,b}$ in dimension d , which has $d(d+b)$ columns and $\binom{d}{2} + db$ rows. Since the banana bunch is minimally rigid, the rank of its rigidity matrix is maximal and equal to the number of rows $\binom{d}{2} + db$. Let the vertex set of $B_{d,b}$ be partitioned into sets V_1 and U , where the set U consists of banana vertices. Assume that the columns of $M_{B_{d,b}}$ are arranged so that the columns corresponding to the vertices in V_1 come first, followed by the columns for U .

Lemma 13 *Each row of the block matrix*

$$\left[\begin{array}{c|c} 0 & M_{K_U} \end{array} \right]$$

with d^2 columns of zeros (d columns for each vertex in the V_1), is in the row space of $M_{B_{d,b}}$.

Proof. Since the banana bunch is minimally rigid and spans \mathbb{R}^d , $M_{B_{d,b}}$ has nullity $\binom{d+1}{2}$. If we add an edge from K_U , the new rigidity matrix will still have nullity $\binom{d+1}{2}$. Thus, each such row must be a linear combination of the rows of $M_{B_{d,b}}$. \square

Proposition 14 *If $B_{d,b} = (V_1 \cup U, E)$ is embedded in \mathbb{R}^d , and the rank of M_{K_U} is $\binom{b}{2}$, then $M_{B_{d,b}}$ is row-equivalent to a matrix of the form*

$$\left[\begin{array}{c|c} M_{B_{d,b}}^* & \\ \hline 0 & M_{K_U} \end{array} \right],$$

where $M_{B_{d,b}}^$ consists of $|E| - \binom{b}{2}$ rows of the original matrix $M_{B_{d,b}}$.*

Proof. Let R be a row in $[0 \mid M_{K_U}]$. By Lemma 13, R may be written as a linear combination of rows of $M_{B_{d,b}}$. Any row of $M_{B_{d,b}}$ appearing in such a linear combination with a nonzero coefficient may be replaced by R through a sequence of elementary row operations. Any subsequent row R' of $[0 \mid M_{K_U}]$ will remain dependent on the rows of the modified matrix. Moreover, when we express R' as a linear combination of the current set of rows, some remaining row of the original matrix $M_{B_{d,b}}$ must appear with a nonzero coefficient as the rows of M_{K_U} are independent. Thus, we can insert each row of $[0 \mid M_{K_U}]$ in this way. \square

With this we can prove the following theorem.

Theorem 15 *If G is the hyperbanana $H_{d,b} \subset \mathbb{R}^d$ where $d = 2b - 1$ or $H_{d,b}^+ \subset \mathbb{R}^d$ where $d = 2b$ and $b \geq 2$, then G is flexible.*

Proof. Consider the hyperbanana $H_{d,b}$ partitioned into two bunches $B_{d,b}(1)$ and $B_{d,b}(2)$. Let $M_{B_{d,b}}(1)$ be the rigidity matrix for $B_{d,b}(1)$, $M_{B_{d,b}}(2)$ be the rigidity matrix for $B_{d,b}(2)$ and M be the rigidity matrix for $H_{d,b}$. If we put the vertices in an order with (V_1, U, V_2) and order the columns of M accordingly, then M is a block matrix of the form

$$\left[\begin{array}{c|c|c} V_1 & U & V_2 \\ \hline \hline B_{d,b}(1) & M_{B_{d,b}}(1) & 0 \\ \hline B_{d,b}(2) & 0 & M_{B_{d,b}}(2) \end{array} \right]$$

By Proposition 14 M is row equivalent to

$$\left[\begin{array}{c|c|c} V_1 & U & V_2 \\ \hline \hline B_{d,b}(1) & M_{B_{d,b}}(1)^* & 0 \\ \hline \hline B_{d,b}(2) & 0 & M_{B_{d,b}}(2)^* \\ \hline & M_{K_U} & 0 \end{array} \right]$$

We can see that there are at least $\binom{b}{2}$ dependencies in M , since the $[0|M_{K_U}|0]$ is seen twice in the matrix. Therefore, since the number of columns is $d|V|$ and the number of rows is $|E|$, the nullity of M is at least $\binom{d+1}{2} + \binom{b}{2}$. Thus, since a framework with at least d vertices is minimally rigid in \mathbb{R}^d if and only if it has nullity $\binom{d+1}{2}$, $H_{d,b}$ is flexible. Moreover, since M is a submatrix of

the rigidity matrix of $H_{d,b}^+$, which satisfies the Maxwell counts, we see that $H_{d,b}^+$ is also flexible. \square

For odd-dimensional bananas, we can show this bound is tight using the following proposition.

Proposition 16 *Any linear combination of rows of $M_{B_{d,b}}^*$ of the form*

$$\begin{array}{c|c} V_1 & U \\ \hline 0 & * \end{array},$$

*must be trivial, where the * represents potentially nonzero entries.*

Proof. Suppose for contradiction that there is a linear combination of rows of $M_{B_{d,b}}^*$ equal to R where R has nonzero entries only in columns corresponding to U . Let \bar{R} be the projection of R to the columns corresponding to U .

If \bar{R} is dependent on the rows in M_{K_U} , then the rank of $M_{B_{d,b}}$ is not maximal, which is a contradiction. So, we must assume that \bar{R} is independent of these rows. Thus, the nullspace of M_{K_U} augmented by the row \bar{R} is smaller than the nullspace of M_{K_U} . But all of the elements of the nullspace of M_{K_U} are obtained from rigid motions of \mathbb{R}^d . So there is a nonzero vector $\mathbf{p}' \in \mathbb{R}^{db}$ in the null space of M_{K_U} which assigns velocities to vertices in U and has the property that $\bar{R} \cdot \mathbf{p}' \neq 0$.

Since K_U is rigid, \mathbf{p}' must be obtained by restricting a rigid motion of \mathbb{R}^d to K_U . Applying this rigid motion to all of $B_{d,b}$ gives a vector \mathbf{q}' that assigns velocities to all vertices in $B_{d,b}$ and is equal to \mathbf{p}' for the vertices in U . As R has nonzero entries only in columns corresponding to U , $\mathbf{q}' \cdot R = \mathbf{p}' \cdot R \neq 0$. R is in the row space of $M_{B_{d,b}}$, so this implies that the nullspace of $M_{B_{d,b}}$ is missing one of the rigid motions of \mathbb{R}^d . This is a contradiction because $M_{B_{d,b}}$ is a rigidity matrix. \square

Theorem 17 *The hyperbanana $H_{d,b} \subset \mathbb{R}^d$ where $d = 2b - 1$ has rigidity matrix $M_{H_{d,b}}$ with nullity exactly $\binom{d+1}{2} + \binom{b}{2}$.*

Proof. We will show that

$$M' = \frac{1}{B_{d,b}(1)} \begin{bmatrix} V_1 & U & V_2 \\ \hline M_{B_{d,b}}(1)^* & 0 & 0 \\ \hline 0 & M_{K_U} & \\ \hline B_{d,b}(2) & 0 & M_{B_{d,b}}(2)^* \end{bmatrix}$$

has full rank and hence nullity $\binom{d+1}{2} + \binom{b}{2}$.

Since $M_{B_{d,b}}(1)$ has full rank, we know that the top block of M' has linearly independent rows. Similarly, the rows in $[0|M_{B_{d,b}}(2)^*]$ are also an independent set.

Now suppose there is a row $R \in [0|M_{B_{d,b}}(2)^*]$ that is dependent on the upper block of M' ; then R is a linear combination of the rows of $[M_{B_{d,b}}(1)^*|0]$ and $[0|M_{K_d}|0]$. There must be at least one row of $[M_{B_{d,b}}(1)^*|0]$ with a

nonzero coefficient or we would contradict the independence of $[0|M_{B_{d,b}}(2)]$. Since R is zero in the columns corresponding to vertices in V_1 , this implies that there is a linear combination of rows of $[M_{B_{d,b}}(1)^*|0]$ that is nonzero only in the banana vertex columns, which contradicts Proposition 16. \square

4 Conclusions and Future Work

We presented a family of hyperbanana graphs and showed that they are Maxwell graphs under certain conditions. We further proved that they are flexible, providing counterexamples to the sufficiency of the Maxwell counts for bar-and-joint rigidity in dimensions 3 and higher.

For hyperbananas embedded in odd-dimensional spaces, we gave a precise analysis of the space of infinitesimal motions. However, it remains an open problem to give an exact analysis for the even hyperbananas, as the addition of the $\frac{d}{2}$ edges prevents us from extending our proof. Based on Mathematica calculations on randomized embeddings of even hyperbananas, we conjecture the following:

Conjecture 1 *The even hyperbanana $H_{d,b}^+ \in \mathbb{R}^d$ where $d = 2b$ and $b \geq 2$ has a rigidity matrix with nullity exactly $\binom{d+1}{2} + \binom{b}{2}$.*

Since counterexamples provide an increased understanding of barriers to finding combinatorial characterizations of higher-dimensional bar-and-joint rigidity, it would also be interesting to further generalize the hyperbananas by parametrizing the number of banana bunches instead of always gluing two.

References

- [1] Jialong Cheng, Meera Sitharam, and Ileana Streinu. Nucleation-free 3d rigidity. In *CCCG*, pages 71–74, 2009.
- [2] Henry Crapo. Structural rigidity. *Structural Topology*, (1):26–45, 1979.
- [3] Gerard Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4:331–340, 1970.
- [4] Andrea Mantler and Jack Snoeyink. Banana spiders: A study of connectivity in 3d combinatorial rigidity. In *CCCG*, pages 44–47, 2004.
- [5] J. C. Maxwell. On the calculation of the equilibrium and stiffness of frames. *Philosophical Magazine*, 27:294, 1864.
- [6] Meera Sitharam and Yong Zhou. A tractable, approximate characterization of combinatorial rigidity in 3d. *5th Automated Deduction in Geometry (ADG)*, 2004.
- [7] Tiong-Seng Tay. On generically dependent bar frameworks in space. *Structural Topology*, (20):27–48, 1993.
- [8] Tiong-Seng Tay and Walter Whiteley. Generating isostatic frameworks. *Structural Topology*, (11):21–69, 1985.

Theta-3 is connected

Oswin Aichholzer* Sang Won Bae † Luis Barba ‡ § Prosenjit Bose ‡ Matias Korman ¶
 André van Renssen ‡ Perouz Taslakian ‡ Sander Verdonschot ‡

Abstract

In this paper, we show that the θ -graph with three cones is connected. We also provide an alternative proof of the connectivity of the Yao-graph with three cones.

1 Introduction

Introduced independently by Clarkson [6] in 1987 and Keil [9] in 1988, the θ -graph of a set P of points in the plane is constructed as follows. We consider each point $p \in P$ and partition the plane into $m \geq 2$ cones (regions in the plane between two rays originating from the same point) with apex p , each defined by two rays at consecutive multiples of $2\pi/m$ radians from the negative y -axis. We label the cones C_0 through C_{m-1} , in clockwise order around p , starting from the cone containing the positive y -axis from p if m is odd, or having this axis as its left boundary if m is even; see Figure 1. If the apex is not clear from the context, we use C_i^p to denote the cone C_i with apex p . We sometimes referred to C_i^p as the i -cone of p . To build the θ -graph, we consider each point p and connect it by an edge with the *closest* point in each of its cones. We measure distance by projecting each point onto the bisector of that cone instead of using the Euclidean distance. We use this definition of distance in the remainder of the paper, except for Section 4, which deals with Yao graphs. For simplicity, we assume that no two points of P lie on a line parallel to either the

*Institute for Software Technology, Graz University of Technology. Research of OA partially supported by the ESF EUROCORES programme EuroGIGA - CRP ‘ComPoSe’, Austrian Science Fund (FWF): I648-N18.

†Department of Computer Science, Kyonggi University. Work by S.W. Bae was supported by the Contents Convergence Software Research Center funded by the GRRC Program of Gyeonggi Province, South Korea.

‡School of Computer Science, Carleton University. Research supported in part by NSERC.

§Boursier FRIA du FNRS, Département d’Informatique, Université Libre de Bruxelles

¶Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya. Received support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia, the European Union, and projects MINECO MTM2012-30951, Gen. Cat. DGR2009SGR1040, ESF EUROCORES programme EuroGIGA – CRP ‘ComPoSe’: MICINN Project EUI-EURC-2011-4306.

||College of Science and Engineering, American University of Armenia

boundary or the angle bisector of a cone, guaranteeing that each point connects to at most one point in each cone. We call the θ -graph with m cones the θ_m -graph.

For θ -graphs with an even number of cones, proving connectedness is easy. As the first $m/2$ cones cover exactly the right half-plane, each point will have an edge to a point to its right, if such a point exists. Thus, we can find a path from any point to the rightmost point and, by concatenating these, a path between any pair of points. Unfortunately, if m is odd this property does not hold, as no set of cones covers *exactly* the right half-plane. Therefore, a point is not guaranteed to have an edge to a point to its right, even if such point exists.

The fact that θ -graphs with more than 6 cones are connected has been known for a long time. In fact, they even guarantee the existence of a *short* path between every pair of points. The length of this path is bounded by a constant times the straight-line Euclidean distance between the two points [3, 5, 6, 9, 11]. Graphs that have this property are called *geometric t-spanners* for some constant $t > 0$. For more information on geometric t -spanners, see the book by Narasimhan and Smid [10].

For a long time, very little was known about θ -graphs with fewer than 7 cones. Bonichon *et al.* [2] broke ground in this area in 2010, by showing that the θ_6 -graph is a geometric spanner. Subsequently, both the θ_4 - and θ_5 -graphs have been shown to be constant spanners [1, 4]. El Molla [8] already showed that the θ_2 - and θ_3 -graphs are not constant spanners. The θ_3 -graph is the last θ -graph for which connectedness has not been proven. In this paper, we settle this question by proving that the θ_3 -graph is always connected.

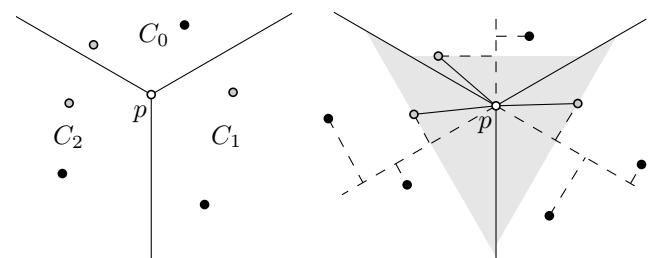


Figure 1: Left: A point p and its three cones in the θ_3 -graph. Right: Point p adds an edge to the closest point in each of its cones, where distance is measured by projecting points onto the bisector of the cone.

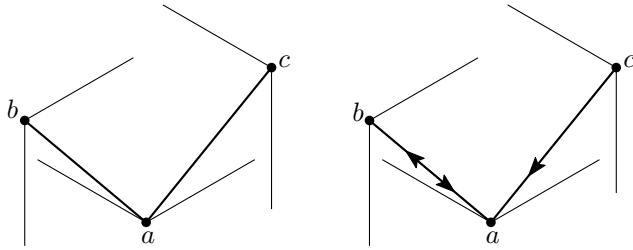


Figure 2: Left: A point set for which θ -routing does not find a path from a to c , as it keeps cycling between a and b . Right: The directed version of the graph is not strongly connected, as there is no path from either a or b to c .

The question of connectedness about the θ_3 graph is interesting because the θ_3 -graph has some unique properties that cause standard proof techniques for θ -graphs to fail. As such, we hope that the techniques we develop here will lead to more insight into the structure of other θ -graphs. As an example, most proofs for a larger number of cones show that the θ -routing algorithm (always follow the edge in the cone that contains the destination) returns a short path between any two points. But in the θ_3 -graph, θ -routing is not guaranteed to ever reach the destination. The smallest point set that exhibits this behavior has three points, such that for each point, both other points lie in the same cone; see Figure 2. In fact, this example shows not only that this exact routing strategy fails; it shows that if we consider the edges to be directed (from the point that added them, to the closest point in its cone), the graph is not strongly connected. Our proof requires more global methods than previous proofs on θ -graphs.

Most proofs for a larger number of cones use induction on the distance between points or on the size of the empty triangle between a point and its closest point. In the θ_3 -graph however, both of these measures can increase when we follow an edge. Thus, applying induction on these distances seems a difficult task. An induction on the number of points similarly fails, as inserting a new point may remove edges that were present before, and it is not obvious that the endpoints of those edges are still connected in the new graph.

The θ_3 -graph is strongly related to the Yao-3-graph, where each point also connects to the closest point in each cone, but the distance measure is the standard Euclidean distance. This graph was shown to be connected by Damian and Kumbhar [7]. Their proof uses induction on a rhomboid distance-measure that was tailored specifically for the Yao-3-graph. Since the ‘closest’ point for the θ_3 -graph can be much further away than in the Yao-3-graph, this method of induction does not translate to the θ_3 -graph either. Conversely, we show that our proof extends to the Yao-3-graph, providing an alternative proof for its connectivity.

2 Properties of the θ_3 -graph

For $i \in \{0, 1, 2\}$, the edge connecting a point with its closest point in cone C_i is called an i -edge. Note that an edge can have one or two roles depending on the position of its endpoints. An example is depicted in Figure 2, where edge ab is both the 0-edge of a and the 1-edge of b .

Lemma 1 *For all $i \in \{0, 1, 2\}$, no two i -edges of the θ_3 -graph can cross.*

Proof. We consider only 0-edges of P ; the proof is analogous for 1- and 2-edges. For a contradiction, assume that there are two 0-edges that cross at a point s . Call these edges u_1v_1 and u_2v_2 , such that v_1 is in the 0-cone of u_1 and v_2 in the 0-cone of u_2 . Assume without loss of generality that the y -coordinate of v_1 is smaller than that of v_2 ; see Figure 3 for an illustration. Because s lies on segments u_1v_1 and u_2v_2 , s lies in the 0-cones of both u_1 and u_2 . Therefore, the 0-cone of s is contained in the intersection of the 0-cones of u_1 and u_2 . As v_1 lies in cone C_0 of s , point v_1 lies in cone C_0 of u_2 as well. Because we assumed that the y -coordinate of v_1 is less than that of v_2 , we conclude that v_1 is closer to u_2 than v_2 . Thus, the edge u_2v_2 is not a 0-edge yielding a contradiction. \square

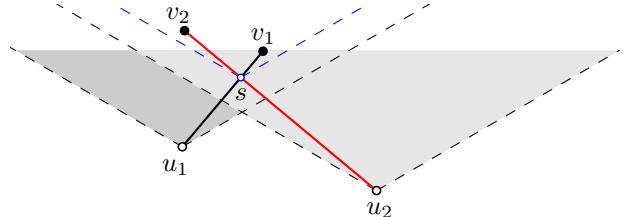


Figure 3: Two 0-edges u_1v_1 and u_2v_2 such that $v_1 \in C_0^{u_1}$ and $v_2 \in v_1 \in C_0^{u_2}$ cannot cross because the lowest point among v_1 and v_2 will be adjacent to both u_1 and u_2 .

We say that a cone is *empty* if it contains no point of P in its interior. A point having an empty i -cone is called an i -sink.

Given a point p of P , the i -path from p is defined recursively as follows: If the i -cone of p is empty, the i -path from p consists of the single point p . Otherwise, let q be the closest point to p in its i -cone. The i -path from p is defined as the union of edge pq with the i -path from q .

Lemma 2 *Every i -path of the θ_3 -graph is well-defined and has an i -sink as one of its endpoints.*

Proof. We consider only 0-paths; the proof is analogous for the other paths. A 0-path from a point p is well defined because the closest point in the 0-cone of p always lies above p . Therefore, the sequence of points

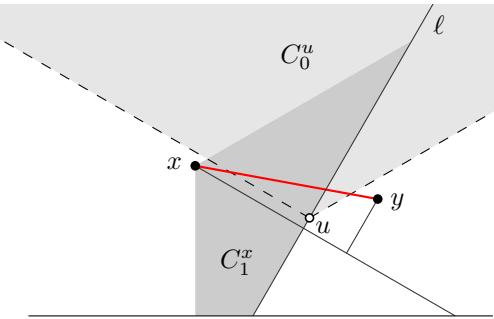


Figure 4: The last case in the proof of Lemma 3 where it is shown that empty i -cones cannot be crossed by edges of the θ_3 -graph.

in the 0-path from p is monotonically increasing in the y -coordinate. Because P is a finite set, the depth of the recursion is finite and must end at a point having an empty 0-cone. \square

Lemma 3 *If a cone of a point is empty, then no edge of the θ_3 -graph can cross this cone.*

Proof. We consider only 0-cones for this proof; analogous arguments hold for the other cones. Let u be a point of P with an empty 0-cone. We prove the lemma by contradiction, so assume that there exists an edge xy that crosses C_0^u . Since no edge between two points in the same cone can cross another cone, assume without loss of generality that $x \in C_2^u$ and $y \in C_1^u$.

Note that y cannot lie in C_0^x , since either C_0^x does not intersect C_1^u (if $u \notin C_0^x$) or the line segment between x and y does not intersect C_0^u (if $u \in C_0^x$). Therefore, y lies in C_1^x .

If $u \in C_0^x$, then C_1^x does not intersect C_0^u and hence, the line segment between x and y cannot intersect C_0^u either. Therefore, both u and y lie in C_1^x . Let ℓ be the perpendicular to the bisector of C_1^x that passes through u . For the edge xy to exist, the projection of y on the bisector of C_1^x must be closer to x than that of u , i.e., y must lie to the left of ℓ . However, all points lying to the left of ℓ are contained in $C_0^u \cup C_2^u$ yielding a contradiction as $y \in C_1^u$; see Figure 4 for an illustration of this case. \square

As a consequence of Lemmas 1 and 3, two sinks connected by an i -path partition the remaining points into two sets such that no i -path can connect a point in one set to a point in the other set, as any such path would cross either the i -path between the sinks, or the empty cone of one of the sinks. Such a construction is called an *i-barrier*; see Figure 5 for an illustration.

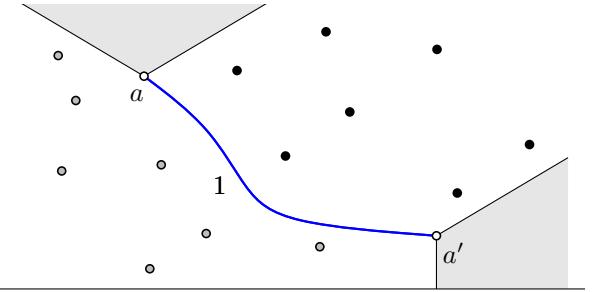


Figure 5: A 1-barrier, defined by the 1-path joining a with a' , splits the remaining points into two sets such that no two points in different sets can be joined by a 1-path.

3 Proving connectedness

In this section we prove that the θ_3 -graph of any given point set is connected. We start by proving that three given 0-sinks in a specific configuration are always connected. We then prove that if the θ_3 -graph has at least two disjoint connected components, then there exist three 0-sinks that are in this configuration and are not all in the same component.

Although the edges of the θ_3 -graph are not directed, by Lemma 2 we can think of an i -path as *oriented* towards the i -sink it reaches. An i -path oriented from point a to point b is denoted by $a \rightarrow b$. The following lemma is depicted in Figure 6.

Lemma 4 *Given three 0-sinks a , b , and c , such that (i) a lies to the left of b and b lies to the left of c , and (ii) the 1-path from a ends at a 1-sink a' , whose 0-path ends at c , then a , b , and c belong to the same connected component.*

Proof. Since there is a path from a to c via a' , a and c must be in the same component. We show that b belongs to this same connected component. The proof proceeds by induction on the number of 0-sinks to the right of c .

In the base case, there are no 0-sinks to the right of c . Consider the 1-sink b' at the end of the 1-path from b ; see Figure 6 (right). If b' and a' are the same point,

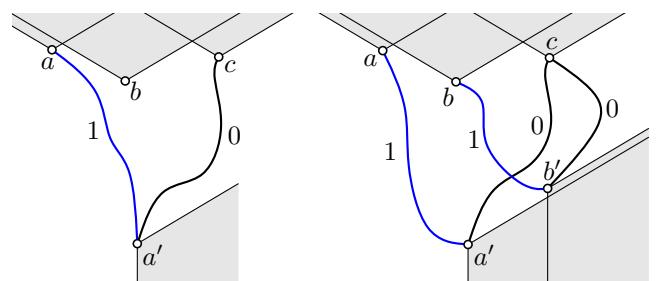


Figure 6: Left: The configuration of three 0-sinks described in Lemma 4. Right: The configuration in the base case of the induction where no 0-sink lies to the right of c .

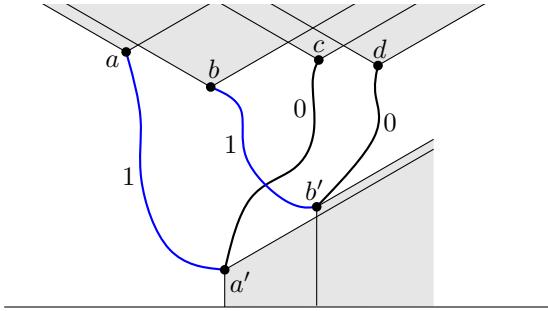


Figure 7: The configuration of the inductive step where the induction hypothesis can be applied on 0-sinks b, c and d .

then b is in the same connected component as a and we are done, so assume that this is not the case. Since the 1-path $a \rightarrow a'$ forms a 1-barrier, b' must lie to the right of a' . The 1-path $b \rightarrow b'$ also has to cross the 0-path $a' \rightarrow c$, as otherwise $a' \rightarrow c$ would cross the empty cone of b' , which is impossible by Lemma 3. Because the 0-path $a' \rightarrow c$ forms a 0-barrier, the 0-path from b' cannot end up to the left of c . Moreover, since there are no 0-sinks to the right of c , the 0-path from b' must end at c . Thus, there is a path connecting b and c , which proves the lemma in the base case.

For the inductive step, let k be the number of 0-sinks to the right of c and assume that the lemma holds for any triple of 0-sinks with fewer than k 0-sinks to their right. By the same argument as in the base case, we have a 1-path from b to a 1-sink b' that lies to the right of a' . Now consider the 0-sink d at the end of the 0-path from b' ; see Figure 7. Since the 0-path $a' \rightarrow c$ forms a 0-barrier, d cannot lie to the left of c . If d and c are the same point, we have a path connecting b and c as in the base case, so assume that this is not the case. Thus d lies to the right of c . Now b, c , and d form a triple of 0-sinks that satisfy criteria (i) and (ii). And since d is a 0-sink to the right of c , there are fewer than k 0-sinks to the right of d . Thus, by induction, we have that b is in the same connected component as c , which proves the lemma. \square

Theorem 5 *The θ_3 -graph is connected.*

Proof. Assume for sake of a contradiction that there exists a point set P whose θ_3 -graph G is not connected. From each point, we can follow its 0-path to end up at a 0-sink. Therefore, G must contain at least one 0-sink for each connected component. Let a be the leftmost 0-sink, and let A be the connected component of G that contains a . Now let b be the leftmost 0-sink that does not belong to A .

We use Lemma 4 to show that, in fact, b must belong to A as well. Before we can do this, we need to define two barriers. The first barrier is formed by the 2-path from b , ending at a 2-sink b' . Because a lies in C_2^b ,

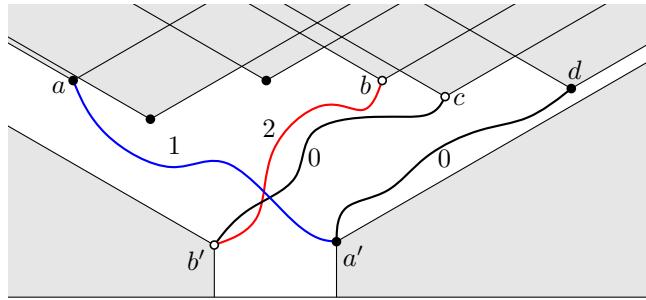


Figure 8: Two 0-sinks a and b are assumed to lie in different components such that both a and b are the leftmost 0-sinks in their component. The 1-path from a ends at a 1-sink a' whose 0-path ends at a 0-sink d lying to the right of b . The 0-sinks a, b and d jointly satisfy the criteria of Lemma 4.

point b does not have an empty 2-cone and hence, b' differs from b . The second barrier is formed by the 0-path from b' , which ends at a 0-sink c ; see Figure 8. Since b is the leftmost 0-sink that does not belong to A , either c and b are the same point, or c lies to the right of b .

Now consider the 1-sink a' at the end of the 1-path from a . This point has to lie to the right of both barriers $b \rightarrow b'$ and $b' \rightarrow c$, as otherwise these paths would cross the empty cone C_1 of a' , which is not allowed by Lemma 3. Because the path $a \rightarrow a'$ is a 1-path and the barriers in question consist of 0- and 2-edges, these crossings are possible. Now let d be the 0-sink at the end of the 0-path from a' . Since this path cannot cross the 0-barrier $b' \rightarrow c$, d cannot lie to the left of c .

Because d belongs to component A , if c and d are the same point, c belongs to component A . Otherwise, if c and d are distinct points, then a, b , and d jointly satisfy the criteria of Lemma 4, which gives us that b belongs to component A as well—a contradiction since b is the leftmost 0-sink that does not belong to A . This contradiction comes from our assumption that G is not connected. Therefore, the θ_3 -graph of any point set is connected. \square

4 The Yao-3-graph

The construction of the Yao-3-graph is very similar to that of the θ_3 -graph. The only difference is in the way distance is measured: the θ -graph uses the length of the projection onto the bisector, whereas the Yao-graph uses the Euclidean distance. Therefore, in every cone a point is connected to its closest Euclidean neighbor. We denote by $|pq|$ the Euclidean distance between two points p and q .

We show that like the θ_3 -graph, the Yao-3-graph is also connected. To this end, we re-introduce the three basic lemmas we had for the θ_3 -graph and show that the same properties hold for the Yao-3-graph.

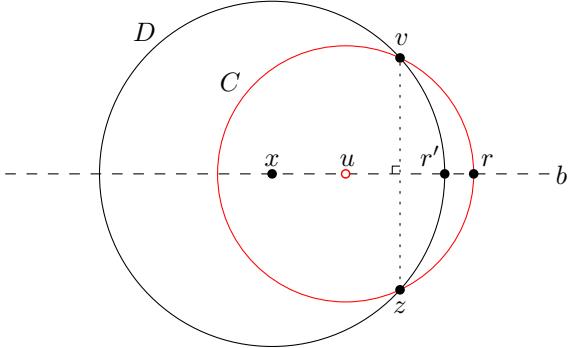


Figure 9: Point x lies to the left of point u and the arcs vr' and $r'z$ are enclosed by circle C centered at u , having radius $|uv|$.

We first prove a geometric auxiliary lemma depicted in Figure 9.

Lemma 6 *Given a non-vertical line b and a circle C centered at a point u on b , let v and z be two points on C such b bisects the segment vz . Let x be a point on b and let D be the circle centered at x having radius $|xv|$. If x lies to the left of u , then the right-side arc of D between v and z is enclosed by C ; otherwise, the left-side arc of D between v and z is enclosed by C .*

Proof. Assume that x lies to the left of u ; the proof of the other case is analogous. Let r and r' respectively be the right intersections of C and D with line b ; see Figure 9. Hence, arcs vr' and $r'z$ lie either entirely inside C or entirely outside C . Therefore, it suffices to show that r' is enclosed by C , i.e., $|ur'| \leq |ur|$. Since x lies to the left of u , we can rewrite $|ur'|$ as $|xr'| - |xu|$. Since $|xr'| = |xv|$ and $|ur| = |uv|$, we thus need to show that $|xv| \leq |xu| + |uv|$. This follows from the triangle inequality. \square

The proof of the following lemma is similar to that of Lemma 1.

Lemma 7 *For all $i \in \{0, 1, 2\}$, no two i -edges of the Yao-3-graph can cross.*

Proof. We look at the 0-edges. The cases for the other edges are analogous. Let uv be a 0-edge such that $v \in C_0^u$ and assume without loss of generality that v lies to the right of u . We prove the lemma by contradiction, so assume that some 0-edge xy crosses uv and let $y \in C_0^x$. Note that for xy to cross uv , C_0^x must contain some part of uv . Hence v lies in C_0^x .

Let k be the line through the right boundary of C_0^u and let l be the line through u such that the angle between l and the vertical line through u is $\pi/6$. We consider four cases, depending on the location of x with respect to u ; see Figure 11 (left): (a) $x \in C_0^u$ to the left

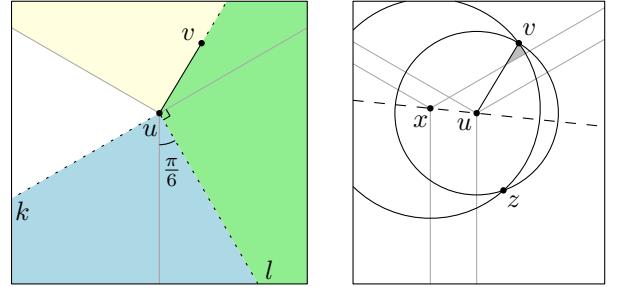


Figure 10: Left: The four cases. Right: The case when x lies in C_0^u and above k .

of the line uv , (b) $x \in C_2^u$ above k , (c) $x \in C_2^u$ below k or $x \in C_1^u$ below l , (d) $x \in C_1^u$ above l or $x \in C_0^u$ to the right of the line uv .

Case (a): $x \in C_0^u$ to the left of the line uv . Since v lies inside C_0^x and v lies to the right of u , x lies in the circle centered at u having radius $|uv|$. Thus, x lies closer to u than v , contradicting the existence of edge uv .

Case (b): $x \in C_2^u$ above k . We apply Lemma 6 as follows, see Figure 11 (right): Let C be the circle centered at u having radius $|uv|$. Let the line through u and x be bisector b , the bisector of v and z . Note that this implies that z lies outside C_0^u . Let D be the circle centered at x having radius $|xv|$. Since x lies to the left of u , Lemma 6 gives us that the right arc vz of circle D is enclosed by circle C . Since the area in which y must lie for xy to cross uv is bounded by the right boundary of C_0^x , edge uv , and the right arc vz of circle D , it is enclosed by C . Therefore, there does not exist a point $y \in C_0^x$ such that xy intersects uv .

Case (c): $x \in C_2^u$ below k or $x \in C_1^u$ below l . Since u lies in C_0^x , y needs to be closer to x than u for edge xy to exist. Hence it must lie inside the circle C centered at x having radius $|xu|$. Look at the lower half-plane defined by the line through u perpendicular to C and note that C is contained in this half-plane. However, the half-plane does not intersect C_0^u to the right of u and hence no point y inside the half-plane can be used to form an edge xy that crosses uv .

Case (d): $x \in C_1^u$ above l or $x \in C_0^u$ to the right of the line uv . We apply Lemma 6 as follows, see Figure 11 (right): Let C be the circle centered at u having radius $|uv|$. Let the line through u and x be bisector b . Note that z lies outside C_0^x . Let D be the circle centered at x having radius $|xv|$. Since x lies to the right of u , Lemma 6 gives us that the left arc vz of circle D is enclosed by circle C . Since the area in which y must lie for xy to cross uv is bounded by edge uv , the left arc vz of circle D , and either the left boundary of C_0^x (if $u \notin C_0^x$) or the line ux (if $u \in C_0^x$), it is enclosed by C . Therefore, there does not exist a point $y \in C_0^x$ such that xy intersects uv . \square

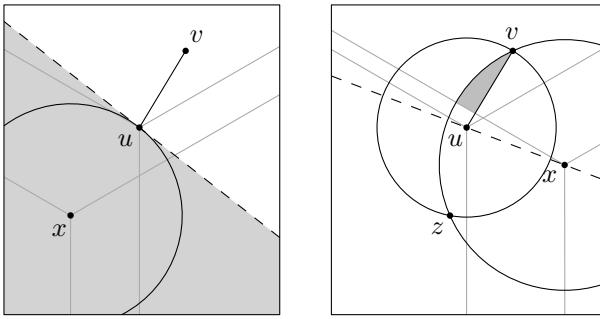


Figure 11: Left: The case when $x \in C_2^u$ below k or $x \in C_1^u$ below l . Right: The case when $x \in C_1^u$ above l or $x \in C_0^u$ to the right of the line uv .

Lemma 8 *Every i -path of the Yao-3-graph is well-defined and has an i -sink as one of its endpoints.*

Proof. The proof of this lemma is analogous to Lemma 2 for the θ_3 -graph. \square

Lemma 9 *If a cone of a point is empty, then no edge in the Yao-3-graph can cross this cone.*

Proof. We assume without loss of generality that C_0^u does not contain any points. We prove the lemma by contradiction, so assume that there exists an edge xy that crosses C_0^u . Since no edge between two points in the same cone can cross another cone, let $x \in C_2^u$ and $y \in C_1^u$.

Point y cannot lie in C_0^x , since either C_0^x does not intersect C_1^u (if $u \notin C_0^x$) or the line segment between x and y does not intersect C_0^u (if $u \in C_0^x$). Hence y must lie in C_1^x .

If $u \in C_0^x$, C_1^x does not intersect C_0^u and thus the line segment between x and y cannot intersect C_0^u either. Therefore both u and y lie in C_1^x . For the edge xy to exist, y must be closer to x than u , i.e., y must lie in the circle centered at x having radius $|xu|$. This circle is contained in the half-plane to the left of the line through u perpendicular to the circle.

If x lies on or above the horizontal line through u , the half-plane does not intersect C_1^u . If x lies below the horizontal line through u , the half-plane does not intersect C_1^u above u and thus xy would not cross C_0^u . Since y is enclosed by the circle, the circle is contained in the half-plane, and there is no point p in the half-plane such that $p \in C_1^u$ and px crosses C_0^u , xy cannot cross C_0^u either. \square

Using Lemmas 7, 8 and 9, the proof of Theorem 5 translates directly to the Yao-3-graph yielding the following result.

Theorem 10 *The Yao-3-graph is connected.*

Acknowledgments. This problem was introduced during the Fields Workshop on Discrete and Computational Geometry held at Carleton University in Ottawa, Canada.

References

- [1] L. Barba, P. Bose, J.-L. De Carufel, A. van Renssen, and S. Verdonschot. On the stretch factor of the Theta-4 graph. In *To appear in WADS*, 2013.
- [2] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *Graph-theoretic concepts in computer science*, volume 6410 of *Lecture Notes in Computer Science*, pages 266–278, 2010.
- [3] P. Bose, J.-L. De Carufel, P. Morin, A. van Renssen, and S. Verdonschot. Optimal bounds on theta-graphs: More is not always better. In *Proceedings of CCCG*, pages 305–310, 2012.
- [4] P. Bose, P. Morin, A. van Renssen, and S. Verdonschot. The θ_5 -graph is a spanner. *To appear in the proceedings of WG'13*, 2013.
- [5] P. Bose, A. van Renssen, and S. Verdonschot. On the spanning ratio of theta-graphs. In *Proceedings of WADS*, 2013.
- [6] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *STOC*, pages 56–65, 1987.
- [7] M. Damian and A. Kumbhar. Undirected connectivity of sparse yao graphs. In *FOMC*, pages 25–32, 2011.
- [8] N. M. El Molla. *Yao spanners for wireless ad hoc networks*. PhD thesis, Villanova University, 2009.
- [9] J. M. Keil. Approximating the complete Euclidean graph. In *SWAT*, volume 318 of *Lecture Notes in Computer Science*, pages 208–213, 1988.
- [10] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.
- [11] J. Ruppert and R. Seidel. Approximating the d -dimensional complete Euclidean graph. In *CCCG*, pages 207–210, 1991.

How to Cover Most of a Point Set with a V-Shape of Minimum Width

Boris Aronov*
aronov@poly.edu

John Iacono†
jiacono@poly.edu

Özgür Özkan
ozgurozkan@gmail.com

Mark Yagnatinsky‡
myag@cis.poly.edu

Polytechnic Institute of NYU, Brooklyn, New York

Abstract

A V-shape is an infinite polygonal region bounded by two pairs of parallel rays emanating from two vertices (see Figure 1). We describe a randomized algorithm that, given n points and an integer $k \geq 0$, finds the minimum-width V-shape enclosing all but k of the points with probability $1 - 1/n^c$ for any $c > 0$, with expected running time $O(cn^2(k+1)^4 \log n (\log n \log \log n + k))$.

1 Introduction

Motivation. The motivation for this problem comes from curve reconstruction: given a set of points sampled from a curve in the plane, find a shape approximating the original curve. It has been suggested in [AD13] that in an area where the curve makes a sharp turn, it makes sense to model the curve by a *V-shape*. The authors remark that it would be natural to investigate a variant that can handle a small number of outliers, to accommodate a few bad data points. We investigate that variant here. The problem is an instance of a large class of problems known as *geometric optimization* or *fitting* questions, (see [AS98] for a survey).

Previous work. In [AD13], the authors develop an algorithm for covering a point set in general position¹ with a V-shape of minimum width (allowing no outliers) that runs in $O(n^2 \log n)$ time and uses quadratic space. They also find a constant-factor approximation algorithm with running time $O(n \log n)$, and a $(1 + \varepsilon)$ -approximation algorithm with a running time of $O((n/\varepsilon) \log n + n/(\varepsilon^{3/2}) \log^2(1/\varepsilon))$, which is $O(n \log n)$ for a constant $\varepsilon > 0$.

*Research supported by NSF Grants CCF-08-30691, CCF-11-17336, and CCF-12-18791, and by NSA MSP Grant H98230-10-0210.

†Research supported by NSF Grant CCF-1018370.

‡Research supported by NSF Grant CCF-11-17336, NSA MSP Grant H98230-10-1-0210, and GAANN Grant P200A090157 from the US Department of Education.

¹We use the same general position assumptions as [AD13]: no vertical line goes through two points, no three points are collinear, and no lines defined by pairs of points are parallel.

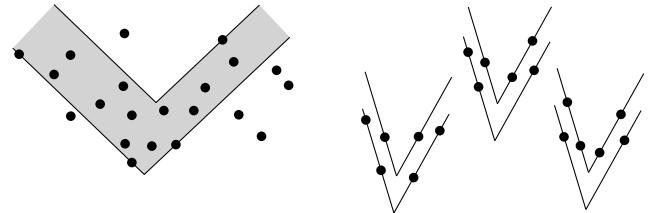


Figure 1: Left to right: a V-shape with six outliers, and both-outer, inner-outer, and both-inner V-shapes.

Our Result. Given a set P of n points in the plane and an integer $k \geq 0$, we show how to find the minimum-width V-shape enclosing all but k of the points.

Definitions and notation. A V-shape is an infinite polygonal region bounded by two pairs of parallel rays emanating from two vertices (see Figure 1). The rays on the region's convex hull are the *outer rays*. The others are the *inner rays*.

The line segment connecting the vertices of the outer and inner rays separates the V-shape V into its *left arm* and *right arm*. The *width* of an arm is the distance between its two delimiting rays. The *width* of V is the width of its wider arm. An *outlier* of V is a point of P not contained in V . Each arm has an associated *strip*, defined by the pair of directed parallel lines going through its boundary. The left and right strips together uniquely determine a V-shape. This is in fact how our algorithm works: by trying to find a pair of strips determining the thinnest V-shape.

Given a point set P , a k -edge (see Figure 2) of P is a directed edge between two points in the set such that exactly k points of P lie to the left of the directed line through the edge (so in general position there are $n - k - 2$ points to the right). For example, a 0-edge is a directed edge on the convex hull. A k -edge is also said to be *an edge at level k* . Let $L(e, P)$ denote the level of edge e in point set P . The set $H(k, P)$ of edges at level k or less are known as the *at-most- k -edges*, or more concisely, the $(\leq k)$ -edges.

It will also be useful to talk about levels in a line arrangement \mathcal{A} (see Figure 3). We use the following definition: an edge of \mathcal{A} is on the k -level if there are

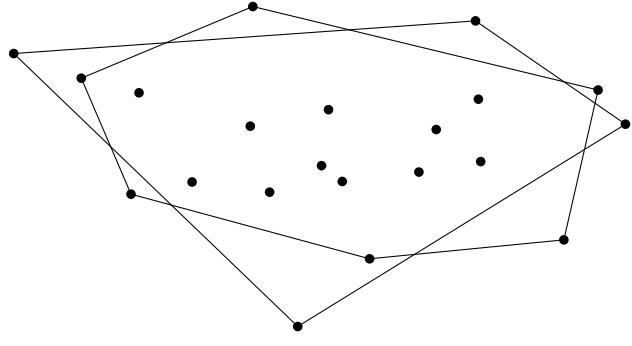


Figure 2: The edges at level 1 of a point set.

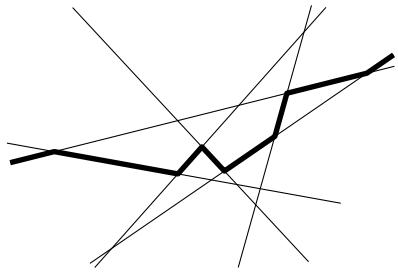


Figure 3: The 2-level of a line arrangement with six lines.

exactly k lines of \mathcal{A} strictly above it.

2 The algorithm

2.1 Overview

We need only consider locally optimal V-shapes. In [AD13], it was shown that there are three types of locally optimal configurations, which they called both-outer, inner-outer, and both-inner (see Figure 1). A *both-outer* V-shape is a V-shape where both outer rays have two points on them (and the inner rays have one). A *both-inner* V-shape is a V-shape where both inner rays have two points on them (and the outer rays have one). An *inner-outer* V-shape is a V-shape where one of the outer rays and one of the inner rays has two points on it and the other two rays have one point each. (Note that even for point sets in general position, one of the arms may have its two rays coincide, and thus the V-shape will have only five points on its boundary instead of six.) A V-shape with k outliers is called a k -outlier V-shape of the point set. Our algorithm works by finding the minimum-width k -outlier V-shape of each type, and returning the one that has the smallest width of all three.

Our approach for the both-outer case and the inner-outer case was inspired by the approach of [AD13] for the inner-outer case, except we use a binary search for one step where they use total enumeration. When there are zero outliers, our algorithm for the both-outer and

inner-outer cases would be easier to implement than theirs, at the cost of a logarithmic factor in the running time. However, most of the complexity of their solution was in the both-outer case, and we use their both-outer algorithm as a black box in our both-outer algorithm, by running it on random subsets of the point set.

We handle both-outer V-shapes and inner-outer V-shapes in almost the same way (see Figure 4). We begin by enumerating the $(\leq k)$ -edges of the point set. Each such j -edge e is considered in turn as a candidate for one of the outer rays to go through, with $j \leq k$ outliers already accounted for. For a fixed e , we do a binary search among remaining points of P , ordered by perpendicular distance from e ; this distance is the width of the first candidate strip. For each point of the search we find the second strip that has the smallest possible width and still covers the remaining points, except for the outliers. If the second strip is wider than the first, the binary search moves farther out from e so that the second strip has fewer points, otherwise it moves closer. To find the second strip, we again enumerate the edges at levels 0 through k of the remaining points. The precise definition of “remaining” here is the key difference between the both-outer and the inner-outer algorithm; see detailed discussion below. By now we have chosen three rays, and have no freedom for the fourth: it is dictated by how many more outliers we need. The running time is $O(n^2(k+1)^2 \log n)$ (see Lemma 3 for proof).

To find the minimum-width both-inner k -outlier V-shape, we use a randomized algorithm that takes random samples of the given point set. For each sample, it enumerates *all* both-inner 0-outlier V-shapes using the algorithm from [AD13]. We show that with enough samples, the minimum-width both-inner k -outlier V-shape will be one of the V-shapes enumerated with probability at least $1 - 1/n^c$ for any real number $c > 0$ (given as an input parameter). The V-shapes we enumerate might have more than k outliers, so we use a range searching data structure from [CY84] to detect and discard such V-shapes. The running time of the both-inner case is $O(cn^2(k+1)^4 \log n(\log n \log \log n + k))$, which dominates the running time of the other two cases.

Theorem 1 *There is a randomized algorithm that, given n points and an integer $k \geq 0$ denoting the desired number of outliers, finds the minimum-width k -outlier V-shape for the points with probability $1 - 1/n^c$ for any $c > 0$, requiring $O(n^2)$ space with expected running time $O(cn^2(k+1)^4 \log n(\log n \log \log n + k))$.*

Proof. We find the thinnest k -outlier V-shape of each of the three types separately and return the thinnest. The both-inner algorithm dominates the running time. The running time and correctness of the algorithms handling the three cases is established in Lemmas 2, 3, and 5. \square

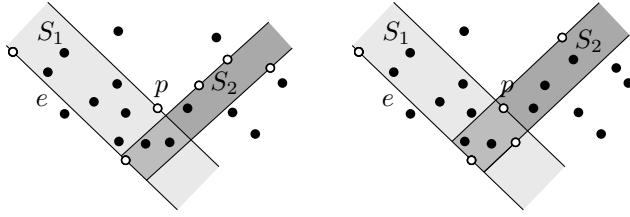


Figure 4: Snapshot of inner-outer algorithm (left) and both-outer algorithm (right).

2.2 Both-Outer and Inner-Outer

The following two algorithms find the thinnest inner-outer k -outlier V-shape and the thinnest both-outer k -outlier V-shape. The algorithms have the same structure: for each candidate first strip S_1 , find the thinnest possible second strip S_2 :

Algorithms 1 and 2. Part I: Finding S_1 .

Input: integer k , point set P such that $|P| = n$.

For each edge $e \in H(k, P)$:

P' = points to the right of e , and the two points on e .
Sort P' by distance from the line through e .

//Perform binary search on P' .

For each point p of the binary search:

S_1 = points contained in strip defined by e and p .

Find thinnest strip S_2 . //see part 2 of algorithm

The binary search is guided by which strip is thicker:

If S_1 is thicker, move p closer to e , else further.

It remains to explain how to find the thinnest S_2 and this depends on whether the k -outlier V-shape we seek is an inner-outer V-shape or a both-outer V-shape. First we define a function *find-line* (described in Lemma 4) that takes a directed edge e , an integer i , and point set P , and finds the $(i + 1)$ st furthest line from e going through a point in P right of e .

The two cases are similar, so the steps that differ are marked with an asterisk. It may help to refer to Figure 4. To find a both-outer k -outlier V-shape, we use Algorithm 1, and to find an inner-outer k -outlier V-shape, we use Algorithm 2.

Algorithm 1. Part II: Finding S_2 of a both-outer V-shape.

$i = L(e, P)$

For each edge $f \in H(k - i, P')$:

$j = k - i - L(f, P')$. * //number of outliers still needed

Let $\ell = \text{find-line}(f, j, P' - S_1)$. *// ℓ may not exist

Let S_2 = the strip formed by f and ℓ .

Record the thinnest S_2 found so far.

If ℓ from Algorithm 1 does not exist, because $P' - S_1$ has less than j points, then the strip determined by S_1 is too wide, and we can proceed to the next f .

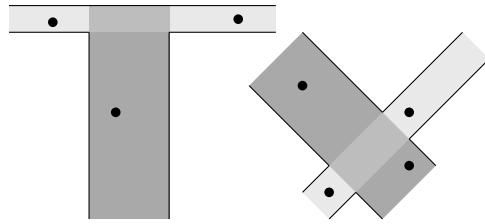


Figure 5: Invalid V-shapes that looks like a T or an X.

Algorithm 2. Part II: Finding S_2 of an inner-outer V-shape.

$i = L(e, P)$

For each edge $f \in H(k - i, P' - S_1)$: *

$j = k - i - L(f, P' - S_1)$. * //# of outliers still needed

Let $\ell = \text{find-line}(f, j, P')$. *

Let S_2 = the strip formed by f and ℓ .

Record the thinnest S_2 found so far.

Lemma 2 *The above algorithms are correct.*

Proof. There are three ways these algorithms can fail. It can fail to find a valid V-shape, the V-shape it finds can have the wrong number of outliers, or it can overlook the thinnest V-shape with k outliers. How do we know that all the V-shapes we just enumerated with the above algorithms are valid? Two arbitrary strips may form a shape that looks like an X or a T instead of a V (see Figure 5). More formally, we want to avoid S_1 having points on *both* sides of S_2 . The points covered by S_1 might indeed be split by S_2 , but this can only happen when the points that were split off were among the k outliers. This is because only the outer ray of S_2 can split off points from S_1 , and it only splits off points near the convex hull: the outliers.

The algorithms never create more than k outliers, because they keep track of how many are needed and at each step never create more than that. Do they ever create less than k ? This can only happen if the algorithm counts some outlier more than once. The algorithms choose outliers three times: first when they choose e , then when choosing f , and finally when choosing ℓ . The outliers caused by e (that is, the i points to its left) are never double-counted, because they are invisible to the rest of the algorithm, which works with P' instead of P . The outliers created by f and those created by ℓ are on opposite sides of f , so they can not be counted twice either.

Lastly, can the thinnest both-outer or inner-outer V-shape be overlooked? For both-outer and inner-outer V-shape, there is at least one outer ray defined by two points, and we consider all edges e that could possibly define it. For a fixed choice of e and p , we look at all

feasible choices of f . For a fixed choice of e , p and f , we have no freedom in choosing ℓ , so no wrong choice is possible. The only place where we do not look at all possibilities is in choosing p , where we do binary search. This is perfectly safe, because if, say, S_1 is thinner than S_2 , moving p closer to e forces S_2 to cover more points, which can not make it any thinner. \square

Lemma 3 *The running time of the above algorithms is $O(n^2(k+1)^2 \log^2 n)$.*

Proof. The algorithms are structured as a triply nested loop, so it suffices to count the number of iterations of each loop. It is well known that the set of $(\leq k)$ -edges has size $O((k+1)n)$ [GP84, AG86], so the loops for e and f both iterate at most that many times. The binary search for p iterates $O(\log n)$ times. We can enumerate the j -edges, for all $j \leq k$, in sorted order along the j -level, in $O((k+1)n \log n)$ time using the algorithm in [EW86, pages 272–278]². By Lemma 4 we can implement find-line to run in $O(\log n)$ time. The claimed running time follows. \square

Lemma 4 *After $O((k+1)n \log n)$ preprocessing, we can find the $(i+1)$ st furthest point from a directed line e among points to the right of e , where $0 \leq i \leq k$, in $O(\log n)$ time.*

Proof. Finding the desired point is equivalent to finding the line ℓ parallel to e which goes through the point in P such that there are i points in P right of ℓ . (This is the line that find-line returns.) To do this, we go to the dual and let \mathcal{A} be the line arrangement induced by P^* , where ℓ dualizes to a point ℓ^* . The requirement in the primal that there be k points right of ℓ means that ℓ^* must lie on an edge in the k -level or the $(|P|-k-1)$ -level of \mathcal{A} , and the fact that ℓ must be right of e eliminates one of these two possibilities. Again using [EW86], we can compute the i -levels, and the $(|P|-1-i)$ -levels, for all $i \leq k$, in sorted order by x -coordinate, in time $O((k+1)n \log n)$. Since we know the x -coordinate of ℓ^* (it is given by the slope of e in the primal), we can do binary search on the i -level to identify the two vertices that ℓ^* lies between. These two vertices lie on a line of P^* , which corresponds to a point of P in the primal. This is the desired point. \square

2.3 Both-Inner

The following algorithm finds the thinnest both-inner k -outlier V-shape with high probability.

Lemma 5 *Algorithm 3 finds the thinnest both-inner k -outlier V-shape with probability at least $1 - 1/n^c$ for any*

²The algorithm of [EW86] depends on a data structure for dynamic convex hull. At the time, the best available such structure was that of [OVL81, pages 169–181]. Using the one described in [J02] instead gives the claimed running time.

Algorithm 3. Finds a min-width both-inner k -outlier V-shape for P with high probability.

Input: integer k , point set P , real number $c > 0$

Let $n = |P|$, and let $K = k + 1$

Repeat $K^6 ce \ln n$ times:

 Initialize R to the empty set

 For each point in P , add it to R with probability $1/K$

$W = \text{Find-empty-V-shapes}(R) // [\text{AD13, pp 303–304}]$

 Remove V-shapes with more than k outliers from W

Return the thinnest V-shape seen.

$c > 0$, in $O(cn^2(k+1)^4 \log n (\log n \log \log n + k))$ expected time and $O(n^2)$ space.

Proof. By [AD13], the above algorithm always produces the right answer if $k = 0$, albeit with needless redundant sampling of the entire point set, so we restrict our attention to the case where $k > 0$. Denote the thinnest both-inner k -outlier V-shape by V . Clearly, V is defined by (at most) six points of P . Consider a subset R of P , which contains the six points defining V but does not contain the k outliers. V is a valid both-inner 0-outlier V-shape for R , though perhaps not the thinnest one. The algorithm simply samples P over and over, in the hopes of eventually picking such a subset R . For each sample R , it enumerates all both-inner 0-outlier V-shapes using the algorithm from [AD13], and checks whether they end up having at most k outliers in P . Note that if all the V-shapes we consider end up resulting in more than k outliers, our algorithm fails to find any valid V-shape. However, we show that this is very unlikely: the probability that the algorithm fails to find the optimal V-shape is less than $1/n^c$, where c is the given positive constant.

Each point in P is independently chosen to be part of R with probability $1/K$. Thus, R has expected size n/K . Now, what is the probability that the optimal thinnest both-inner k -outlier V-shape is one of the valid both-inner 0-outlier V-shapes for R ? The probability of having the required six defining points is $1/K^6$, and the probability of avoiding the k outliers is $(1 - 1/K)^k = (1 - 1/(k+1))^k > 1/e$, since $(1 - 1/(k+1))^k$ converges to $1/e$ from above. So, the probability of our random sample containing the six points we need and not containing the k points we should avoid is at least $p = 1/eK^6$. If we call this the probability of success, then the probability of failure is at most $1 - p$. If instead of taking just one such random sample, we take $m = K^6$ samples, the probability of them all failing is at most $(1 - p)^m$. Now using the fact that, for all x , $1 - x < e^{-x}$, we conclude that the probability q of all m samples failing to contain the optimum both-inner V-shape is $(1 - p)^m < e^{-pm}$. Since $pm = (1/eK^6)(K^6) = 1/e$, we have $q < e^{-pm} = e^{-1/e}$. If we increase the number of samples from m to $mce \ln n$, then the probability of failure q reduces to at

most

$$e^{-pmce \ln n} = e^{-(ce \ln n)/e} = (e^{\ln n})^{-c} = 1/n^c,$$

which concludes the high-level description of the algorithm, and the proof that its probability of failure is at most $1/n^c$.

The crucial operation in the above algorithm is to check that the V-shapes returned by the algorithm from [AD13] do not have too many outliers. This can be done using range searching with wedges, which is a special case of simplex range searching, for which there are a variety of data structures with various space/time trade-offs. (A wedge is simply the convex region bounded by two rays with a common vertex.) We use a data structure that takes $O(n^2)$ space and gives $O(\log n \log \log n + k)$ query time [CY84, pages 41–45]. The time taken by the algorithm from [AD13] to enumerate all both-inner 0-outlier V-shapes of a point set with $O(n/k)$ points is $O((n^2/k^2) \log n)$. The subset may have as many as $O(n^2/k^2)$ V-shapes, each of which take $O(\log n \log \log n + k)$ time to check to make sure the number of outliers is not too high, for a total time of $O((n^2/k^2)(\log n \log \log n + k))$ per random sample.

We have glossed over a statistical subtlety here. If the expected value of a random variable X is $E[X]$, then in general $E[X^2]$ may not be $O(E[X]^2)$, or indeed, it might not even be finite. In this case, how do we know, just because the expected size of R is $O(n/k)$, that the expected number of V-shapes is $O(n^2/k^2)$? What is true for all random variables X from distributions with finite mean and variance is that $E[X^2] = E[X]^2 + \text{Variance}[X]$. The size of R follows the binomial distribution with mean n/K and variance $n(1/K)(1 - 1/K)$, so we have

$$E(|R|^2) = n^2/K^2 + n(1/K)(1 - 1/K) < n^2/K^2 + n/K,$$

which is $O(n^2/k^2)$.

Since we are taking $O(ck^6 \log n)$ random samples, finding the best both-inner k -outlier V-shape takes time $O(cn^2(k+1)^4 \log n (\log n \log \log n + k))$. \square

Remark. We have calculated how many samples we need in order to find a particular both-inner k -outlier V-shape with high probability (specifically, the thinnest one). A natural question to ask is how many samples we would need to find *all* both-inner k -outlier V-shapes.

If the probability of failing to find an arbitrary both-inner V-shape is q , then the probability of there being at least one both-inner V-shape we fail to find is at most q times the number of both-inner V-shapes present in the point set. Clearly, regardless of the value of k , this number is at most n^6 , and we already computed $q < 1/n^c$. By choosing $c > 7$, we have $q < 1/n^7$, and thus our probability of failure is less than $1/n$.

Acknowledgments

We would like to thank Sariel Har-Peled for the randomized algorithm, and Muriel Dulieu for participating in discussions of a simpler version of the problem.

References

- [AD13] B. Aronov and M. Dulieu, How to cover a point set with a V-shape of minimum width, *Comput. Geom.: Theory Appl.*, 46(3) (2013) 298–309.
- [AG86] N. Alon and E. Györi, The number of small semispaces of a finite set of points in the plane, *J. Combinatorial Theory, Ser. A*, 41(1) (1986) 154–157.
- [AS98] P.K. Agarwal and M. Sharir, Efficient algorithms for geometric optimization, *ACM Computing Surveys*, 30(4) (1998) 412–458.
- [CY84] R. Cole and C.K. Yap, Geometric retrieval problems, *Information and Control*, 63(1–2) (1984) 39–57.
- [EW86] H. Edelsbrunner and E. Welzl, Constructing belts in two-dimensional arrangements with applications, *SIAM J. Computing*, 15(1) (1986) 271–284.
- [GP84] J.E. Goodman and R. Pollack, On the number of k -subsets of a set of n points in the plane, *J. Combinatorial Theory, Ser. A*, 36(1) (1984) 101–104.
- [J02] R. Jacob, *Dynamic Planar Convex Hull*, PhD thesis, May 2002, University of Aarhus, Denmark, <http://brics.dk/DS/02/3>.
- [OvL81] M.H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Computer System Sciences*, 23(2) (1981) 166–204.

Computing Covers of Plane Forests*

Luis Barba^{†‡}Alexis Beingessner[†]Prosenjit Bose[†]Michiel Smid[†]

Abstract

Let ϕ be a function that maps any non-empty subset A of \mathbb{R}^2 to a non-empty subset $\phi(A)$ of \mathbb{R}^2 . A ϕ -cover of a set $T = \{T_1, T_2, \dots, T_m\}$ of pairwise non-crossing trees in the plane is a set of pairwise disjoint connected regions such that

1. each tree T_i is contained in some region of the cover,
2. each region of the cover is either
 - (a) $\phi(T_i)$ for some i , or
 - (b) $\phi(A \cup B)$, where A and B are constructed by either 2a or 2b, and $A \cap B \neq \emptyset$.

We present two properties for the function ϕ that make the ϕ -cover well-defined. Examples for such functions ϕ are the convex hull and the axis-aligned bounding box. For both of these functions ϕ , we show that the ϕ -cover can be computed in $O(n \log^2 n)$ time, where n is the total number of vertices of the trees in T .

1 Introduction

Let a *geometric tree* be a plane straight-line embedding of a tree in \mathbb{R}^2 . Consider a set $T = \{T_1, T_2, \dots, T_m\}$ of m pairwise non-crossing geometric trees with a total of n vertices in general position. The *coverage* of these trees is the set of all points p in \mathbb{R}^2 such that every line through p intersects at least one of the trees. Beingessner and Smid [1] showed that the coverage can be computed in $O(m^2 n^2)$ time. They also presented an example of $m = n/2$ pairwise non-crossing geometric trees (each one being a line segment) whose coverage has size $\Omega(n^4)$. Thus, the worst-case complexity of computing the coverage is $\Theta(n^4)$.

Since the worst-case inputs are rather artificial, we consider the following heuristic for reducing the running time. Let Conv denote the convex hull. We observe that the coverage of the trees in T is equal to the coverage of their convex hulls. Moreover, if two convex hulls $\text{Conv}(T_i)$ and $\text{Conv}(T_j)$ overlap, then we can replace them by the convex hull of their union without

changing the coverage. By repeating this process, we obtain a collection of pairwise disjoint convex polygons whose coverage is equal to the coverage of the input trees. Ideally, the number of these convex polygons and their total number of vertices are much less than m and n , respectively. If this is the case, then running the algorithm of [1] on the convex polygons gives the coverage of the input trees in a time that is much less than $\Theta(n^4)$ time, provided that we are able to quickly compute the collection of pairwise disjoint convex polygons. In this paper, we show that this is possible, by providing an $O(n \log^2 n)$ -time algorithm.

We now formally state the above process.

1. Let $\mathcal{C} = \{\text{Conv}(T_i) \mid 1 \leq i \leq m\}$.
2. While the elements of \mathcal{C} are not pairwise disjoint:
 - (a) Take two arbitrary elements C and C' in \mathcal{C} for which $C \neq C'$ and $C \cap C' \neq \emptyset$.
 - (b) Let $C'' = \text{Conv}(C \cup C')$.
 - (c) Set $\mathcal{C} = (\mathcal{C} \setminus \{C, C'\}) \cup \{C''\}$.
3. Return the set \mathcal{C} .

The output \mathcal{C} is a collection of pairwise disjoint convex polygons, which we refer to as the *hull-cover* of T . See Figure 1 for two examples. Since in Step 2(b), the two elements C and C' are chosen *arbitrarily* (as long as they are distinct and overlap), the reader may object to the use of the word “the” in front of “hull-cover”. In Section 2 we justify the use of this word by proving that, no matter what choices are made in Step 2(b), the output \mathcal{C} is always the same.

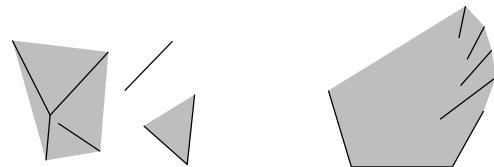


Figure 1: Two examples of hull-covers. Note that the hull-cover on the right demonstrates what is in some sense the worst case for the number of times intersections will need to be re-evaluated.

*The authors were supported by NSERC. A.B. was supported by Carleton University’s I-CUREUS program.

[†]School of Computer Science, Carleton University, Ottawa, Canada.

[‡]Boursier FRIA du FNRS, Département d’Informatique, Université Libre de Bruxelles

2 ϕ -Covers

Consider a function ϕ that maps any non-empty subset A of \mathbb{R}^2 to a non-empty subset $\phi(A)$ of \mathbb{R}^2 . We assume that this function satisfies the following properties:

Property 1: For any non-empty subset A of \mathbb{R}^2 ,

$$A \subseteq \phi(A).$$

Property 2: For any two non-empty subsets A and B of \mathbb{R}^2 ,

$$\text{if } A \subseteq \phi(B), \text{ then } \phi(A) \subseteq \phi(B).$$

Both the convex hull and axis-aligned bounding box functions satisfy these properties. However, the minimum enclosing circle function does not satisfy Property 2.

We rewrite the algorithm described in Section 1 using the function ϕ instead of *Conv*. We also use a forest \mathcal{F} of binary trees to keep track of the history of the process; each node u in this forest stores a value $\phi(u)$. The forest helps us to prove that the ϕ -cover is well-defined.

1. For each i with $1 \leq i \leq m$, let \mathcal{T}_i be the tree consisting of the single node r_i , whose value $\phi(r_i)$ is equal to $\phi(T_i)$.
2. Initialize the forest $\mathcal{F} = \{\mathcal{T}_i \mid 1 \leq i \leq m\}$.
3. Let $\mathcal{C} = \{\phi(r_i) \mid 1 \leq i \leq m\}$.
4. While the elements of \mathcal{C} are not pairwise disjoint:
 - (a) Take two arbitrary roots r and r' in the forest \mathcal{F} for which $r \neq r'$ and $\phi(r) \cap \phi(r') \neq \emptyset$.
 - (b) Let \mathcal{T} and \mathcal{T}' be the trees in \mathcal{F} whose roots are r and r' , respectively.
 - (c) Let r'' be a new node and set its value $\phi(r'')$ to $\phi(\phi(r) \cup \phi(r'))$.
 - (d) Create a new tree \mathcal{T}'' whose root is r'' and make \mathcal{T} and \mathcal{T}' the two children of r'' .
 - (e) Set $\mathcal{F} = (\mathcal{F} \setminus \{\mathcal{T}, \mathcal{T}'\}) \cup \{\mathcal{T}''\}$.
 - (f) Set $\mathcal{C} = (\mathcal{C} \setminus \{\phi(r), \phi(r')\}) \cup \{\phi(r'')\}$.
5. Return the forest \mathcal{F} and the set \mathcal{C} .

We refer to the output set \mathcal{C} as the ϕ -cover of T . In Theorem 2 below, we prove that the ϕ -cover is well-defined. Before we prove this theorem, we present a third property of the function ϕ :

Property 3: For any two non-empty subsets A and B of \mathbb{R}^2 ,

$$\phi(A) \subseteq \phi(\phi(A) \cup \phi(B)).$$

Note that this property follows trivially from Property 1, because

$$\phi(A) \subseteq \phi(A) \cup \phi(B) \subseteq \phi(\phi(A) \cup \phi(B)).$$

Lemma 1 *Let \mathcal{C} and \mathcal{C}' be two ϕ -covers with corresponding forests \mathcal{F} and \mathcal{F}' , respectively. For each node u in \mathcal{F} , there exists a root r' in \mathcal{F}' such that $\phi(u) \subseteq \phi(r')$.*

Proof. We prove the lemma by induction on the height of the subtree rooted at u . First assume that u is a leaf in \mathcal{F} . Let i be the index such that $\phi(u) = \phi(T_i)$, let u' be the leaf in \mathcal{F}' for which $\phi(u') = \phi(T_i)$, let \mathcal{T}' be the tree in \mathcal{F}' that has u' as a leaf, and let r' be the root of \mathcal{T}' . We prove that $\phi(u) \subseteq \phi(r')$.

Let $u'_1 = u', u'_2, \dots, u'_k = r'$ be the nodes in \mathcal{T}' on the path from u' to r' . For each i with $1 \leq i < k$, let v'_i be the sibling of u'_i . Since

$$\phi(u'_{i+1}) = \phi(\phi(u'_i) \cup \phi(v'_i)),$$

it follows from Property 3 that $\phi(u'_i) \subseteq \phi(u'_{i+1})$. From this, it follows that

$$\phi(u) = \phi(u') = \phi(u'_1) \subseteq \phi(u'_2) \subseteq \dots \subseteq \phi(u'_k) = \phi(r').$$

Now assume that u is not a leaf. Let v and w be the children of u . Observe that $\phi(v) \cap \phi(w) \neq \emptyset$. By induction, there exist roots r' and r'' in \mathcal{F}' such that $\phi(v) \subseteq \phi(r')$ and $\phi(w) \subseteq \phi(r'')$. Since $\phi(r') \cap \phi(r'') \neq \emptyset$, we must have $r' = r''$. Thus, since $\phi(v) \cup \phi(w) \subseteq \phi(r')$, Property 2 implies that

$$\phi(u) = \phi(\phi(v) \cup \phi(w)) \subseteq \phi(r').$$

□

Theorem 2 *The ϕ -cover is well-defined.*

Proof. Let \mathcal{C} and \mathcal{C}' be two ϕ -covers with corresponding forests \mathcal{F} and \mathcal{F}' , respectively. We have to prove that $\mathcal{C} = \mathcal{C}'$. Observe that

$$\mathcal{C} = \{\phi(r) \mid r \text{ is a root in } \mathcal{F}\}$$

and

$$\mathcal{C}' = \{\phi(r') \mid r' \text{ is a root in } \mathcal{F}'\}.$$

Let r be a root in \mathcal{F} . By Lemma 1, there exists a root r' in \mathcal{F}' such that $\phi(r) \subseteq \phi(r')$. Again by Lemma 1, applied with the roles of \mathcal{F} and \mathcal{F}' interchanged, there exists a root r'' in \mathcal{F} such that $\phi(r') \subseteq \phi(r'')$. Thus, we have

$$\phi(r) \subseteq \phi(r') \subseteq \phi(r'').$$

Since $\phi(r) \cap \phi(r'') \neq \emptyset$, we must have $r = r''$. Therefore, $\phi(r) = \phi(r')$. We conclude that $\mathcal{C} \subseteq \mathcal{C}'$. By a symmetric argument, we can show that $\mathcal{C}' \subseteq \mathcal{C}$. □

Thus, the ϕ -cover is well-defined for both the convex hull and the axis-aligned bounding box. If ϕ is the minimum enclosing circle function, then, in addition to not satisfying Property 2, the ϕ -cover is not well-defined: In Figure 2, an example is given for which the order in which merges are performed can result in different outputs.

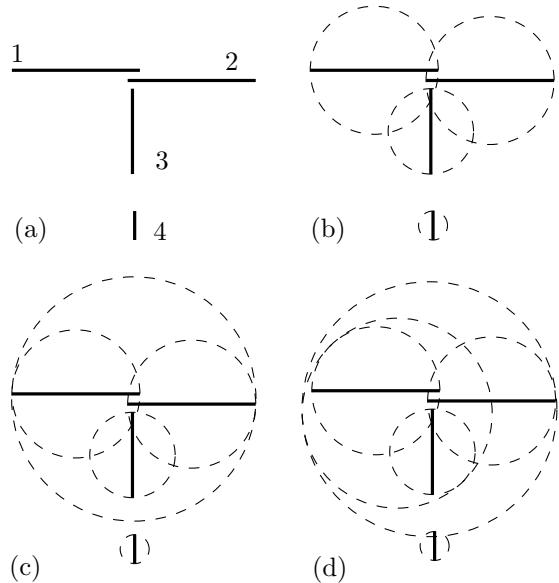


Figure 2: (a) The input forest with trees numbered; (b) The minimum enclosing circle of each tree; (c) Merging 1 and 2 first results in no intersection with 4; (d) Merging 1 and 3 first results in an intersection with 4.

3 Computing the Hull-Cover

In this section, we take for ϕ the convex hull function and show that the ϕ -cover can be computed in $O(n \log^2 n)$ time.

3.1 Weakly Disjoint Polygons

Finding the convex hull of two convex polygons can be a relatively expensive operation due to the fact that their boundaries can cross in $\Omega(n)$ different places. For example, consider a regular n -gon being merged with a copy of itself rotated ϵ degrees. In this section we demonstrate that, because our convex polygons are the convex hulls of disjoint trees, they behave much nicer than general convex polygons.

Let a *weakly disjoint pair* of convex polygons P, Q be a pair of convex polygons such that $P \setminus Q$ and $Q \setminus P$ are both connected sets of points, and P does not share a vertex with Q . Then a *weakly disjoint set* of polygons is a set of polygons such that all pairs of polygons are weakly disjoint. For simplicity, we assume that the convex hull of a line segment is a valid degenerate convex polygon consisting of two edges. We also assume all vertices are in general position. In this section we prove that weakly disjoint polygons are better behaved than general convex polygons, and that the convex hulls of disjoint trees are weakly disjoint.

Lemma 3 *If two convex polygons P, Q are weakly disjoint, then their boundaries intersect at at most two*

points.

Proof. Assume the intersection of their boundaries, $\partial P \cap \partial Q$, contains more than two points. Further, assume without loss of generality that P contains points outside of Q . Start at a point on P 's boundary ∂P that is outside of Q , and walk along ∂P . Eventually we intersect ∂Q , and now P is separated into two connected regions: points inside of Q , and points outside of Q . If we continue walking along ∂P , we eventually cross ∂Q again. Now there are three regions of P : two outside Q , and one inside Q , but the two outside Q may be the same. Continuing along ∂P we must eventually intersect Q again. Now the second outside region has been completed, and is clearly disconnected from the first. Therefore P and Q aren't weakly disjoint. \square

Lemma 4 *If two convex polygons P, Q are weakly disjoint, but not disjoint, then one contains a vertex of the other.*

Proof. If two convex polygons are not disjoint, then they have a non-empty intersection. If this intersection has no area, then they only share part of a boundary. However the vertices are in general position, so this cannot be the case. So their intersection has some non-zero area. Remark that the vertices of $P \cap Q$ are either vertices of P, Q , or points on $\partial P \cap \partial Q$. Since $P \cap Q$ has positive area, it must have at least 3 vertices. However, by Lemma 3, we know that there are at most two points in $\partial P \cap \partial Q$. So it follows that one of these three vertices must be a vertex of P or Q . Therefore a vertex of one is inside the other. \square

Lemma 5 *The convex hulls of two disjoint trees are weakly disjoint.*

Proof. Assume there exists two disjoint trees R, S , but their convex hulls are not weakly disjoint. Let $P = \text{Conv}(R)$ and $Q = \text{Conv}(S)$. If R and S share a vertex, then clearly they are not disjoint, and we have a contradiction. Then either $P \setminus Q$ is disconnected, or $Q \setminus P$ is. Assume without loss of generality that $P \setminus Q$ is disconnected. Then there exists two points $p, p' \in P \setminus Q$ such that there exists no path between p and p' inside of $P \setminus Q$. Since both P and Q are convex and share no vertices, the connected components p and p' are part of must contain a vertex of P . Therefore, without loss of generality, we may assume p and p' are vertices of P . However, that means p and p' are points on R , which has by definition a path that connects them. So either R and S intersect, or there exists a path between p and p' ; both of which are contradictions. Therefore, if two trees are disjoint, their convex hulls must be weakly disjoint. \square

Since the convex hulls of disjoint trees are weakly disjoint, unlike general convex polygons, finding the convex

hull of their union is simply a matter of finding at most two tangents to join them by. However, in merging two convex hulls it is no longer guaranteed that the new set of convex hulls has this property. Therefore, it would be desirable to merge convex hulls in some way in which we can maintain this property as an invariant.

3.2 Shoot and Insert

If two trees R and S in T have intersecting convex hulls, and we can find an edge to connect R and S without intersecting any other tree in T , then we have effectively merged the two trees, while maintaining the invariant of having a set of pairwise non-crossing trees.

Lemma 6 *Assume R and S are two non-crossing trees whose convex hulls intersect. Then the convex hull of one is strictly inside the other, or there exists a pair of adjacent vertices on the convex hull of one whose visibility is blocked by the other tree.*

Proof. By Lemma 4, we know that one contains a vertex of the other. Assume without loss of generality that a vertex r of $\text{Conv}(R)$ is inside of $\text{Conv}(S)$. If every other vertex of $\text{Conv}(R)$ is inside of $\text{Conv}(S)$, then $\text{Conv}(R)$ is strictly inside of $\text{Conv}(S)$ and we are done. Assume this is not the case. Then there exists some path along R from r to the outside of $\text{Conv}(S)$. This path must pass between two vertices of $\text{Conv}(S)$, and therefore obstruct their visibility. \square

Consider shooting a ray between the two vertices p, q of $\text{Conv}(S)$ that are obstructed by one or more other trees. This ray will necessarily intersect some other tree R first at a point q' . By definition, the edge pq' is an edge that joins R and S without intersecting any other tree. If this is the case, then we can stop shooting rays along S , replace S and R with $S \cup R \cup pq'$, and starting shooting rays along the convex hull of that new connected component. Furthermore, if we perform this process for all adjacent pairs of vertices of $\text{Conv}(S)$, and every ray reached the target vertex, we can conclude that either S is disjoint from all other convex hulls, or part of a well-nested hierarchy of boundary-disjoint convex hulls. If the former, then S is part of our output. If the latter, then the largest convex hull that contains S is part of our output.

Ishaque et al.[3] provide a ray shooting data structure that supports shooting rays from the boundary of obstacles, that are themselves inserted into the obstacles. Using their structure, a set of n pairwise disjoint polygonal obstacles can be preprocessed in $O(n \log n)$ time and space to support m permanent ray shootings in $O((n + m) \log^2 n + m \log m)$ time. Therefore shooting n rays takes $O(n \log^2 n)$ time. We refer to this data structure as *permashoot*.

3.3 Algorithm

We start by computing the sets

$$\mathcal{C} = \{\text{Conv}(T_i) | 1 \leq i \leq m\}$$

and

$$E = \{e | e \text{ is an edge of some element of } \mathcal{C}\}.$$

We build a permashoot instance R on T , and a union-find data structure U on T . The latter structure is used for determining what connected component a given edge is part of.

As long as E is non-empty, we do the following: Take an arbitrary edge e in E and remove it from E . If e is not stored in R , search in U for s , the component e is part of. Shoot a ray in R from one endpoint of e along e , and return the component r that was hit. If $s \neq r$, then merge $\text{Conv}(s)$ and $\text{Conv}(r)$ in \mathcal{C} ; remove and add edges from E to reflect the new state of \mathcal{C} ; and union s and r in U .

At this moment, the set E is empty. We perform a plane-sweep on \mathcal{C} , and return all the convex hulls that are not contained inside another convex hull.

An example is given in Figure 3.

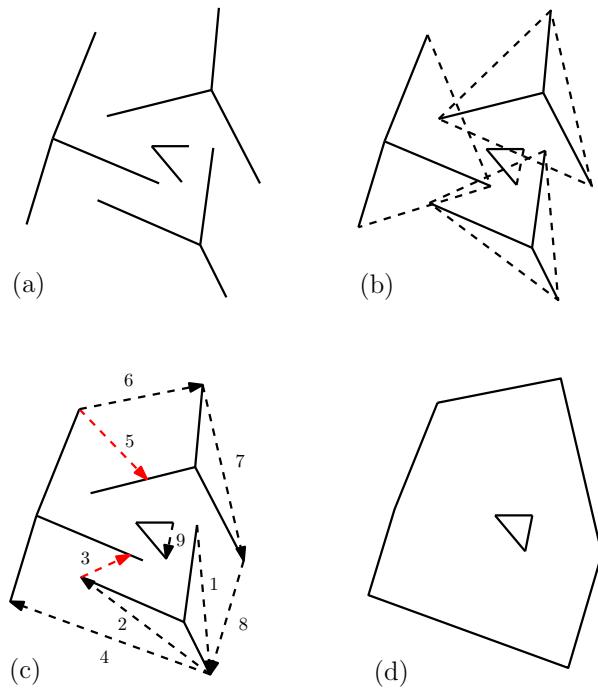


Figure 3: (a) The input; (b) Initial convex hulls of the input; (c) Rays shot by the algorithm (numbered in order they were shot), with rays that caused a merge in red; (d) Well nested hierarchy of hulls that results

Our algorithm shoots a ray for every edge of every convex hull. If any two convex hulls intersect, but are

not well-nested, then they are found during this process, and replaced by the convex hull of their union with an edge that joins them without intersecting any other components. This ensures that the invariant of having a set of pairwise weakly disjoint polygons holds. This continues until no more intersections are found in this way. By Lemma 6, we can conclude that we now have a set of convex hulls that are either disjoint, or part of a well-nested hierarchy. Our plane-sweep then finds all the maximal hulls, and returns only these.

3.4 Analysis

Because we maintain the invariant of having a set of pairwise weakly disjoint polygons, we know that each union adds at most two edges to the set of edges (the tangents between the two hulls). At worst, we perform $O(m) = O(n)$ unions, which adds $O(n)$ edges to check. Initially, there are $O(n)$ edges to check from the starting hulls. Therefore we end up shooting $O(n)$ rays, which takes $O(n \log^2 n)$ time.

For each ray shot we perform a constant amount of union and find operations to our union-find structure, each of which can easily be done in $O(\log n)$ time [2, Chapter 21]. So union-find only takes us $O(n \log n)$ time in total.

Merging two weakly disjoint convex hulls takes $O(\log n)$ time if we maintain them using height balanced binary search trees [5, Section 3.3.7]. Since we merge at most $O(m) = O(n)$ trees, merging the trees takes $O(n \log n)$ time.

Finally, the plane-sweep takes $O(n \log n)$ time to find all the maximal convex hulls.

Therefore our algorithm takes $O(n \log^2 n)$ time. This proves the following theorem.

Theorem 7 *The hull-cover of a set of pairwise non-crossing trees with a total of n vertices can be computed in $O(n \log^2 n)$ time.*

4 Computing the Box-Cover

We now assume that ϕ is the axis-aligned bounding box cover. We refer to the ϕ -cover as the *box-cover*.

Let $Box(S)$ be the axis-aligned bounding box of a tree S . A simple solution to box-cover is as follows. Create a dynamic range searching data structure that stores axis-aligned line segments and supports queries for those line segments in an axis-aligned query box. For each tree S in the input, query the structure for the segments found in the $Box(S)$. For each segment found, remove its parent bounding box from the structure. Then perform a query on the structure with the bounding box of all the boxes found in this way, plus the bounding box we just queried with. Repeat this until no segments are found. Then insert the last box we queried with into

the structure. Then run a plane sweep to find all the outermost boxes.

When our algorithm finishes inserting boxes we have a set of boundary-disjoint boxes, as in our hull-cover algorithm. Therefore, as before, it is correct.

Dynamic structures for axis-aligned segment queries exist that take $O(\log^2 n + k)$ time for queries, insertion, and deletion[4]. Since we start with an empty structure, preprocessing time is irrelevant. When we find an intersection, we replace $O(k)$ boxes with a single box. Since there are $O(m) = O(n)$ boxes, and each box gets inserted and removed at most once, it follows that our algorithm takes $O(n \log^2 n)$ time to perform this process in total. The plane sweep takes only $O(n \log n)$ time. Therefore, our algorithm takes $O(n \log^2 n)$ time in total. This proves the following theorem.

Theorem 8 *The box-cover of a set of pairwise non-crossing trees with a total of n vertices can be computed in $O(n \log^2 n)$ time.*

5 Conclusions and Open Problems

We are able to compute the solutions to hull-cover and box-cover in $O(n \log^2 n)$ time. However this is not obviously optimal. It remains to be seen whether there are better algorithms for these problems.

While the hull-cover is a potentially powerful preprocessing step for computing the actual coverage, the relationship between the two is fairly weak. In the best case the hull-cover is the convex hull of the input, and the two are the same. However in the worst case the hull-cover is exactly the input, but the coverage is something of size $\Omega(n^4)$.

Given a set \mathcal{O} of orientations, an \mathcal{O} -convex set S is a set of points such that every line with an orientation in \mathcal{O} has either an empty or connected intersection with S . The \mathcal{O} -hull of a set T of points is then the intersection of all \mathcal{O} -convex sets that contain T . When $\mathcal{O} = \{[0, 180]\}$, the \mathcal{O} -hull is the convex hull. When $\mathcal{O} = \emptyset$, the \mathcal{O} -hull is the identity function. The \mathcal{O} -hull satisfies our properties for being well-defined [6]. However, an algorithm for the general \mathcal{O} -hull is not immediately obvious. Further, it is unclear as to whether there are other non-trivial well-defined covering functions beyond the \mathcal{O} -hull and the axis-aligned bounding box. The geodesic hull does satisfy our properties, but without a bounding domain the geodesic hull is just the convex hull. We know from the start of the paper that the minimum enclosing circle does not produce well-defined results, and a similar argument applies to the minimum enclosing square.

Remark that our proof that general ϕ -covers are well-defined does not rely on the fact that we are working in two dimensions. This allows us to easily extend the problem into higher dimensions, where the convex hull and bounding box still work. However,

while our technique for bounding boxes generalizes to d -dimensions nicely, our technique for the convex hull does not. Therefore, a technique for computing the hull-cover that generalizes well would be desirable.

Acknowledgement

Special thanks to Pat Morin for consultation on certain proofs.

References

- [1] A. Bringmann and M. Smid. Computing the coverage of an opaque forest. pages 95–99. Canadian Conference on Computation Geometry, 2012.
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.
- [3] M. Ishaque, B. Speckmann, and C. Tóth. Shooting permanent rays among disjoint polygons in the plane. *SIAM Journal on Computing*, 41(4):1005–1027, 2012.
- [4] M. Overmars. *The Design of Dynamic Data Structures*. Lecture Notes in Computer Science. Springer, 1983.
- [5] F. Preparata and M. Shamos. *Computational geometry: an introduction*. Texts and monographs in computer science. Springer-Verlag, 1988.
- [6] G. J. Rawlins and D. Wood. Restricted-oriented convex sets. *Information sciences*, 54(3):263–281, 1991.

An Efficient Exact Algorithm for the Natural Wireless Localization Problem*

Bruno E. Crepaldi[†]

Pedro J. de Rezende[†]

Cid C. de Souza[†]

Abstract

Considered a variation of the art gallery problem, the wireless localization problem deals with the placement of the smallest number of broadcasting antennas required to satisfy some property within a given polygon. The case dealt with here consists of antennas that propagate a unique key within a certain antenna-specific angle of broadcast, so that the set of keys received at any given point is sufficient to determine whether that point is inside or outside the polygon. To ascertain this localization property, a Boolean formula must be produced along with the placement of the antennas.

In this paper, we propose an exact algorithm based on integer linear programming for solving the NP-hard natural wireless localization problem. The efficiency of our algorithm is certified by experimental results which include the solution of instances of up to 600 vertices in less than five minutes on a standard desktop computer.

1 Introduction

The Art Gallery Problem (AGP) [9, 10, 8] is a long-standing research topic in Computational Geometry. New problems of this type arose upon the introduction of a novel concept of visibility in which guards are able to see through the gallery boundary [7]. The motivation for this formulation originated from applications to wireless networks, where signals from antennas are not blocked by walls.

To illustrate this situation consider the following folkloric example, which captures the essence of the problem [1]. The owner of a café would like to provide wireless internet access to her customers while preventing those outside her shop to access the network infrastructure. To accomplish this, antennas may be installed, each of which broadcasting a unique (secret) key within an arbitrary but fixed angular range. The goal is to place these antennas and to adjust their angles of broadcast so that customers within the area of the café could be distinguished from those outside simply by having them name the keys received at their location. In a more formal way, one seeks to characterize the poly-

gon corresponding to the area of the shop by means of a monotone Boolean formula whose variables are the keys transmitted by the antennas. Since installation and maintenance of the antennas carry a cost, a natural optimization problem amounts to finding a solution with the minimum number of such devices.

Similarities between this problem and the traditional art gallery problem are self-evident, e.g., guards of the latter correspond to antennas in the former. Notwithstanding that the notions of visibility differ, henceforth we will use the term guard and antenna indistinctly.

As in the classical AGP, the *wireless localization problem* (WLP) has several variants depending on the choice of potential locations for guards, their angular range and maximum visibility distance. In this paper, we assume visibility to be unbounded.

Now, assume that the gallery floor plan is described by a simple polygon P . In the most general situation, guards may be placed anywhere inside P and can broadcast in any direction, in which case they are called *internal guards*. In a more restricted version, guard placement is limited to the vertices of P , and they are referred to as *vertex guards*. Moreover, another situation often found in the literature is the one known as *natural guarding*. Here, the guards are limited to lie on vertices or edges of P and to transmit their signals within the range corresponding to the interior angle of the polygon at that point.

The corresponding *Natural Wireless Localization Problem* (NWLP) is known to be NP-hard [2].

In [1] an alternative NP-hardness proof is given, which can be extended to more general types of guards, such as vertex and internal guards. There are also results [7, 6, 3] that lead to upper bounds on the number of guards sufficient for coverage, but these bounds are not always tight.

To the best of our knowledge, no exact algorithm has been proposed to this date to solve the NWLP. Furthermore, we are also unaware of any computational experiments reported in the literature for this problem.

Contribution This paper aims at filling these two gaps. To this end, in Sections 3 to 6, we model the NWLP problem as an integer program and in Section 7 we describe ingenious ways to use this formulation algorithmically. Computational results are presented in Section 8 validating this technique as a viable method for computing optimal solutions for instances comprised of hole-free polygons with up to 600 vertices. Conclusions

*This work was partially supported by grants from CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) and FAEPEX/UNICAMP.

[†]Institute of Computing, University of Campinas, Campinas, Brazil, brunoecrepaldi@gmail.com, {rezende|cid}@ic.unicamp.br

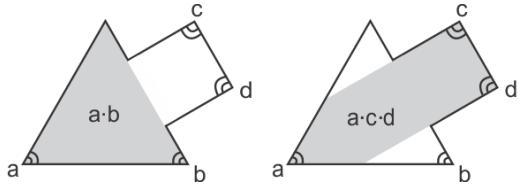


Figure 1: Polygon with guards on vertices a, b, c and d and Boolean formula $a \cdot b + a \cdot c \cdot d$.

and future directions follow.

2 Problem Definition and Terminology

A guard can be viewed as a wireless station positioned at a given location, which broadcasts a signal in a pre-defined *angle* and *direction*. The region, $Vis(g)$, covered by a guard g positioned at a point p is the cone with apex at p defined by two rays emanating from it. The bounding rays establish the *angle* and the *direction* of transmission of the corresponding guard in a natural way. Hence, from this point on, a *guard* will be identified to its cone of broadcast: the position of its apex and its angle of transmission.

We may now associate to a guard g a Boolean variable that, for every point p in the plane, takes a true value if and only if p belongs to $Vis(g)$. Given a polygon P and set of guards G , one may ask whether there exists a Boolean formula B on these variables that is satisfied uniquely on the points in P . In the affirmative case, G is said to form a *guarding* of P . Figure 1 illustrates this idea. For simplicity, in the remainder of the text, Boolean formulas are assumed to be in disjunctive normal form.

In the context of WLP, one is given a guard candidate set G known to contain a guarding of P . When a unitary cost is assigned to each guard in a guarding subset of G , the optimization problem seeks a guarding subset with minimum total cost. Variants of the problem depending on how the set of candidate guards G is defined can be formulated. Usually, G consist of a predefined finite set of locations, broadcasting angles and directions. Common locations for guards are the vertices and edges of P . In this work, we focus on the so-called *natural* guardings and on the resulting optimization problem, NWLP. A guard placed on a vertex of the polygon P is a *natural vertex guard* if its angle is the interior angle at that vertex, relative to P . A guard placed anywhere on an edge of P and broadcasting within an angle of π directed to the interior of P is called a *natural edge guard*. Since any two of these guards on a single edge would cover the same region, we can restrict the placement of natural edge guards to midpoints of edges. Accordingly, we will refer to a guarding consisting only of natural vertex and edge guards as a *natural guarding* [7].

3 Discretization

Viewing the NWLP as a continuous problem, for any point in the plane, the resulting Boolean formula should correctly identify whether it is inside or outside the polygon. In this section, we show that it actually suffices to ensure the validity of the formula for a finite set of points in the plane.

The rays on the boundary of the visibility regions of all natural guards define a planar arrangement. Notice that this arrangement coincides with the one obtained from the lines that support the edges of P . Moreover, since P has n edges, the planar subdivision induced by this arrangement has $O(n^2)$ faces. From here on, we use the term *face* to refer to a face of this subdivision. The next result shows that the correctness of a Boolean formula that solves the NWLP follows from its validity on any single point in each of these faces.

Lemma 1 *Given a simple polygon, let G be the set of its natural guards. Denote by $Sub(G)$ the planar subdivision induced by the visibility regions of all guards in G . A guard $g \in G$ covers one point in the interior of a face f of $Sub(G)$ if and only if g covers all points in f .*

Proof. Let $g \in G$ and let p be a point in the interior of a face f of $Sub(G)$ so that p is guarded by g . Suppose, by contradiction, that there exists a point q in f that is not guarded by g . Then, one of the rays that form the boundary of $Vis(g)$ must separate p from q . This contradicts the fact that f is a face of $Sub(G)$. The converse is immediate. \square

Recall that a Boolean formula that solves NWLP must be satisfied at all points in the closure of P but not at the external ones. Lemma 1 establishes that it suffices to verify this property at a single point per face of the resulting subdivision and hence on $O(n^2)$ points.

4 An Integer Programming Model

We now turn our attention to the algorithm we propose for solving the NWLP to optimality. It is divided into two phases: a preprocessing phase, where the discretization described in Section 3 is computed and a solution phase, where we create and solve an Integer Linear Programming (ILP) model. In this section, we describe this model.

Consider an instance of the NWLP in which a polygon P is given. Recall that a solution consists of a Boolean formula, in disjunctive normal form, that discriminates the points in P from the points in the exterior of P . We say that a Boolean variable accepts (rejects) a point if it is true (false) for that point. Similarly, it accepts (rejects) a face if it accepts (rejects) all points of that face. Therefore, it suffices to create a clause that accepts the points (a single point actually will do) of *each* internal face while rejecting the points of *all* external faces.

Clearly, redundant clauses (covering the same internal faces) may be eliminated in a post-processing phase.

Let G be the set of all natural guards of P and F be the set of faces of the corresponding planar subdivision. Denote by $F_P \subset F$ ($F_{\bar{P}} \subset F$) the subset of faces internal (external) to P . Furthermore, we denote by $C_f \subset G$ the set of guards which cover face f , and by $N_{fh} \subset C_f$ the subset of its guards that, while covering face f , do not cover face h .

To each $g \in G$, we associate a binary variable x_g , which is 1 whenever guard g is used in the solution and 0 otherwise. Moreover, to each guard g and interior face $f \in F_P$, we relate a binary variable y_{gf} , which is 1 if and only if the Boolean variable corresponding to guard g is part of the clause built to ensure that face f is satisfied by the Boolean formula. We now formulate the Integer Linear Program:

$$\begin{aligned} \min \quad & \sum_{g \in G} x_g, \\ \text{s.t.} \quad & \sum_{g \in C_f} y_{gf} \geq 1, \forall f \in F_P, \end{aligned} \quad (1)$$

$$\sum_{g \in N_{fh}} y_{gf} \geq 1, \forall f \in F_P, \forall h \in F_{\bar{P}}, \quad (2)$$

$$y_{gf} \leq x_g, \forall f \in F_P, \forall g \in C_f, \quad (3)$$

$$x_g \in \{0, 1\}, y_{gf} \in \{0, 1\}, \forall g \in G, \forall f \in F_P.$$

The objective function seeks to minimize the number of natural guards used. It is easy to see that the required Boolean formula may be built from the y_{gf} variables in the following fashion. A clause is associated to each $f \in F_P$ and the Boolean variable corresponding to a guard g will be part of this clause if, and only if, $y_{gf} = 1$. Constraints (1) guarantee that the internal faces are accepted, since at least one guard covers each face in F_P . Constraints (2) assure that exterior faces are not accepted by the formula, since for every pair of an internal face f and an external face h there is at least one guard accepting f and rejecting h . Constraints (3) prevent a clause from containing Boolean variables associated with a unused guard.

It is easy to modify this model so that the resulting Boolean formula is minimized along the process.

5 Strengthening the Model

Usual techniques to increase the computational efficiency of an ILP model amount to making it stronger in relation to dual bounds and more compact by reducing the number of constraints and variables in the formulation. In this section, we describe how these techniques can be applied to the model given in the previous section.

Firstly, notice that any single guard always covers external faces of the polygon, so, there is no point al-

lowing for a clause consisting of a single Boolean variable. Therefore, we can tighten constraints (1) to require at least two guards to cover any given internal face. This already leads to a slightly more restricted linear relaxation. However, we can strengthen the model even further as a consequence of the following lemma:

Lemma 2 *For every edge e of a polygon P , any feasible solution of P includes a guard whose visibility cone contains e on its boundary.*

Proof. Since e is an edge of P , there is a pair of faces $f \in F_P$ and $h \in F_{\bar{P}}$ adjacent to e on the subdivision induced by the (natural) guard candidates. If p and q are interior points of f and h , respectively, any Boolean formula that accepts p and rejects q must contain a Boolean variable that corresponds to a guard g whose cone contains p and excludes q . This is only possible if $Vis(g)$ is bounded by a ray that contains e , otherwise, both f and h would not be faces. \square

Let E denote the set of edges of P and G_e the set of natural guards g such that one of the rays that define $Vis(g)$ contains e . By Lemma 2, we can add the following constraints to the model:

$$\sum_{g \in G_e} x_g \geq 1, \forall e \in E \quad (4)$$

6 Shadow and Light Faces

Solving the ILP model proposed in Section 4 using all faces can be very costly. However, we can significantly reduce the number of faces considered in constraint (2) and still guarantee that the algorithm finds a valid formula using the minimum number of guards. To accomplish this, we can extend to the NWLP the notion of *shadow* and *light* faces, presented in [4].

Firstly, define a partial order \prec both on F_P and on $F_{\bar{P}}$ as follows. If $f, f' \in F_P$ ($\in F_{\bar{P}}$) then $f \prec f'$ if and only if $C_{f'} \subset C_f$. We call $f \in F_P$ ($\in F_{\bar{P}}$) an internal *shadow face* (external *light face*) if f is minimal (maximal) with respect to \prec .

Lemma 3 *If a Boolean formula accepts all internal shadow faces, then it accepts all internal faces.*

Proof. Let B be a Boolean formula that accepts all internal shadow faces. Let f be any internal face. If f is a shadow face, we are done. Suppose f is not a shadow face. Then, there must exist an internal shadow face f' such that $C_{f'} \subset C_f$. Since B accepts f' , there is at least one clause of B whose Boolean variables represent guards that cover f' . Since $C_{f'} \subset C_f$, this clause also accepts f . \square

Lemma 4 *If a Boolean formula rejects all external light faces, then it rejects all external faces.*

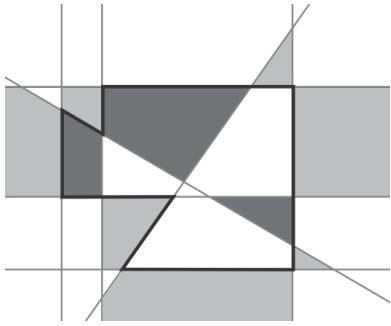


Figure 2: Example of internal shadow and external light faces of a polygon.

Proof. Analogous to the previous proof. \square

From Lemmas 3 and 4, the following theorem follows.

Theorem 5 *A Boolean formula is a solution to an instance of the NWLP if and only if it accepts all internal shadow faces and rejects all external light faces.*

Let S_P be the set of internal shadow faces and $L_{\bar{P}}$ be the set of external light faces. Theorem 5 implies that we can replace the sets F_P and $F_{\bar{P}}$ by the sets S_P and $L_{\bar{P}}$, respectively, hence reducing the size of the ILP model. Based on our experimental results, this reduction has a significant impact on the efficiency of our algorithm.

7 An Efficient Iterative Algorithm

In this section, we further enhance the model proposed in Section 4, by incorporating the strengthening and compression refinements proposed in Sections 5 and 6. Furthermore, we propose a more effective way for solving the model in order to arrive at a more efficient algorithm able to quickly handle instances of considerable size.

From constraints (3), the Boolean variable associated to a used guard might not be present in the clause liable for accepting a face covered by that guard. However, in order to make the model more compact, we may tighten the constraints (3) to $y_{gf} = x_g$, effectively requiring the clause responsible for accepting face f to contain *all* variables associated to used guards that cover f . Therefore, we can remove all variables y_{gf} , obtaining following

ILP model:

$$\min \sum_{g \in G} x_g,$$

$$\text{s.t. } \sum_{g \in C_f} x_g \geq 2, \forall f \in L_P, \quad (5)$$

$$\sum_{g \in N_{fh}} x_g \geq 1, \forall f \in L_P, \forall h \in S_{\bar{P}}, \quad (6)$$

$$\sum_{g \in G_e} x_g \geq 1, \forall e \in E \quad (7)$$

$$x_g \in \{0, 1\}, \forall g \in G.$$

This model finds a solution that minimizes the number of guards, but the resulting formula may be much larger than necessary, since the clause responsible for accepting a face f will have all variables that represent used guards that cover f . However, this model can be solved much more efficiently than the initial model and, for now, we are not particularly concerned with the length of the Boolean formula.

Let us look into the growth of the number of constraints (6) compared to the increase in the size of the instances (i.e., the number of edges of the input polygons). While the model contains only n constraints (7), the number of constraints (5) is $O(n^2)$ – proportional to the number of internal shadow faces. However, there is one constraint (6) for each pair of internal shadow and external light faces, leading to $O(n^4)$ of these constraints. Hence, if we found constraints (6) that we could avert checking, we might end up with a much smaller and more efficient model.

We observed, experimentally, that if a small set of constraints (6) are satisfied by the guards and clauses used, many other constraints (6) are automatically satisfied as well. Building upon this observation, we devised the following iterative algorithm.

Preprocessing phase. Two procedures are executed: the first one computes the visibility regions of the guards (cones) while the second one creates the planar subdivision and identifies the light and shadow faces.

Solution phase. The model is built without the constraints (6). Iteratively, the restricted model is solved to optimality and any violated constraints (6) are added to the model prior to the next iteration, until a viable (and optimal) solution is found.

8 Computational Experiments

In this section, we discuss the experimental investigation we carried out to evaluate the algorithm proposed in Section 7.

Our programs were coded in C++, compiled with GNU g++ 4.6, and made use of CGAL 4.1 (Computational Geometry Algorithms Library). The solver used to compute the ILP models was IBM ILOG CPLEX 12.2. As for

Table 1: Average number of faces.

Vertices	Internal Faces	Shadow Int Faces	External Faces	Light Ext Faces
20	49	12	161	22
40	211	38	610	69
60	493	73	1338	134
80	852	116	2389	233
100	1331	175	3720	355
200	5541	604	14560	1268
300	12566	1278	32585	2762
400	22549	2191	57652	4829
500	35124	3336	90127	7386
600	51968	4815	128333	10477

hardware, a desktop PC featuring an AMD Phenom II X6 1055T @ 2.80GHz and 8GB RAM was employed.

The instances tested correspond to simple polygons randomly generated by a procedure present in CGAL. This procedure starts off by randomly distributing the vertices of the polygon uniformly on a given rectangle and then applies the method of elimination of self-intersections using 2-opt moves. The instances that comprise our benchmark may be downloaded from www.ic.unicamp.br/~cid/Problem-instances/Wireless-Localization.

The number of vertices of the polygons associated to these instances was chosen in the ranges: [20, 100] with step size 20 and (100, 600] with step size of 100. For each polygon size, 30 instances were created.

The first aspect to be considered in our analysis relates to the reduction on the size of the original ILP model described in Section 4 as a consequence of the application of Theorem 5. Recall that the number of faces on the planar subdivisions is the main determining factor of the number of constraints in the model. Table 8 show the average number of *internal*, *external*, *shadow internal* and *light external* faces per polygon size. Using only the internal shadow and external light faces, we reduced the number of internal and external faces to be considered on average by $86.7\% \pm 4.7\%$ and $90.4\% \pm 1.7\%$, respectively. Taking into account that in the original model there is one constraint of type (2) for each pair of internal and external faces, when we limited these pairs to the internal shadow and external light faces, the number of constraints in the ILP formulation dropped by $98.6\% \pm 0.8\%$ on average. This huge decrease in the model size evoked by the results presented in Section 6 was one of the key ideas that made solutions of instances of hundreds of vertices possible.

As our algorithm has two phases, the next analysis focus on how the computation time breaks up between them. Figure 3 summarizes the average percentage of the time spent by the iterative algorithm in the preprocessing and solution phases. The average total time to solve an instance is displayed over each bar. Notice

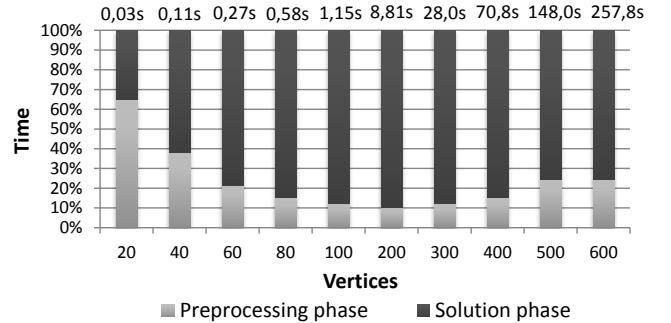


Figure 3: Average computation times, total and per phase.

that solutions we found for all instances in less than five minutes and the preprocessing phase took on average $23.7\% \pm 16.6\%$ of that time. These results show that our approach is a very good choice for calculating optimal solutions for the NWLP on polygons of several hundreds of vertices as they can be obtained in only a few minutes. The fact that preprocessing requires about one-third of the time spent by the solution phase may seem surprising at first. After all, the former is a polynomial time procedure while the latter involves multiple solutions of an NP-hard problem. However, as observed earlier in experiments on the classical AGP (see [5]), the current technology of ILP solvers is extremely advanced and allows for handling difficult problems very efficiently in practice.

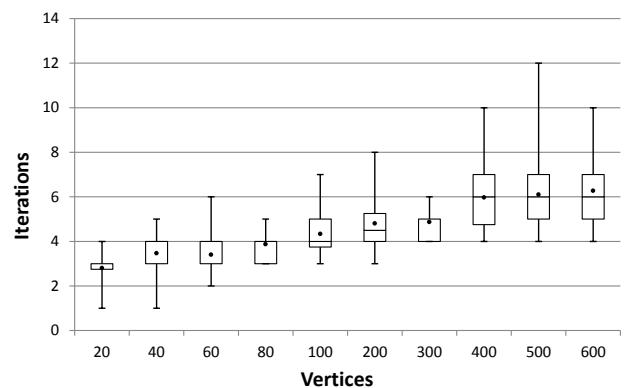


Figure 4: Number of iterations by polygon size.

An important point on the analysis of our algorithm relates to how the number of iterations increases with the size of the instances. This can be assessed by analyzing the data displayed in Figure 4. We see that, on average, 4.5 ± 1.2 iterations were sufficient to reach the optimum and that no instance in our benchmark required more than 12 iterations. Preliminary tests, where the *entire* initial ILP model was given as input to the solver, failed to attain optimal solutions on polygons of 50+ vertices within acceptable times. On the other

hand, the data in Figure 4 show that the number of iterations until convergence is reached is small. Bringing to mind that each iteration requires the solution of a much lighter ILP, we conclude that the iterative computation is indeed crucial in achieving the small computation times shown in Figure 3.

To perceive how much smaller the ILP models solved at each iteration are compared to the full model given in Section 7, we measured the number of constraints (6) added along the iterations and compared it to the total of constraints of this type. On average, in the last iteration of the algorithm, the model has only $0.6\% \pm 1.1\%$ of all constraints (6).

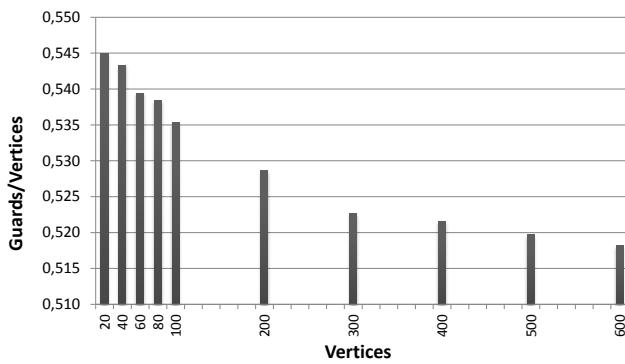


Figure 5: Ratio of guards to vertices by polygon size.

Lastly, an interesting insight on the guards per vertices ratio. It is known that, for a polygon of n edges, $n/2$ is a lower bound for the number of guards on an optimal solution. Figure 5 shows that for the random polygons in our benchmark, the number of guards used in the optimal solutions approaches $n/2$ as n increases.

9 Comments and Future Directions

To the best of our knowledge, this investigation on practical solutions to the natural wireless localization problem is unprecedented. Besides being known as an NP -hard problem [2] only a few theoretical studies on the NWLP have been undertaken [7, 6, 3].

The algorithm we proposed in this paper is based on an integer linear programming model and derives its effectiveness from an elaborate reduction on the number of constraints. An iterative approach has lead to significant gains in efficiency, which yielded solutions to instances of up to 600 vertices in less than five minutes of computation.

Extensions to this approach that might solve instances where the polygons contain holes or when antennas are not restricted to polygon vertices are worth investigating.

Heuristics or alternative ILP models may be compared to the results we described here by accessing our

set of benchmark instances made public together with the optimal solutions we found.

Lastly, we believe that the knowledge of exact solutions to a large collection of instances may lead to new theoretical developments on the NWLP, hence improving the understanding of the problem.

References

- [1] Tobias Christ and Michael Hoffmann. Wireless localization with vertex guards is NP -hard. In *Canadian Conference on Computational Geometry (CCCG)*, pages 149–152, 2009.
- [2] Tobias Christ, Michael Hoffmann, and Yoshi Okamoto. Natural wireless localization is NP -hard. In *Proceedings of the 25th European Workshop Comput. Geom.*, pages 175–178, 2009.
- [3] Tobias Christ, Michael Hoffmann, Yoshi Okamoto, and Takeaki Uno. Improved bounds for wireless localization. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory*, pages 77–89, Berlin, 2008. Springer-Verlag.
- [4] Marcelo C. Couto, Pedro J. de Rezende, and Cid C. de Souza. Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. In *7th International Workshop on Experimental Algorithms*, volume 5038 of *Lecture Notes in Computer Science*, pages 101–113, 2008.
- [5] Marcelo C. Couto, Cid C. de Souza, and Pedro J. de Rezende. An exact algorithm for minimizing vertex guards on art galleries. *International Transactions in Operational Research*, 18:425–448, 2011.
- [6] Mirela Damian, Robin Y. Flatland, Joseph O'Rourke, and Suneeta Ramaswami. A new lower bound on guard placement for wireless localization. *CoRR*, abs/0709.3554, 2007.
- [7] David Eppstein, Michael T. Goodrich, and Nodari Sitchinava. Guard placement for efficient point-in-polygon proofs. In *Symposium on Computational Geometry*, pages 27–36, 2007.
- [8] Alexander Kröller, Tobias Baumgartner, Sándor P. Fekete, and Christiane Schmidt. Exact solutions and bounds for general art gallery problems. *ACM Journal of Experimental Algorithms*, 17(1), 2012.
- [9] Joseph O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.
- [10] Jorge Urrutia. Art gallery and illumination problems. In *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.

On k -Enclosing Objects in a Coloured Point Set

Luis Barba* Stephane Durocher†‡ Robert Fraser† Ferran Hurtado§¶ Saeed Mehrabi†
 Debajyoti Mondal† Jason Morrison† Matthew Skala† Mohammad Abdul Wahid†

Abstract

We introduce the exact coloured k -enclosing object problem: given a set P of n points in \mathbb{R}^2 , each of which has an associated colour in $\{1, \dots, t\}$, and a vector $\mathbf{c} = (c_1, \dots, c_t)$, where $c_i \in \mathbb{Z}^+$ for each $1 \leq i \leq t$, find a region that contains exactly c_i points of P of colour i for each i . We examine the problems of finding exact coloured k -enclosing axis-aligned rectangles, squares, discs, and two-sided dominating regions in a t -coloured point set.

1 Introduction

Given a set P of n points in \mathbb{R}^2 and a positive integer k , the problem of finding a region (e.g., a disc, square, or rectangle) that encloses exactly k points of P while optimizing specific parameters (e.g., minimizing area or perimeter) has been examined extensively [3, 17, 19, 20, 25]. In many applications, the input data are classified into categories, or *colours*, leading us to consider the following natural generalization. Given a set P of n points in \mathbb{R}^2 , each of which has an associated colour in $\{1, \dots, t\}$, and a vector $\mathbf{c} = (c_1, c_2, \dots, c_t)$, where $c_i \in \mathbb{Z}^+$ for each $1 \leq i \leq t$, find a region enclosing at least c_i points in P of colour i for each i . Such problems commonly appear in pattern recognition [25] (e.g., when features are represented as a point set, and the objective is to identify a precise cluster with the prescribed number of features), as database queries (e.g., find a holiday destination with five tourist attractions, two hotels, and six restaurants), and in facility location [1] (e.g., selecting a location for a bus stop in a densely populated area). Unlike the smallest k -enclosing rectangle or disc problems, the solution to the exact coloured k -enclosing object problem may not always exist. Therefore, in the

exact coloured k -enclosing object problem, the primary objective is to find *any* coloured k -enclosing object that contains exactly the required number of points of each colour (if such a region exists), rather than finding the smallest such object. The problem is defined formally as follows.

EXACT COLOURED k -ENCLOSING OBJECT PROBLEM

INPUT: A set P of n points in \mathbb{R}^2 , each of which is assigned a colour in $\{1, \dots, t\}$, and a t -tuple $\mathbf{c} = (c_1, \dots, c_t)$, where $c_i \in \mathbb{Z}^+$ for each i .

QUESTION: Find a region (such as an axis-aligned rectangle, square, or disc) in \mathbb{R}^2 that encloses exactly c_i points of P of colour i for each i .

Although smallest k -enclosing object problems are well explored, very little is known about the exact coloured k -enclosing object problem. In this paper we introduce the exact coloured k -enclosing object problem for axis-aligned rectangles, squares, discs, and two-sided dominance regions in polychromatic point sets. Section 2 begins with an examination of related work. In Sections 3–5, we show that exact coloured k -enclosing axis-aligned rectangles, discs, and two-sided dominance regions can be found in $O(n^2k)$, $O(KVD(n, k))$, and $O(n \log n)$ time, respectively, where $KVD(n, k)$ denotes the time required to construct the k th order Voronoi diagram. In Section 6, we discuss generalizations to higher dimensions.

Throughout the paper, n denotes the number of points in P , t denotes the number of distinct colours of points in P , and k denotes the number of points to be contained in the bounding object, i.e., $k = \sum_{i=1}^t c_i$. Also, we assume that points are in general position.

2 Related Work

The exact coloured k -enclosing object problem generalizes several known problems, and was motivated by a desire to generalize the jumbled pattern matching problem to higher dimensions. Jumbled pattern matching [9, 10] asks whether a given sequence contains any permutation of some given query string. Given an arbitrary binary sequence of length n (i.e., $t = 2$), Burcsi et al. [9] show how to construct an $O(n)$ -space data structure in $O(n^2)$ preprocessing time that supports queries in $O(k)$ time

*Carleton University, Canada, and Université Libre de Bruxelles, Belgium. luis.barba@carleton.ca

†University of Manitoba, Canada.
{durocher,fraser,mehrabi,jyoti}@cs.umanitoba.ca,
[{mskala,wahid}@cs.umanitoba.ca](mailto:Jason.Morrison@umanitoba.ca)

‡Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

§Universitat Politècnica de Catalunya (UPC), Spain.
Ferran.Hurtado@upc.edu

¶Work of the author is supported in part by projects MINECO MTM2012-30951, Gen. Cat. DGR2009SGR1040, and ESF-EuroGIGA-CRP ComPoSe, MICINN EUI-EURC-2011-4306.

for any arbitrary query string of length k . Given an arbitrary sequence of length n (i.e., any $t \geq 2$), Burcsi et al. gave an $O(n)$ -space data structure with $O(n)$ preprocessing time to report all occurrences of a query string in $O(n\sqrt{t/(k \log t)})$ expected time. The one-dimensional exact coloured k -enclosing problem reduces to jumbled pattern matching, since a t -coloured set of n points in \mathbb{R} can be mapped to an array A of length n . One can slide an interval of width k over A to find an exact coloured k -enclosing interval in $O(n)$ time (or $O(n \log n)$ time if the point set is unsorted).

The problem of finding a smallest axis-aligned square, rectangle or disc that encloses k of n uncoloured points in \mathbb{R}^2 has been studied for over two decades. In 1991, Aggarwal et al. [3] showed that a smallest k -enclosing rectangle or square can be computed in $O(nk^2 \log n)$ time and $O(nk)$ space. Both the time and space complexities were subsequently improved several times [11, 17, 20, 29], and the current best known algorithms in \mathbb{R}^2 require $O(nk^2 + n \log n)$ time and $O(n)$ space for rectangles [20] and $O(n \log n + n \log^2 k)$ time and $O(n)$ space for squares [17]. The smallest k -enclosing disc problem also has a long history. The current best known algorithms require $O(n \log n + nk \log k)$ time and $O(nk + k \log^2 k)$ space [20], and $O(nk \log^2 n)$ time and $O(nk)$ space [19]. The lower bound on time is believed to be $\Omega(nk)$ [26].

A generalization of the smallest k -enclosing object problem with respect to a t -coloured point set is to find the smallest colour-spanning object, i.e., the smallest rectangle or disc that contains at least one point of each colour. Abellanas et al. [1] showed how to compute the smallest colour-spanning rectangle in $O(n(n-t) \log^2 t)$ time, which was later improved to $O(n(n-t) \log t)$ [16]. The best known algorithm for computing the smallest colour-spanning disc takes $O(nt \log n)$ time [22].

In \mathbb{R}^2 , the exact coloured k -enclosing rectangle problem reduces to a subarray sum problem considered by Takaoka [30] that, given an $m \times m$ array and a value v , asks to find a subarray such that the sum of its values is v . Takaoka showed that such a subarray can be computed in $O(m^3 \log m)$ time. The exact coloured k -enclosing rectangle problem can be solved in $O(n^3 \log n)$ time by reduction to this reverse range query problem. In Section 3, we show how to achieve $O(n^2k)$ time, improving the running time by a linear factor for small values of k .

Problems that involve finding discs that enclose a prescribed set of points with few outliers can be viewed as variants of the exact coloured k -enclosing object problem. For example, consider a point set with r red and b blue points, and the problem of finding a disc that encloses all of the red points and at most c_b blue points. Cheung and Daescu [13] showed that the existence of such a disc can be decided in $O(n + n^{1/4} c_b^{11/4} \log^{O(1)} n)$

time, and later gave an improved $O(rb \log b + r \log r)$ -time algorithm to find the smallest disc that minimizes the number of blue points [7]. Backer and Keil [5] showed that given a red-blue point set, an axis-aligned rectangle with the maximum number of red points but no blue points can be computed in $O(n \log^3 n)$ time. Dobkin et al. [18] considered the problem of computing a rectangle that maximizes the difference between the numbers of enclosed red and blue points, and gave an $O(n^2 \log n)$ -time algorithm to find such a rectangle.

Finding an exact coloured k -enclosing object is an inverse formulation of the range query problem in a poly-chromatic point set. One may consider such inverses for a variety of different shapes of query ranges. For example, in Section 5 we examine an inverse problem of the dominance range query that, given a t -coloured point set P in \mathbb{R}^2 , asks whether there is a point in P that dominates exactly c_i points of P of colour i for each i . JáJá et al. [23] gave an $O(n \log n / \log \log n)$ -space data structure for dominance counting queries, i.e., counting the number of points dominated by the query point, in $O(\log n / \log \log n)$ time. One could use such data structures for every colour class to determine whether there is a point in P that dominates exactly c_i points of P of colour i for each i , but it is not obvious how to combine the dominance counts efficiently since the dominating points returned will not coincide in general. In Section 5, we give an algorithm to solve this problem in $O(n \log n)$ time.

3 Axis-Parallel Rectangles

In this section, we study the exact coloured k -enclosing rectangle problem. Given a set P of n coloured points in \mathbb{R}^2 , where the colour of a point is an integer in $\{1, \dots, t\}$, and given a query as a t -tuple $\mathbf{c} = (c_1, \dots, c_t)$, where $\forall i, c_i \in \mathbb{Z}^+$, the problem is to determine whether there exists an axis-aligned rectangle which covers exactly c_i points of colour i for all i . The algorithm works by considering every possible choice for the top and bottom of the rectangle. That is, the algorithm checks whether a solution exists within any of the horizontal strips of the plane determined by a pair of horizontal lines passing through points of P .

We proceed by fixing the bottom of the strip and then increasing its height monotonically, i.e., new points are added to the strip, one by one, in order of increasing y -coordinates. We store all the points contained in the strip in a linked list \mathcal{L} sorted by x -coordinates in increasing order. As the height of the strip increases, new points are inserted to \mathcal{L} . Upon insertion of a point, we check each *window* of width k (i.e., a horizontal interval on the strip containing exactly k points) containing the new point (recall that $k = \sum_{i=1}^t c_i$). Since this involves sliding a fixed window from left to right, this check may

be performed in $O(k)$ time. For example, keep an array A of width t , where $A[i]$ is the number of points of colour i in the window, and a counter a that tracks the current number of colours satisfied in the query. When the window moves one step, a single point is added to the window and one is removed. For each of these, update $A[i]$ accordingly and compare $A[i]$ with c_i to see whether a should be updated. If $a = t$, then a solution exists containing the points in the window.

To insert a point in \mathcal{L} , we use a preprocessing step that allows us to perform this insertion in $O(1)$ time. Prior to starting the algorithm, sort the points of P and store them in order of increasing x -coordinates in a linked list L_x .

With the bottom of the strip fixed on a line through some point p_j of P , we assume that list L_x is updated to store only the points of P that lie above p_j . The preprocessing is described as follows. We copy list L_x into a list L in $O(n)$ time. Then, remove points from list L , one by one, in order of decreasing y -coordinates. Before removing a point p , we store two pointers $p.left$ and $p.right$ to the predecessor and successor of p in L .

After preprocessing, to insert a point p_i into \mathcal{L} , we splice it between $p_i.left$ and $p_i.right$. Because points were removed from top to bottom, both $p_i.left$ and $p_i.right$ have y -coordinates less than p_i . Thus, as the insertions into \mathcal{L} occur from bottom to top, both $p_i.left$ and $p_i.right$ belong to \mathcal{L} when p_i is inserted. Since L_x stored the points in sorted order by x -coordinates, p_i lies to the right of $p_i.left$ and to the left of $p_i.right$. Moreover, no point with y -coordinate less than p_i and larger than p_j lies between $p_i.left$ and $p_i.right$. The corresponding pseudocode is given in Algorithm 1.

Theorem 1 *The exact coloured k -enclosing rectangle problem can be solved in $O(n^2k)$ time.*

Proof. The outer for-loop iterates n times, and $O(n)$ -time preprocessing is performed on each iteration. An *insert* operation is performed in the inner loop in Step 11. As each insert requires only to splice a new point in the list, it is performed in $O(1)$ time. By Step 31, list L_x always contains the points at or above p_j sorted by x -coordinates and, consequently, list \mathcal{L} is also kept sorted in order of increasing x -coordinates.

Each iteration of the inner loop to determine whether a solution exists around the inserted point takes $O(k)$ time, resulting in a total running time of $O(n^2k)$.

For correctness, consider a solution whose lowest (resp., highest, leftmost, rightmost) point is p_{bot} (resp., p_{top} , p_ℓ , p_r). We consider all pairs of top and bottom points, so one iteration of the inner loop exists where $i = \text{top}$ and $j = \text{bot}$. Since p_{top} is the point inserted into the linked list at this step, the algorithm checks all rectangles whose highest and lowest points are p_{top} and p_{bot} and also contain exactly k points, one of which is p_{top} . Every solution, if any exists, is such a rectangle. \square

Algorithm 1 $\text{RECT}(P, \mathbf{c})$

```

1: Sort  $P \cup \{(0, -\infty), (0, \infty)\}$  by  $x$ -coordinates and
   store its points in order of increasing  $x$ -coordinates
   in a linked list  $L_x$ .
2: Sort  $P$  in order of increasing  $y$ -coordinates. Let  $p_i$ 
   be the  $i$ -th point in this ordering.
3:  $k \leftarrow \sum_{i=1}^t c_i$ 
4: for  $j := 0$  to  $n - 1$  do
5:   Copy list  $L_x$  into a list  $L$ .
6:   for  $i := n - 1$  to  $j + 1$  do
7:      $p_i.left \leftarrow \text{pred}_L(p_i)$ ,  $p_i.right \leftarrow \text{succ}_L(p_i)$ .
8:     Remove  $p_i$  from list  $L$ .
9:    $\mathcal{L} \leftarrow [(0, -\infty), p_j, (0, \infty)]$ 
10:  for  $i := j + 1$  to  $n - 1$  do
11:    Splice  $p_i$  into  $\mathcal{L}$  between  $p_i.left$  and  $p_i.right$ .
12:     $A[1 \dots t] \leftarrow 0$ ,  $a \leftarrow 0$ 
13:     $prev \leftarrow p_i$ ,  $next \leftarrow p_i$ 
14:    % Put the nodes for the  $k$  predecessors and
       successors of  $p_i$  into an array  $C$ .
15:    for  $l := 0$  to  $k$  do
16:       $C[k - l] \leftarrow prev$ ,  $prev \leftarrow \text{pred}_{\mathcal{L}}(prev)$ 
17:       $C[k + l] \leftarrow next$ ,  $next \leftarrow \text{succ}_{\mathcal{L}}(next)$ 
18:    % Slide a window of width  $k$  along the array.
19:    for  $l := 0$  to  $2k - 1$  do
20:       $cur \leftarrow \text{col}(C[l])$ 
21:      Increment  $A[cur]$ 
22:      If  $A[cur] = c_{cur}$  then increment  $a$ .
23:      If  $A[cur] = c_{cur} + 1$  then decrement  $a$ .
24:      if  $l \geq k$  then
25:         $cur \leftarrow \text{col}(C[l - k])$ 
26:        Decrement  $A[cur]$ 
27:        If  $A[cur] = c_{cur}$  then increment  $a$ .
28:        If  $A[cur] = c_{cur} - 1$  then decrement  $a$ .
29:      if  $a = t$  then
30:        return a rectangle bounded by  $p_j$ ,  $p_i$ ,
            $C[l]$ , and  $C[l - k + 1]$ 
31:    Remove  $p_j$  from list  $L_x$ .
32:    % There is no rectangle satisfying the query in  $P$ .
33:  return  $\emptyset$ 

```

See Section 6.1 for a discussion of modifications to Algorithm 1 to reduce running time.

4 Discs and Axis-Parallel Squares

The k th order Voronoi diagram of a set P of n points in the plane is a partition of the plane into maximal convex cells such that any two points in a common cell have the same k nearest neighbours in P . The number of k th order Voronoi cells is $\Theta(k(n - k))$ [24]. Thus, if C is a k th order Voronoi cell whose set of nearest neighbours is $P_C = \{p_1, \dots, p_k\} \subseteq P$, for any point $p \in C$, there exists a disc D_p centered at p such that $D_p \cap P = P_C$. If the points of P are coloured, it suffices

to verify whether there exists a k th order Voronoi cell C such that the frequencies of the colours of the points in P_C correspond to the input colour t -tuple \mathbf{c} .

Traversing the cells of the k th order Voronoi diagram by a breadth-first or depth-first search on the dual graph requires $O(k(n - k))$ steps. The sets of k nearest neighbours in any two adjacent cells C_a and C_b differ in exactly two points. Specifically, there exist points $\{p_a, p_b\} \subseteq P$ such that the edge e common to C_a and C_b is on the bisector of p_a and p_b and for any point in C_a close to e , p_a and p_b are respectively its k th and $(k + 1)$ st closest points in P , whereas the relationship is reversed for C_b . When transitioning from C_a to C_b , the set of k nearest neighbours is updated in $O(1)$ time by $P_{C_b} = (P_{C_a} \cup \{p_b\}) \setminus \{p_a\}$. We find the k nearest neighbours for the first cell in the traversal in $O(n)$ time using selection and partitioning, compute the frequencies of the corresponding colours, and initialize a count s of the number of frequencies that match the input t -tuple \mathbf{c} . Upon moving from the cell C_a to its neighbouring cell C_b during the traversal, it suffices to increment the frequency count for the colour of p_b , check whether this new value matches the corresponding value in \mathbf{c} , decrement the frequency count for the colour of p_a , check whether this new value matches the corresponding value in \mathbf{c} , and update s accordingly. If $s = t$, then a disc centered at any point in C_b with radius $r = \max_{q \in P_{C_b}} \text{dist}(p, q)$ is a solution to the exact coloured k -enclosing disc problem. Since the k th order Voronoi diagram has size $\Theta(k(n - k))$ in general, constructing it requires $\Omega(k(n - k))$ time in the worst case. This gives the following theorem.

Theorem 2 *The exact coloured k -enclosing disc problem can be solved in $O(KVD(n, k))$ time, where $KVD(n, k)$ denotes the time required to construct the k th order Voronoi diagram.*

Efficient deterministic algorithms for constructing the k th order Voronoi diagram include those of Chazelle and Edelsbrunner in $O(n^2 + k(n - k) \log^2 n)$ time using $O(n^2)$ space and $O(n^2 \log n + k(n - k) \log^2 n)$ time using $O(k(n - k))$ space [12], Lee in $O(nk^2 \log n)$ time using $O(n^2(n - k))$ space [24], and Aurenhammer in $O(nk^2 \log n)$ time using $O(k(n - k))$ space [4]. Efficient randomized algorithms include those of Clarkson in $O(n^{1+\epsilon}k)$ expected time for any fixed $\epsilon > 0$ [14], Ramos in $O(n \log n + nk 2^{c \log^* n})$ expected time, where c is constant [28], and Agarwal et al. in $O(k(n - k) \log n + n \log^3 n)$ expected time [2].

Under ℓ_∞ distance, a disc of radius r centered at a point p is realized as an axis-parallel square of side length $2r$ centered at point p . Consequently, just as we did for discs under ℓ_2 distance, the k th order Voronoi diagram under ℓ_∞ distance can be used to find a square that contains exactly c_i points of P of colour i for each i , if any such square exists. Equivalently, ℓ_1 distance

can be used with a $\pi/4$ rotation of the axes. Several of the algorithms for constructing the k th order Voronoi diagram under ℓ_2 distance can be applied under ℓ_1 or ℓ_∞ distance. For example, Lee [24] states that his $O(nk^2 \log n)$ -time algorithm applies to the ℓ_∞ and ℓ_p distance metrics for any $p \in [1, \infty)$. This gives the following corollary.

Corollary 3 *The exact coloured k -enclosing axis-parallel square problem can be solved in $O(\overline{KVD}(n, k))$ time, where $\overline{KVD}(n, k)$ denotes the time required to construct the k th order Voronoi diagram under the ℓ_∞ distance metric.*

5 Two-Sided Dominating Regions

Let P be a set of n points in the plane, each with a colour from $\{1, \dots, t\}$. For each i , let P_i denote the subset of P of colour i and let $n_i = |P_i|$. The point $p = (p_x, p_y)$ dominates the point $q = (q_x, q_y)$ if $p_x > q_x$ and $p_y > q_y$. We show how to determine in $O(n \log n)$ time if there exists some point r in the plane that dominates $\mathbf{c} = (c_1, \dots, c_t)$ points of P , i.e., r dominates c_i points of P_i for each i , and to return such a point r if one exists.

For each i , the region of points that dominate at least c_i points of P_i is bounded by a monotonic non-increasing orthogonal polygonal chain. The region of points that dominate exactly c_i points of P_i is bounded by two such chains. This boundary is defined by Bose and Morrison [8] as the c_i - and c_{i+1} -levels in P_i , consisting of a staircase that can be partitioned into an x -monotone set of $O(n_i)$ rectangles, as shown in Figure 1. Any solution point r must be contained in one of these rectangles. For each colour i , the corresponding rectangles are bounded by $O(n_i)$ segments [8] and can be constructed in $O(n_i)$ time (or $O(n_i \log n_i)$ time if the points must be sorted).

Once the t sets of candidate rectangles are constructed and stored in t lists, each in order of increasing x -coordinates, the rectangles for any pair of colours (i, j) can be intersected in $O(n_i + n_j)$ time. The time bound follows from the constant complexity of each set of rectangles for any given y -coordinate using Bentley and Ottman's line sweep [6]. Since any pair of rectangles intersect in either zero or one rectangle, recursive pairwise intersection of these sets requires only $O(n)$ space with $O(n)$ time per round and $O(\log t)$ rounds. Thus our algorithm requires $O(n \log t)$ time, $O(n \log n)$ preprocessing time for sorting, and $O(n)$ space.

Theorem 4 *The exact coloured k -enclosing two-sided dominating region problem can be solved in $O(n \log n)$ time.*

6 Discussion

In this section, we address generalizations and refinements of the exact coloured k -enclosing object problem.

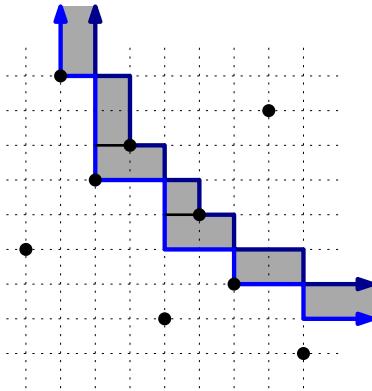


Figure 1: Points in the shaded region dominate exactly two black points. The partition into $O(n_i)$ rectangles is illustrated, at most one of which intersects any horizontal line. Analogous candidate regions are constructed for each colour set P_i . Their common intersection is non-empty if and only if there exists a solution to the k -enclosing two-sided dominating region problem.

6.1 Improved Time for Axis-Parallel Rectangles

Considering that the fastest known algorithm for the uncoloured version of the problem takes $O(nk^2 + n \log n)$ time in the worst case [20], the running time of Theorem 1 compares relatively favourably. Nevertheless, in this section we present two refinements which slightly improve our upper bound.

Since a solution must contain at least k points, Algorithm 1 can be modified so that the two outer loops skip k points and iterate $O(n - k)$ times each. This results in a running time of $O(n \log n + (n - k)^2 k)$. In the remainder of this section, we discuss reducing the factor k of the running time. Complete details are omitted due to space constraints.

Let m denote the most frequent colour in the query, i.e., $c_m = \max_{i \in \{1, \dots, t\}} c_i$. A point that does not have colour m is called \bar{m} -coloured. Let $k' = k - c_m$ denote the number of \bar{m} -coloured points in the query.

To improve the running time, the approach described in Algorithm 1 is used to search for a solution satisfying the query on the \bar{m} -coloured points. Upon finding a match, it is determined whether a rectangle (still an interval of the strip) containing exactly these \bar{m} -coloured points may also contain c_m m -coloured points.

To address this, the linked list \mathcal{L} of Section 3 is built containing only \bar{m} -coloured points. Given a point p_i in the strip, let $\text{succ}_{\mathcal{L}}^j(p_i)$ denote the j th successor in \mathcal{L} . Let $\text{succ}_{\mathcal{L}}^0(p_i) = p_i$, and let $n_m(\text{succ}_{\mathcal{L}}^j(p_i))$ denote the number of m -coloured points with x -coordinate between $\text{succ}_{\mathcal{L}}^{j-1}(p_i)$ and $\text{succ}_{\mathcal{L}}^j(p_i)$ in this strip. A rightmost dummy point p_∞ at $x = \infty$ is used so that $n_m(p_\infty)$ counts the m -coloured points to the right of the last \bar{m} -coloured point. Therefore, a solution exists with a

leftmost m -coloured point p_i if the query (except c_m) is satisfied by p_i and its $k' - 1$ successors in \mathcal{L} , and $\sum_{j=i+1}^{i+k'} n_m(\text{succ}_{\mathcal{L}}^j(p_i)) \leq c_m \leq \sum_{j=i}^{i+k'+1} n_m(\text{succ}_{\mathcal{L}}^j(p_i))$.

The challenge is to update the values of $n_m(i)$ and $n_m(i+1)$ following the insertion of an \bar{m} -coloured point. The preprocessing step may be augmented to track the number of m -coloured points on each side of an inserted \bar{m} -coloured point. For example, the disjoint set union data structure of Gabow and Tarjan [21] would allow (with some extra bookkeeping) to track the numbers of m -coloured points between consecutive \bar{m} -coloured points using *find* when encountering an m -coloured point, and storing the sizes of the sets prior to using *union* upon the removal of an \bar{m} -coloured point. Complete details are omitted due to space constraints. $O(n)$ union and find operations are performed on the data structure, which requires $O(n)$ time total. Therefore, the algorithm runs in $O(n \log n + n^2(k - \max_i c_i))$ time, which may again be improved by substituting $(n - k)^2$ for n^2 as discussed at the beginning of this section. Note that for a binary alphabet, this yields an overall running time of $O(n \log n + (n - k)^2 \min_i c_i)$.

6.2 Smallest Exact Coloured k -Enclosing Object

The algorithms described are straightforward to modify to return the *smallest* exact coloured k -enclosing object with at most an $O(k)$ increase in running time. For example, in the case of discs it suffices to compute the minimum enclosing disc of the k points associated with each candidate k th order Voronoi cell, which can be achieved in $O(k)$ time per cell using the algorithm of Megiddo [27]. In the case of axis-parallel rectangles, Algorithm 1 can be easily modified to compute the area of every window and return the smallest rectangle without any asymptotic increase in running time.

6.3 Higher Dimensions

Several of the algorithms described have natural generalizations to higher dimensions. For example, a k th order Voronoi diagram in \mathbb{R}^d has $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ cells [15], and so generalizing Theorem 2 to \mathbb{R}^d gives a running time of $O(d \cdot KVD_d(n, k))$, where $KVD_d(n, k)$ denotes the time required to construct the d -dimensional k th order Voronoi diagram. Similarly, Theorem 1 generalizes to give a running time of $O(n^{2(d-1)}kd)$.

6.4 Directions for Future Research

Several questions remain open. Can the time complexity be reduced in \mathbb{R}^2 ? For discs and axis-parallel squares, can the problem be solved faster than the time required to construct a k th order Voronoi diagram? Can the time complexity be improved if the input set of points is bichromatic (i.e., when $t = 2$)?

Acknowledgements

The authors thank the participants of the 2013 Bellairs Workshop on Geometry and Graphs for stimulating discussion of ideas related to this paper. The authors also thank Sharma Thankachan for discussion of query data structures for jumbled pattern matching in strings.

References

- [1] M. Abellanas, F. Hurtado, C. Icking, R. Klein, E. Langetepe, L. Ma, B. Palop, and V. Sacristan. Smallest color-spanning objects. In *Proc. ESA*, volume 2161 of *LNCS*, pages 278–289, 2001.
- [2] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comp.*, 27(3):654–667, 1998.
- [3] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12(1):38–56, 1991.
- [4] F. Aurenhammer. A new duality result concerning Voronoi diagrams. *Disc. & Comp. Geom.*, 5:243–254, 1990.
- [5] J. Backer and J. M. Keil. The bichromatic square and rectangle problems. *Technical Report 2009-01, University of Saskatchewan*, 2009.
- [6] J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-29:643–647, 1979.
- [7] S. Bitner, Y. K. Cheung, and O. Daescu. Minimum separating circle for bichromatic points in the plane. In *Proc. ISVD*, pages 50–55, 2010.
- [8] P. Bose and J. Morrison. Translating a star over a point set. In *Proc. CCCG*, pages 179–182, 2005.
- [9] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.
- [10] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On approximate jumbled pattern matching in strings. *Theory Comput. Syst.*, 50(1):35–51, 2012.
- [11] T. M. Chan. Geometric applications of a randomized optimization technique. *Disc. & Comp. Geom.*, 22(4):547–567, 1999.
- [12] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing k -th-order Voronoi diagrams. *IEEE Trans. Comp.*, 36(11):1349–1354, 1987.
- [13] Y. Cheung and O. Daescu. Minimum separating circle for bichromatic points by linear programming. In *Proc. FWCG*, 2010.
- [14] K. L. Clarkson. New applications of random sampling to computational geometry. *Disc. & Comp. Geom.*, 2:195–222, 1987.
- [15] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Disc. & Comp. Geom.*, 4:387–421, 1989.
- [16] S. Das, P. P. Goswami, and S. C. Nandy. Smallest color-spanning object revisited. *Int. J. Comp. Geom. & App.*, 19(5):457–478, 2009.
- [17] A. Datta, H.-P. Lenhof, C. Schwarz, and M. H. M. Smid. Static and dynamic algorithms for k -point clustering problems. *J. Algorithms*, 19(3):474–503, 1995.
- [18] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy with applications to computer graphics and machine learning. *J. Comp. & Sys. Sciences*, 52(3):453–470, 1996.
- [19] A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k -enclosing circle and related problems. *Comp. Geom.: Theory & App.*, 4(3):119–136, 1994.
- [20] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Disc. & Comp. Geom.*, 11:321–350, 1994.
- [21] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comp. & Sys. Sci.*, 30(2):209 – 221, 1985.
- [22] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Disc. & Comp. Geom.*, 9:267–291, 1993.
- [23] J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. ISAAC*, volume 3341 of *LNCS*, pages 558–568, 2004.
- [24] D. T. Lee. On k -nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comp.*, C-31:478–487, 1982.
- [25] P. R. S. Mahapatra, A. Karmakar, S. Das, and P. P. Goswami. k -enclosing axis-parallel square. In *Proc. ICCSA*, volume 6784 of *LNCS*, pages 84–93, 2011.
- [26] J. Matoušek. On geometric optimization with few violated constraints. *Disc. & Comp. Geom.*, 14:365–384, 1995.
- [27] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comp.*, 4:759–776, 1983.
- [28] E. A. Ramos. On range reporting, ray shooting and k -level construction. In *Proc. SoCG*, pages 390–399, 1999.
- [29] M. Segal and K. Kedem. Enclosing k points in the smallest axis parallel rectangle. *Inf. Proc. Let.*, 65(2):95–99, 1998.
- [30] T. Takaoka. The reverse problem of range query. *Elec. Notes Theor. Comp. Sc.*, 78:281–292, 2003.

On the Rectangle Escape Problem

Sepehr Assadi*

Ehsan Emamjomeh-Zadeh*

Sadra Yazdanbod*

Hamid Zarrabi-Zadeh*†

Abstract

Motivated by a bus routing application, we study the following *rectangle escape* problem: Given a set S of n rectangles inside a rectangular region R , extend each rectangle in S toward one of the four borders of R so that the maximum density over the region R is minimized, where the density of each point $p \in R$ is defined as the number of extended rectangles containing p . We show that the problem is hard to approximate to within a factor better than $3/2$ in general. When the optimal density is sufficiently large, we provide a randomized algorithm that achieves an approximation factor of $1 + \varepsilon$ with high probability improving upon the current best 4-approximation algorithm available for the problem. When the optimal density is one, we provide an exact algorithm that finds an optimal solution in $O(n^4)$ time, improving upon the current best $O(n^6)$ -time algorithm.

1 Introduction

Consider a set of electrical components (e.g., chips) placed on a printed circuit board (PCB), where both the board and the chips are axis-parallel rectangles. We want to connect each chip to one of the four sides of the board using a rectangular bus (see Figure 1). The goal is to find a routing direction for the chips so that the maximum number of bus conflicts at any single point over the board is minimized. This is equivalent to minimizing the number of layers needed for routing all the chips on the board. The problem is called the *rectangle escape problem* [3], and has been extensively studied in the literature (see, e.g., [1, 2, 3, 4, 5, 7, 8, 9, 10]). The problem is formally defined as follows:

Problem 1 (Rectangle Escape Problem (REP))

Given an axis-parallel rectangular region R , and a set S of n axis-parallel rectangles inside R , extend each rectangle in S toward one of the four borders of R , so that the maximum density over R is minimized, where the density of a point $p \in R$ is defined as the number of extended rectangles containing p .

*Department of Computer Engineering, Sharif University of Technology. {s_asadi,emamjomeh,yazdanbod}@ce.sharif.edu, zarrabi@sharif.edu

†School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.

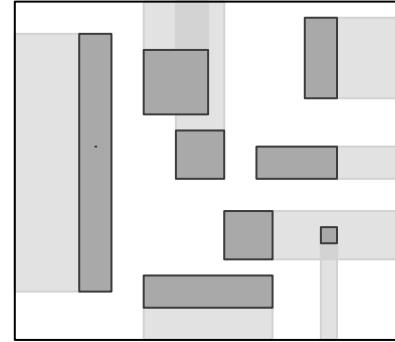


Figure 1: An instance of the rectangle escape problem. Chips are shown in dark, and buses in light gray.

An example of the rectangle escape problem is illustrated in Figure 1. In this example, the optimal density, which is equal to the minimum number of layers needed for routing the chips is two.

The rectangle escape problem is known to be NP-hard [3]. The *decision version* of the problem, called k -REP, is defined as follows: Given an instance of the rectangle escape problem and an integer $k \geq 1$, determine whether any routing is possible with a density of at most k . It is known that the k -REP problem is NP-complete, even for $k = 3$ [3]. The best current approximation algorithm for the optimization version of the problem is due to Ma *et al.* [3] that achieves an approximation factor of 4, using a deterministic linear programming (LP) rounding technique.

For a special case when the optimal density is 1 (i.e., when all chips can be routed with no conflict), the problem can be solved exactly using a polynomial-time algorithm for the related *maximum disjoint subset* problem, for which an $O(n^6)$ -time algorithm is proposed by Kong *et al.* [1].

Our results. In this paper, we obtain some new results on the rectangle escape problem, a summary of which is listed below.

- We show that the k -REP problem is NP-complete for any $k \geq 2$. Given that the problem is polynomially solvable for $k = 1$, this fully settles the complexity of the problem for all values of k . An important implication of this result is that the rectangle escape problem is hard to approximate to within any factor better than $3/2$, unless $P = NP$.

- We present a new algorithm that solves the 1-REP problem in $O(n^4)$ time, improving upon the current best solution for the problem that requires $O(n^6)$ time [1]. Our algorithm can indeed solve the following more general optimization version of the problem: given an instance of the rectangle escape problem, find a maximum-size subset of rectangles in S that can be routed disjointly.
- Despite the fact that the problem is hard to approximate to within a constant factor when the optimal density is low, we present a randomized algorithm that achieves an approximation factor of $1 + \varepsilon$ with high probability, when the optimal density is at least $c_\varepsilon \log n$, for some constant c_ε . This improves, for instances with high density, upon the current best algorithm of Ma *et al.* [3] that guarantees an approximation factor of 4 for all instances. Our algorithm is based on a randomized rounding technique applied to a linear programming formulation of the problem.

2 Hardness Result

We first show that the k -REP problem is NP-complete, for any $k \geq 2$. As a corollary, we show that the rectangle escape problem is hard to approximate to within any factor better than $3/2$, unless $P = NP$. Our hardness result holds even in a more restricted setting where the input rectangles are all disjoint.

Theorem 1 *The k -REP problem is NP-complete for $k \geq 2$, even if all input rectangles are disjoint.*

Proof. We prove by reduction from 3-SAT. The reduction is similar to that of [3], but uses a more clever construction to handle the special case of $k = 2$, and a more restricted setting where all rectangles are disjoint. Given an instance of 3-SAT, we create an instance of 2-REP as follows. Fix a rectangular region R . We partition R into four (virtual) sub-regions, labeled with top, left, variables, and clauses, as shown in Figure 2. Then, we start building a set of rectangles S inside R as follows. We first add one long rectangle to the right side of the variables region, and three long rectangles to the left, right, and bottom sides of the clauses region, as shown in Figure 2. The following rectangles are then added to S .

- For each variable x_i , we add a pair of “variable rectangles” v_i and \bar{v}_i along each other to the variables region in such a way that no two rectangles from different variables can be stabbed by a single horizontal or vertical line.
- For each clause C_j , we add three “literal rectangles” in a horizontal row in the clauses region. Each literal rectangle is placed beneath a variable rectangle

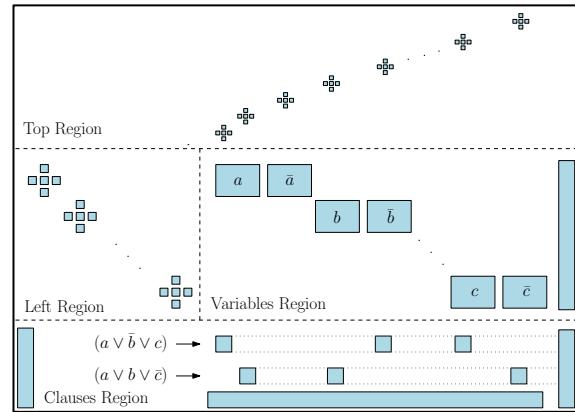


Figure 2: Reduction from 3-SAT to 2-REP.

corresponding to the literal appeared in the clause. Again, no two literal rectangles intersect, and no two of them can be stabbed by a vertical line.

- For each variable, we add a “block gadget” to the left region, directly to the left of the corresponding variable row. Each gadget is composed of five smaller rectangles in a cross-shape arrangement, as shown in Figure 2. Likewise, for each literal in each clause, we add a block gadget to the top region directly above the corresponding literal rectangle. If a literal appears in no clause, we add a block gadget above the corresponding variable rectangle in the top region. The block gadgets are placed in a way that no two rectangles from different gadgets can be stabbed by a single horizontal or vertical line.

Now, we claim that the answer to the constructed instance of 2-REP is yes if and only if the corresponding 3-SAT instance is satisfiable. First, suppose that the answer to the 2-REP is yes, i.e., there is a proper routing of rectangles with a density of at most 2. We show that there is a satisfying assignment for the 3-SAT instance, in which a literal is set to true (resp., false), if the corresponding variable rectangle is routed rightward (resp., downward). To show this, first observe that for each variable v_i , the two variable rectangles v_i and \bar{v}_i cannot be routed simultaneously to the right, because otherwise, they will cause a density of 3 on the rectangle located to the right side of the variables region. Moreover, for each gadget in the top and the left region, the density over at least one of the gadget rectangles is more than one, and hence, in a proper routing of rectangles, no variable rectangle can be routed neither to the top, nor to the left side.

For each clause, observe that none of its three literal rectangles can escape upward because of the block gadgets in the top region, and no two of them can escape simultaneously to neither left nor right, because

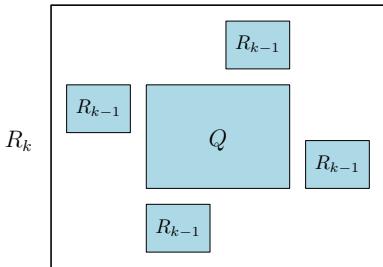


Figure 3: Constructing an instance of k -REP from four instances of $(k-1)$ -REP.

of the rectangles put on the left and the right sides of the clauses region. Therefore, at least one literal rectangle from each clause must be routed downward. Furthermore, notice that if a variable rectangle escapes downward, none of the literal rectangles below it can be routed downward, because of the rectangle put at the bottom side of the clauses region.

Now, given a proper routing of the 2-REP instance, we set variable v_i in the 3-SAT instance to 1 if rectangle v_i escape to the right, otherwise, we set it to 0. Note that rectangles for v_i and \bar{v}_i cannot simultaneously escape to the right, so this assignment is feasible. Moreover, for each clause, at least one of its literal rectangles, say x_i , must escape downward, meaning that its corresponding variable x_i is set to 1 for sure, and thus the clause is satisfied. Therefore, the 3-SAT instance is satisfiable. The opposite side can be proved using the same exact mapping, and taking into account the fact that there is a proper routing for the top and the left gadget rectangles, in which they do not interfere with the rectangles in the variables and the clauses regions. This completes the NP-completeness proof for $k = 2$.

To show NP-completeness for other values of $k > 2$, we use the following recursive construction. Let R_{k-1} be an instance of $(k-1)$ -REP. We construct an instance R_k of k -REP by putting a large rectangle Q in the middle, and four instances of R_{k-1} around Q , as shown in Figure 3. The four instances are placed in a way that no horizontal or vertical line can simultaneously stab any two of them. Now, suppose that R_k has a proper routing of density k . In this routing, Q escapes to one of the four directions, and hence, one of the R_{k-1} instances must have a proper routing of density $k-1$. Therefore, the corresponding 3-SAT instance is satisfiable by induction. The opposite side can be proved analogously (details are omitted in this version). \square

As a corollary of Theorem 1, we obtain the following inapproximability result.

Theorem 2 *For any $\alpha < 3/2$, there is no α -approximation algorithm for the rectangle escape problem, even if all input rectangles are disjoint, unless $P = NP$.*

Proof. Suppose by way of contradiction that there is an algorithm with an approximation factor of $\alpha < 3/2$. If we run this algorithm on an instance of the rectangle escape problem with an optimal density of 2, the algorithm must return a solution with density less than $3/2 \times 2$, which is at most 2 due to the integrality of the density. Such an algorithm solves the 2-REP problem exactly, which is a contradiction. \square

3 An Exact Algorithm for Unit Density

In this section, we present a dynamic programming algorithm that solves the 1-REP problem in $O(n^4)$ time, improving upon the previous solution due to Kong *et al.* [1] that requires $O(n^6)$ time. Our algorithm solves the following optimization problem.

Problem 2 (Maximum Disjoint Routing) *Given an instance of the rectangle escape problem (Problem 1) with disjoint rectangles, find the maximum number of rectangles that can be routed disjointly, i.e., with unit density.*

It is easy to observe that any algorithm for Problem 2 can also solve 1-REP: we first find the maximum number of rectangles that can be routed disjointly, and then verify if this number is equal to n . Note that in the above definition, the initial locations of unescaped rectangles are also important: an escaped rectangle cannot collide with any other rectangle, even if that rectangle is not escaped.

Let R_1, \dots, R_n be the input rectangles, sorted in decreasing order of the y -coordinates of their bottom sides. For a rectangle R_i , the direction $d \in \{\text{left}, \text{right}, \text{up}, \text{down}\}$ is said to be *free* if by escaping toward that direction, R_i does not collide with any other rectangle in its initial place. Note that the freeness of direction d for R_i is independent of the escaping direction of other rectangles. Furthermore, we define the set $\{v_1, \dots, v_k\}$ ($k \leq 2n$) as the set of all vertical lines obtained by extending the vertical sides of the rectangles, sorted from left to right.

To solve Problem 2, we first solve two simpler cases in which the escaping directions are only vertical. Given integers $0 \leq i \leq n$ and $1 \leq l, r \leq k$, we define the following two subroutines:

- **ONE-DIRECTION(i, l, r):** returns the maximum number of rectangles among R_1, \dots, R_i that are between v_l and v_r and can be routed upward in unit density.
- **TWO-DIRECTIONS(i, l, r):** returns the maximum number of rectangles among R_1, \dots, R_i that are between v_l and v_r and can be routed either upward or downward in unit density.

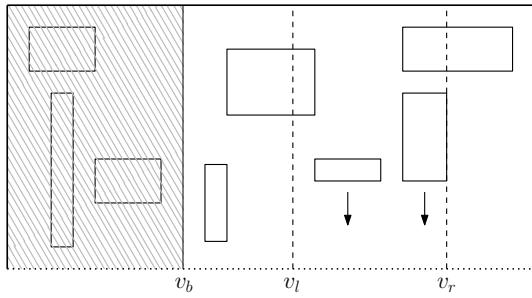


Figure 4: Illustrating Problem 3.

For each triple (i, l, r) , the value of both ONE-DIRECTION(i, l, r) and TWO-DIRECTIONS(i, l, r) can be calculated by the following simple greedy algorithm. For each rectangle R_j ($1 \leq j \leq i$) between v_l and v_r , find a free direction upward (and downward, depending on the subproblem). If such direction exists, route R through that direction. Note that routing a rectangle vertically poses no additional restriction on other rectangles in these two subproblems. Next, we define the following additional subproblem.

Problem 3 (No-Left-Escape) *Given integers $0 \leq i \leq n$ and $1 \leq b, l, r \leq k$, NO-LEFT-ESCAPE(i, b, l, r), is defined as the maximum number of rectangles among R_1, \dots, R_i which can be routed in unit density under the following restrictions:*

- only rectangles to the right of v_b are allowed to escape,
- no rectangle is allowed to escape leftward, and
- only rectangle between v_l and v_r are allowed to escape downward.

See Figure 4 for an illustration. To find the value of NO-LEFT-ESCAPE(i, b, l, r) recursively, we consider all possible actions for R_i . The first possible action for R_i is not to escape at all. In this case, the solution is equal to the solution of NO-LEFT-ESCAPE($i - 1, b, l, r$). The other possible three actions for R_i are listed below. In what follows, we assume that the considered direction is *free* for R_i , and that R_i is allowed to escape through that direction according to the problem restrictions described above. Otherwise, we simply rule out that direction from the possible actions of R_i . Let v_α and v_β be the vertical lines obtained by extending the left and the right sides of R_i , respectively.

- *Downward* If R_i escapes downward, the maximum number of rectangles among R_1, \dots, R_{i-1} that can escape is equal to NO-LEFT-ESCAPE($i - 1, b, l, r$), since routing R_i imposes no new restriction on R_1, \dots, R_{i-1} .

Algorithm 1 MAX-ROUTE(i, l, r)

```

1: if  $i = 0$  then
2:   return 0
3:  $ans_n \leftarrow$  MAX-ROUTE( $i - 1, l, r$ )
4:  $ans_d \leftarrow ans_u \leftarrow ans_l \leftarrow ans_r \leftarrow 0$ 
5:  $\alpha, \beta \leftarrow$  indices of the vertical lines through the left
   and the right sides of  $R_i$ , respectively.
6: if down is feasible for  $R_i$  then
7:    $ans_d \leftarrow$  MAX-ROUTE( $i - 1, l, r$ ) + 1
8: if left is feasible for  $R_i$  then
9:    $ans_l \leftarrow$  MAX-ROUTE( $i - 1, \max\{l, \beta\}, r$ ) + 1
10: if right is feasible for  $R_i$  then
11:    $ans_r \leftarrow$  MAX-ROUTE( $i - 1, l, \min\{r, \alpha\}$ ) + 1
12: if up is feasible for  $R_i$  then
13:    $ans_u \leftarrow$  NO-RIGHT-ESCAPE( $i - 1, \alpha, l, r$ ) + NO-
      LEFT-ESCAPE( $i - 1, \beta, l, r$ ) + 1
14: return  $\max\{ans_n, ans_d, ans_u, ans_l, ans_r\}$ 

```

• *Upward* If R_i escapes upward, one additional restriction must be considered: rectangles not to the right of v_β cannot escape rightward. Therefore, by the problem definition, each rectangle between v_b and v_β can only escape upward or downward. As such, escaping the maximum number of rectangles between v_b and v_β can be solved independently using subroutines ONE-DIRECTION and TWO-DIRECTIONS, depending on the position of v_l and v_r . The rectangles to the right of v_β form another subproblem, whose optimal answer is NO-LEFT-ESCAPE($i - 1, \beta, l, r$).

• *Rightward* By escaping rightward, one more restriction is posed to other rectangles: for any $1 \leq j < i$, R_j can escape downward if its initial place is not only to the left of v_r , but is also to the left of v_α . It means that if initial position of R_j is not to the left of $v_{\min\{r, \alpha\}}$, it cannot be routed downward. Therefore, the optimum answer for R_1, \dots, R_{i-1} in this case is NO-LEFT-ESCAPE($i - 1, b, l, \min\{r, \alpha\}$).

The *No-Right-Escape* is analogously defined, and can be solved similarly. Now, we have all ingredients necessary to solve Problem 2. Indeed, we solve the following more general problem:

Problem 4 (Max-Route) *Given integers $0 \leq i \leq n$ and $1 \leq l, r \leq k$, find the maximum number of rectangles among R_1, \dots, R_i that can be routed in unit density under the following restriction: if a rectangle is not between v_l and v_r , it is not allowed to escape downward.*

The procedure MAX-ROUTE(i, l, r) defined in Algorithm 1 solves the problem as follows. We consider all

possible actions for R_i . Except for escaping upward, all remaining actions can be solved like the previous problems. When R_i escapes upward, it is enough to calculate the sum of $\text{NO-LEFT-ESCAPE}(i - 1, \beta, r, l)$ and $\text{NO-RIGHT-ESCAPE}(i - 1, \alpha, r, l)$, since routing rectangles to the left of v_α and routing rectangles to the right of v_β are two independent subproblems.

Lemma 3 *Problem 4 can be solved in $O(n^4)$ time.*

Proof. To solve this problem, consider a dynamic-programming version of MAX-ROUTE algorithm. First, using a greedy algorithm, solve the ONE-DIRECTION and TWO-DIRECTIONS problems for any tuple (i, l, r) , and store them in a table. This can be done in $O(n^4)$ time. Then, by the definition of problem 3, we can solve NO-LEFT-ESCAPE and NO-RIGHT-ESCAPE independently using dynamic programming. Note that in dynamic programming, the value of each tuple (i, b, l, r) can be obtained in $O(1)$ time from four previously-calculated values as described above. Putting all together, by using the description of Problem 4, each value of $\text{MAX-ROUTE}(i, l, r)$ can be obtained from the previously-calculated values of this function, or solutions of NO-LEFT-ESCAPE and NO-RIGHT-ESCAPE. This can be done in $O(1)$ time assuming that the previous values are stored in a table. Thus, using a dynamic programming algorithm, Problem 4 can be solved in $O(n^4)$ time and space. \square

The following theorem summarizes the result of this section.

Theorem 4 *1-REP can be solved in $O(n^4)$ time.*

Proof. Observe that the answer to 1-REP is *yes* iff the answer to Problem 4 for $(n, 1, k)$ is equal to n , where k is the index of the rightmost vertical line. The running time therefore follows from Lemma 3. \square

4 A Randomized Approximation Algorithm

As noted in Section 2, the rectangle escape problem is NP-hard, even when the optimal density is 2. Therefore, it is natural to look for approximation algorithms for the problem. The current best approximation algorithm is due to Ma *et al.* [3], which achieves an approximation factor of 4. The algorithm is based on a deterministic rounding of an integer programming formulation of the problem. In this section, we show that a standard randomized rounding technique [6] applied to the same integer programming formulation of the problem, yields an approximation factor of $1 + \varepsilon$, when the optimal density is at least $c_\varepsilon \log n$, for some constant c_ε .

The integer programming formulation of the problem is as follows. Let $S = \{r_1, \dots, r_n\}$ be the set of input rectangles inside a region R . We build a grid on top of R

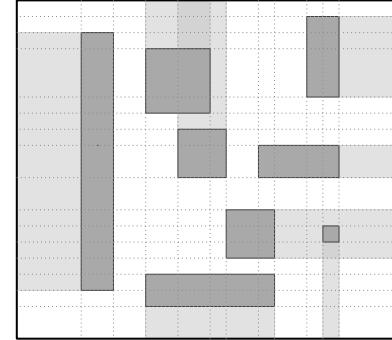


Figure 5: The grid cells for an instance of the rectangle escape problem.

Algorithm 2 RANDOMIZED-ROUNDING

- 1: find an optimal solution x^* to the LP relaxation
 - 2: route each r_i to exactly one direction λ according to the probability distribution $x_{i,\lambda}^*$
-

by extending each side of the rectangles in S into a line (see Figure 5). This partitions R into a set \mathcal{C} of $O(n^2)$ grid cells, where the density over each cell is fixed.

For each rectangle r_i , we define four 0-1 variables $x_{i,l}, x_{i,r}, x_{i,u}$, and $x_{i,d}$, corresponding to the four directions left, right, up, and down, respectively. For a direction $\lambda \in \{l, r, u, d\}$, we set $x_{i,\lambda} = 1$ if r_i is escaped toward direction λ , otherwise, $x_{i,\lambda} = 0$. Since any rectangle r_i can escape toward only one direction, we have the constraint $x_{i,l} + x_{i,r} + x_{i,u} + x_{i,d} = 1$. For each grid cell $c \in \mathcal{C}$, let $P_c = \{(i, \lambda) \mid r_i \text{ passes } c \text{ if it goes toward direction } \lambda\}$. Note that if cell c is contained in r_i , then $(i, \lambda) \in P_c$ for all directions λ . Let Z be the maximum density over the region R . Then, for each grid cell $c \in \mathcal{C}$ we can add the constraint $\sum_{(i,\lambda) \in P_c} x_{i,\lambda} \leq Z$. Now, the problem can be formulated as the following integer program.

$$\begin{aligned} & \text{minimize} && Z \\ & \text{subject to} && \sum_{(i,\lambda) \in P_c} x_{i,\lambda} \leq Z && \forall c \in \mathcal{C} \\ & && x_{i,l} + x_{i,r} + x_{i,u} + x_{i,d} \geq 1 && \forall 1 \leq i \leq n \\ & && x_{i,l}, x_{i,r}, x_{i,u}, x_{i,d} \in \{0, 1\} && \forall 1 \leq i \leq n \end{aligned}$$

The randomized rounding algorithm for the rectangle escape problem is provided in Algorithm 2. The algorithm works as follows. We first relax the integer program to a linear program by replacing the constraints $x_{i,\lambda} \in \{0, 1\}$ with $x_{i,\lambda} \geq 0$, and solve the linear programming relaxation to obtain a solution x^* with objective value Z^* . Then, we randomly route each rectangle to exactly one direction by interpreting the value of $x_{i,\lambda}^*$ as the probability of routing r_i toward direction λ .

Theorem 5 Algorithm 2 is a $(1 + \varepsilon)$ -approximation algorithm for the rectangle escape problem with high probability, when $Z^* \geq 9/\varepsilon^2 \ln n$.

Proof. For each cell c , let D_c be the density of c in the solution returned by the algorithm. Define random variables $X_{i,\lambda}$, where $X_{i,\lambda} = 1$ if rectangle r_i is routed toward direction λ by the algorithm, and $X_{i,\lambda} = 0$ otherwise. Then, we have $D_c = \sum_{(i,\lambda) \in P_c} X_{i,\lambda}$. Therefore,

$$\begin{aligned} E[D_c] &= \sum_{(i,\lambda) \in P_c} E[X_{i,\lambda}] \\ &= \sum_{(i,\lambda) \in P_c} \Pr\{X_{i,\lambda} = 1\} \\ &= \sum_{(i,\lambda) \in P_c} x_{i,\lambda}^* \quad (\text{by line 2 of algorithm}) \\ &\leq Z^*. \quad (\text{by LP constraint}) \end{aligned}$$

Moreover, for each cell c , the variables $X_{i,\lambda}$ for all $(i, \lambda) \in P_c$ are independent. To see this, notice that there are two types of variables contributing to the density of c . If c is contained in a rectangle r_i , then $X_{i,\lambda}$, for all directions λ , pass through c . In this case, we can replace these four variables in the constraint of c by just a number 1, since one and exactly one of these variables will be 1 in any optimal solution of LP. If c is not contained in r_i , then (i, λ) contributes to the density of c for at most one value of λ , since no two directions of r_i can pass through c simultaneously. Therefore, after substituting the first type of variables in the constraint of cell c by 1, all other variables $X_{i,\lambda}$ for all $(i, \lambda) \in P_c$ are independent, due to the fact that the direction of rectangles are chosen independently.

We can now use Chernoff bound to show that D_c is close to Z^* with high probability. We use the following statement of Chernoff bound: If X_1, \dots, X_n are independent 0-1 random variables, $X = \sum X_i$, $E[X] \leq U$, and $0 \leq \varepsilon \leq 1$, then $\Pr\{X \geq (1 + \varepsilon)U\} \leq e^{-U\varepsilon^2/3}$. Since $E[D_c] \leq Z^*$, by Chernoff bound we have

$$\Pr\{D_c \geq (1 + \varepsilon)Z^*\} \leq e^{-Z^*\varepsilon^2/3}.$$

The solution produced by our algorithm has density $\max_c\{D_c\}$. Since there are at most $(2n)^2$ grid cells, assuming $Z^* \geq c_\varepsilon \ln n$ for some constant $c_\varepsilon > 0$, we get

$$\begin{aligned} \Pr\{\max_c\{D_c\} \geq (1 + \varepsilon)Z^*\} &\leq \sum_c \Pr\{D_c \geq (1 + \varepsilon)Z^*\} \\ &\leq (2n)^2 \times n^{-c_\varepsilon \varepsilon^2/3} \\ &= 4n^{2-(c_\varepsilon \varepsilon^2/3)}. \end{aligned}$$

Therefore, for a proper constant $c_\varepsilon \geq 9/\varepsilon^2$, the probability that the solution returned by our algorithm is greater than $(1 + \varepsilon)Z^*$ is at most $\frac{4}{n}$. Taking into account that $Z^* \leq \text{OPT}$, it shows that our algorithm has an approximation factor of $1 + \varepsilon$ with high probability if $Z^* \geq c_\varepsilon \ln n$. \square

5 Conclusions

In this paper, we presented some new results on the rectangle escape problem. In particular, we presented a lower bound of $3/2$ on the approximability of the problem, and a $(1 + \varepsilon)$ -approximation algorithm for the problem when the optimal density is high enough. It remains open what the best approximation factor is for the problem in general case.

Acknowledgments The authors would like to thank Hesam Monfared for suggesting the rectangle escape problem, and the anonymous referees for their helpful comments.

References

- [1] H. Kong, Q. Ma, T. Yan, and M. D. F. Wong. An optimal algorithm for finding disjoint rectangles and its application to PCB routing. In *Proc. 47th ACM/EDAC/IEEE Design Automation Conf.*, DAC '10, pages 212–217, 2010.
- [2] H. Kong, T. Yan, and M. D. F. Wong. Automatic bus planner for dense PCBs. In *Proc. 46th ACM/EDAC/IEEE Design Automation Conf.*, DAC '09, pages 326–331, 2009.
- [3] Q. Ma, H. Kong, M. D. F. Wong, and E. F. Y. Young. A provably good approximation algorithm for rectangle escape problem with application to PCB routing. In *Proc. 16th Asia South Pacific Design Automation Conf.*, ASPDAC '11, pages 843–848, 2011.
- [4] Q. Ma, E. Young, and M. D. F. Wong. An optimal algorithm for layer assignment of bus escape routing on PCBs. In *Proc. 48th ACM/EDAC/IEEE Design Automation Conf.*, pages 176–181, 2011.
- [5] M. M. Ozdal, M. D. F. Wong, and P. S. Honsinger. An escape routing framework for dense boards with high-speed design constraints. In *Proc. 2005 IEEE/ACM Internat. Conf. Computer-Aided Design*, ICCAD '05, pages 759–766, 2005.
- [6] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
- [7] P.-C. Wu, Q. Ma, and M. D. Wong. An ILP-based automatic bus planner for dense PCBs. In *Proc. 18th Asia South Pacific Design Automation Conf.*, ASPDAC '13, pages 181–186, 2013.
- [8] J. T. Yan and Z. W. Chen. Direction-constrained layer assignment for rectangle escape routing. In *Proc. 2012 IEEE Internat. System-on-Chip Conf.*, SOCC '12, pages 254–259, 2012.
- [9] J. T. Yan, J. M. Chung, and Z. W. Chen. Density-reduction-oriented layer assignment for rectangle escape routing. In *Proc. Great Lakes Sympos. VLSI*, GLSVLSI '12, pages 275–278, 2012.
- [10] T. Yan, H. Kong, and M. D. F. Wong. Optimal layer assignment for escape routing of buses. In *Proc. 2009 IEEE/ACM Internat. Conf. Computer-Aided Design*, ICCAD '09, pages 245–248, 2009.

Grid Proximity Graphs: LOGs, GIGs and GIRLs

River Allen*

Laurie Heyer†

Rahnuma Islam Nishat*

Sue Whitesides*

Abstract

This paper discusses three types of proximity graphs called LOGs, GIGs and GIRLs, defined on unit grids. We show that it can be decided in linear time whether a LOG graph is a GIG graph. We also show that it is NP-complete to recognize LOGs and GIGs, and explore the relationship between these graph classes and their properties. Enumeration results and open problems are also presented.

1 Introduction

Consider an $m \times n$ unit grid and define on this grid a *limited outdegree grid* directed graph, or LOG graph, as follows: the vertices are the mn vertices of the unit grid, the underlying edges are a subset of the unit grid edges such that each edge has unit length and each vertex has outdegree at most one. In other words, each vertex can point to at most one of its neighbors in the underlying grid. See Figure 1 for an example of a 3×4 LOG graph.

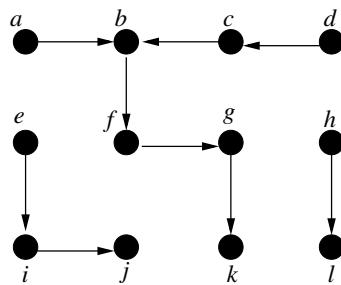


Figure 1: A LOG graph with 12 vertices.

Let $G = (V, E)$ be an $m \times n$ LOG graph with vertex set V and edge set E . For each vertex u in V , let $N(u)$ denote the set of the vertices of G that have unit distance from u in the underlying grid. We call $N(u)$ the *potential neighbors* of u .

One way to obtain a LOG graph is to make a one-to-one assignment of the labels $1, 2, \dots, mn$ to the mn vertices of the grid, then include a directed edge (u, v) if the label at v is greater than the label of u and the

*Department of Computer Science, University of Victoria, BC, Canada, riverallen@gmail.com, rnishat@uvic.ca, sue@uvic.ca

†Mathematics Department, Davidson College, NC, USA, lahey@davidson.edu

greatest among all the labels of $N(u)$. This construction motivates the following definition, where we denote the label of vertex u by $L(u)$.

Definition 1 A Greatest Increase Grid *directed graph* or a *GIG graph* is a LOG graph in which the vertices can be labeled with distinct integers $1, 2, \dots, mn$ such that $(u, v) \in E$ if and only if $v \in N(u)$, $L(v) > L(u)$ and $L(v) > L(w)$ for all $w \in N(u)$, $w \neq v$.

See Figure 2 for an example construction of a 3×3 GIG graph. In [2], a GIG graph is interpreted as a representation of a discrete 3-dimensional search space in which the vertices of G are the states, and $L(u)$ is the utility of the state. A hill-climbing algorithm with initial state u would succeed in finding the global maximum state if and only if there is a directed path from u to that state.

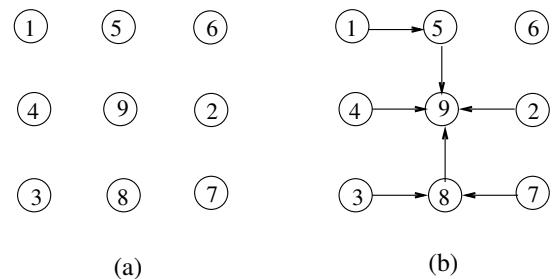


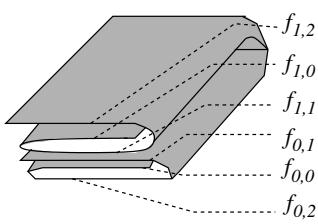
Figure 2: (a) A random labeling of the vertices of a 3×3 grid, and (b) a GIG graph generated from the labeling.

In this paper, we present an alternative interpretation of a GIG as a representation of a folded map. An $m \times n$ map is a rectangular piece of paper that is divided into mn unit squares by a $m \times n$ square grid on the paper. The edges of the grid on the paper (not on the boundary of the paper) are called *creases*. A map can be folded only along the creases. Figure 3(a) shows a 2×3 map and it has seven creases, three horizontal and four vertical.

A famous open problem in map folding posed by Jack Edmonds asks whether it can be decided in polynomial time whether a map can be folded into a unit square or not [4]. Suppose that a map can be folded into a unit square. Then in such a folded state, there is a linear ordering of the faces of the map from top to bottom [6] as shown in Figure 3(b).

$f_{0,0}$	$f_{0,1}$	$f_{0,2}$
$f_{1,0}$	$f_{1,1}$	$f_{1,2}$

(a)



(b)

Figure 3: (a) A 2×3 map with the faces labeled, (b) the map folded on a unit square.

We represent each face of the map as a fixed vertex of a GIG G depending on the row and column of the face as shown in Figure 4(a). We then label the vertices of the GIG G as shown in Figure 4(b), where each vertex of G gets a label depending on its height from the plane on which the faces are stacked, and add the directed edges according to the definition of a GIG graph. A directed edge (u, v) in this graph G denotes that u is below v in the stack of faces and v is the topmost among all the neighbors of u .

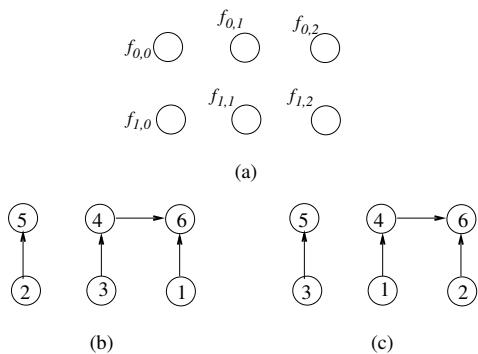


Figure 4: (a) The faces of the map in Figure 3(a) shown as vertices of G , (b) labeling of the vertices G according to the ordering in Figure 3(b), (c) another labeling of the same GIG G which causes the paper to self-intersect.

Now, suppose a GIG G is given that represents a possible linear ordering of the faces of a map. Depending on the labeling of the vertices of G , we might or might not get a linear ordering of the faces that is valid. Figures 4(b) and (c) show two different labelings of the same GIG associated with a 2×3 map. Although the labeling of Figure 4(b) gives the valid linear ordering for the folded state in Figure 3(b), the ordering from Figure 4(c) cannot be obtained.

Figure 5(a) and (b) show a 2×4 map and a GIG representing a possible linear ordering of its faces. In this GIG, the faces $f_{0,2}$ and $f_{1,3}$ must receive the labels 7 and 8, respectively. For this reason, any labeling of this GIG gives an ordering of the faces of the map such that the paper would have to self-intersect.

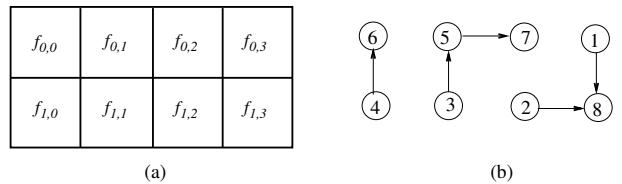


Figure 5: (a) A 2×4 map with the faces labeled, (b) a GIG that gives no linear ordering.

The rest of the paper is organized as follows. Section 2 gives an algorithm to decide whether a given LOG graph is a GIG graph. Section 3 presents an algorithm to generate all possible labelings of the vertices of a GIG graph. In Section 4, we show that it is NP-complete to decide whether a graph is a LOG or a GIG. Section 5 and 6 give generalizations and variations. Section 7 concludes the paper.

2 Recognizing GIG graphs

Since the set of GIG graphs on an $m \times n$ grid is a proper subset of the set of LOG graphs on the same size grid (e.g., edge-free LOG graphs are not GIG graphs), we are interested in deciding whether a given LOG graph is a GIG graph. Here, we give a polynomial time algorithm to solve this decision problem. The algorithm rests on the construction of a new graph that represents a set of inequalities implied by the edges in G . We define this new graph as follows:

Definition 2 *The augmented graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of a LOG graph $G = (V, E)$ is a directed graph that satisfies the following conditions.*

- (a) $\mathcal{V} = V$ and $E \subset \mathcal{E}$.
- (b) If there is an edge from u to v in G , then \mathcal{G} also has edges from all other potential neighbors of u to v . In other words, $(w, v) \in \mathcal{E}$ for each $w \in N(u), w \neq v$.
- (c) If the outdegree of u is 0 in G , then for every $w \in N(u)$, there must be an edge (w, u) in \mathcal{E} .

Figure 6 shows an example of the augmented graph of GIG graph on a 3×3 grid.

If a \mathcal{G} is an augmented graph of a LOG graph that is a GIG graph, then \mathcal{G} can be reconstructed from the labeled grid vertices of the GIG. Thus the labeled vertices provide a geometric and compressed representation of \mathcal{G} .

Theorem 3 *Let G be a LOG graph with mn vertices and let \mathcal{G} be the augmented graph of G . Then G is a GIG graph if and only if \mathcal{G} is acyclic.*

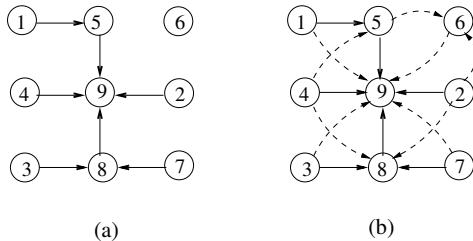


Figure 6: (a) A GIG graph on a 3×3 grid, and (b) its augmented graph.

Proof. If G is a GIG graph then there exists a labeling of the vertices of G such that every vertex points to its biggest potential neighbor that has a bigger label than the vertex itself in the grid embedding of G . Then by the definition of augmented graphs, each directed edge (u, v) in \mathcal{G} denotes that $v > u$. Suppose there is a directed cycle $v_1, v_2, \dots, v_k, v_1$ in \mathcal{G} . Then we get $v_1 < v_2 < \dots < v_k < v_1$, which is a contradiction. Therefore, \mathcal{G} is acyclic.

We now assume that the augmented graph \mathcal{G} is acyclic. Then we can get a topological sort [3] of the vertices of \mathcal{G} and we assign the resulting labels $1, 2, \dots, mn$ to the vertices of \mathcal{G} . We now show that any of these labelings satisfies the definition of GIG graphs. Let u be a vertex in G . We have to consider two cases:

- (a) G contains an outgoing edge (u, v) in G . Then in the augmented graph \mathcal{G} , we have an edge (w, v) for each $w \in N(u)$, where $w \neq v$. Thus, all the potential neighbors of u receive smaller index than v .
- (b) There is no outgoing edge from u in G . Then all its potential neighbors points to it in \mathcal{G} and therefore they all receive smaller index than v .

Therefore, the labeling of G obtained above satisfies the definition of GIGs and hence G is a GIG graph. \square

3 Generating all the Labelings

In this section, we give an algorithm to generate all possible labelings of the vertices of a GIG graph since a GIG graph can have multiple labelings (see Figure 7).

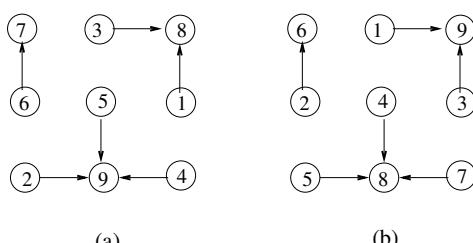


Figure 7: Two labelings of the same GIG graph.

Pruesse and Ruskey [7] gave an output sensitive algorithm to generate all possible linear extensions of a given poset. Let \mathcal{P} be a poset and let $\mathcal{E}(\mathcal{P})$ be the set of all linear extensions of \mathcal{P} . Then their algorithm generates all linear extensions in time $O(|\mathcal{E}(\mathcal{P})|)$, which results in constant amortized time. We can use this algorithm to generate all possible labelings of a given GIG graph in constant amortized time. Here we give a sketch of a simpler recursive algorithm which suffices for our purpose.

Let G be the given GIG graph with n vertices and let \mathcal{G} be the augmented graph of G . Then \mathcal{G} must be a directed acyclic graph. Since there are no cycles in G , there must be at least one vertex in G that has no incoming edges. We denote by *sources* the vertices with no incoming edges in G . We now label any of the sources with the least available index from $1, \dots, n$. Let the source be v and label of the source be $l(v) = i$. We then remove v from G and recurse the procedure for $G \setminus v$, where the least available index is $i + 1$. A pseudocode of our algorithm **LabelGIG** is given below. The initial call is **LabelGIG**($G, 1$).

Algorithm 1: LabelGIG(G, i)

```

1  $S$  is the set of all the sources in  $G$ 
2 if  $G = \emptyset$  then
3   print the labeling.
4   return.
5 for each  $v \in S$  do
6    $l(v) = i$ 
7   LabelGIG( $G \setminus v, i + 1$ )

```

4 Complexity of Embedding a GIG on a Grid

In this section, we show that it is NP-complete to determine whether a given abstract graph is a LOG graph. We also show that the recognition problem remains NP-complete for GIGs.

A formal definition of the LOG recognition problem is given below.

Problem : LOG-RECOG

Instance : Two integers $m, n > 0$ and a planar directed graph G with mn vertices such that the maximum degree of G is less than or equal to four and each vertex has outdegree less than or equal to one.

Question : Does G have a plane rectilinear embedding on an $m \times n$ integer grid?

We reduce the 3-PARTITION problem to LOG-RECOG to prove the NP-hardness. The 3-PARTITION problem is described as follows.

Problem : 3-PARTITION

Instance : A set of integers $S = \{x_1, x_2, \dots, x_{3p}\}$, where $p > 0$, and an integer $B > 0$ such that $\sum_{i=1}^{3p} x_i = pB$ and $B/2 > x_i > B/4$, $1 \leq i \leq 3p$.

Question : Can S be partitioned into p disjoint sets such that each set contains exactly three integers that sum up to B ?

We use a very similar construction as Dolev *et al.* [5]. We create a *directed frame tree* as shown in Figure 8(b) from the frame tree of Dolev *et al.* shown in Figure 8(a). We choose the degree four vertex r as the root and direct each edge from the child node to the parent node.

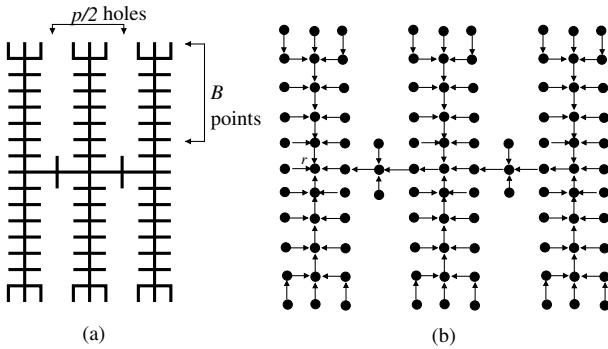


Figure 8: (a) The frame tree used by Dolev *et al.* [5], and (b) the corresponding directed frame tree.

We now prove the NP-completeness of LOG-RECOG.

Theorem 4 LOG-RECOG is NP-complete.

Proof. If a rectilinear embedding of G is given on an $m \times n$ grid, it can be checked in polynomial time whether each edge has unit length. Thus the problem is in NP. To prove that it is NP-hard, take an instance $S = \{x_1, x_2, \dots, x_{3p}\}$ of 3-PARTITION, where $\sum_{i=1}^{3p} x_i = pB$ and $B/2 > x_i > B/4$, $1 \leq i \leq 3p$, and construct an instance of LOG-RECOG from the instance of 3-PARTITION as follows.

First assume that p is even. Create a directed frame tree T with $(2p+3)(2B+3) - pB$ vertices as shown in Figure 8(b). Then create a *directed path* of x_i vertices for each of the integers $x_i \in S$, $1 \leq i \leq 3p$. The graph G containing T and all the $3p$ directed paths has $(2p+3)(2B+3)$ vertices. Finally, choose $m = 2p+3$ and $n = 2B+3$. We now show that the given instance of the 3-PARTITION problem has a solution if and only if G has a rectilinear embedding on an $m \times n$ integer grid.

First assume that the instance of 3-PARTITION has a solution, so S can be partitioned into p disjoint sets S_1, S_2, \dots, S_p such that each of the sets has exactly three integers that sum to B . From Lemma 6 of [5], T has only two possible embeddings on an $m \times n$ grid,

and in each of the cases there are p holes of B grid points each. Therefore, for each S_j , $1 \leq j \leq p$, lay the paths corresponding to the integers in S_j in one hole and get a rectilinear embedding of G .

Now assume that G has an embedding on the integer grid. Since any embedding of T leaves p holes of B grid points and the number of grid points is equal to the number of vertices in G , each of the holes must contain three paths that have B vertices in total. Take the integers corresponding to the paths in a hole to form a subset. In this way we get a partition of S into p disjoint subsets as required.

Now assume that p is odd. Create a directed frame tree T with $(2(p+1)+3)(2B+3) - (p+1)B$ vertices and the $3p$ directed paths representing $3p$ integers as before. We also create a directed path of B vertices as shown in Figure 9. In this case G contains $(2p+5)(2B+3)$ vertices and hence, we take $m = 2p+5$ and $n = 2B+3$. As in the previous case, it can be proved that the given instance of 3-PARTITION has a solution if and only if G has a rectilinear embedding on an $m \times n$ integer grid. \square

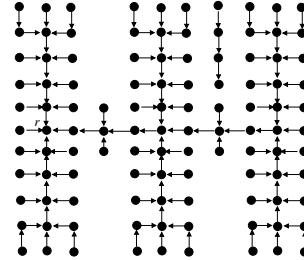


Figure 9: The directed frame tree and the directed path of B vertices when p is odd.

We now define the GIG recognition problem.

Problem : GIG-RECOG

Instance : Two integers $m, n > 0$ and a planar directed graph G with mn vertices such that the maximum degree of G is less than or equal to four and each vertex has outdegree less than or equal to one.

Question : Does G have a plane rectilinear embedding on an $m \times n$ integer grid such that the augmented graph is a directed acyclic graph?

Now we prove that GIG-RECOG is NP-complete.

Theorem 5 GIG-RECOG is NP-complete.

Proof. If a rectilinear embedding of G is given on an $m \times n$ grid, it can be checked in polynomial time whether each edge has unit length and whether the augmented graph is a directed acyclic graph. Thus the problem is in NP. To prove that it is NP-hard, reduce the 3-PARTITION problem to GIG-RECOG as in the proof of Theorem 4.

Create a directed frame tree T with $3p(B+3) + 4(p-1) + 12 \times 2 = 3pB + 13p + 20$ vertices as shown in Figure 10. Choose $m = 4$ and $n = p(B+3) + p - 1 + 3 \times 2 = pB + 4p + 5$. On an $m \times n$ grid, T has the unique embedding shown in Figure 10, which creates p holes with $B+3$ grid points in each. Then create a *directed path* of x_i+1 vertices for each of the integers $x_i \in S$, $1 \leq i \leq 3p$. The graph G containing T and all the $3p$ directed paths has $3pB + 13p + 20 + pB + 3p = 4(pB + 4p + 5)$ vertices in total. As in the proof of Theorem 4, it can be shown that the given instance of 3-PARTITION has a solution if and only if G has a rectilinear embedding on the $m \times n$ integer grid, where each edge on the directed paths points right to left. We now show that the augmented graph obtained from such an embedding of G is acyclic.

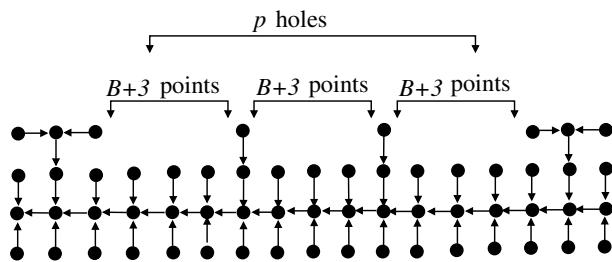


Figure 10: The frame tree for the proof of Theorem 5.

Figure 11(a) shows the augmented graph of the left-most portion of the directed frame tree T with just one hole, and Figure 11(b) shows the augmented graph when the directed paths representing the integers in a subset of S are laid out such that each edge on the directed paths points right to left. Because of the symmetric structure of the augmented graph, it is easy to see that the augmented graph in Figure 11(b) is acyclic and hence, the augmented graph of such a rectilinear embedding of G on the grid is an acyclic directed graph. \square

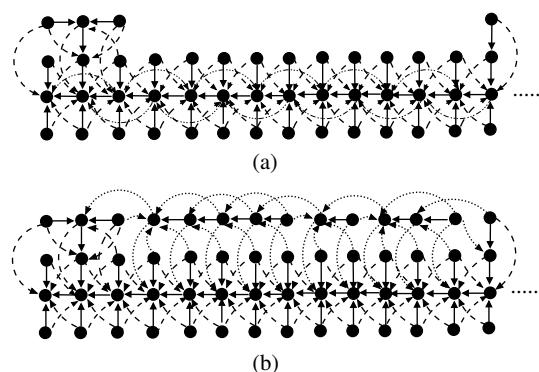


Figure 11: Illustration for the proof of Theorem 5. The dot dashed lines in (a) are not shown in (b).

5 Generalizations of LOGs and GIGs

In this section, we show that the concepts of LOG graphs and GIG graphs extend to \mathbb{R}^d , where $d \geq 2$, and also to other kinds of grids than rectangular grids.

Figure 12(a) and (b) show a LOG and a GIG in \mathbb{R}^3 . Each vertex here has at most 6 potential neighbors and

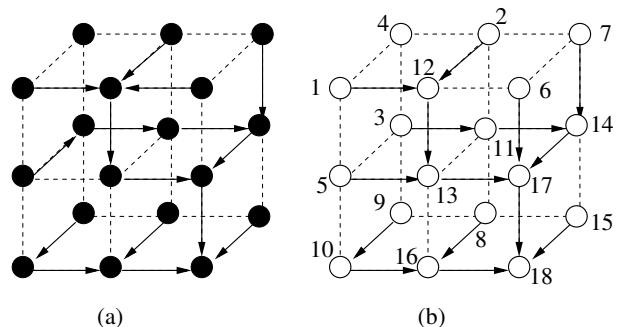


Figure 12: (a) A $3 \times 3 \times 2$ LOG, (b) a $3 \times 3 \times 2$ GIG.

outdegree at most one. Therefore, in \mathbb{R}^d , each vertex has at most $2d$ potential neighbors. In other words, if we have a directed graph with outdegree at most one and maximum degree Δ , a necessary condition for it to be represented as a LOG in \mathbb{R}^d is that $d \geq \Delta/2$. Figure 13(a) and (b) show examples of LOG graphs on a triangular grid and a hexagonal grid in \mathbb{R}^2 , respectively. Each vertex of a LOG has at most 6 and 3 potential

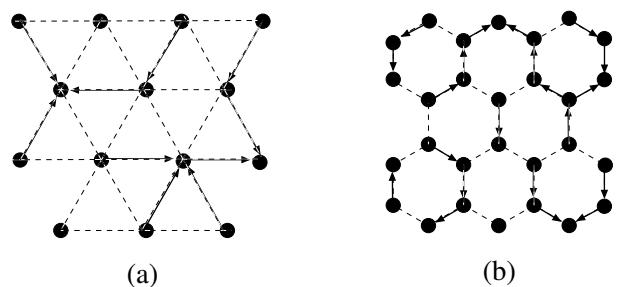


Figure 13: LOGs on (a) a triangular grid, and (b) a hexagonal grid.

neighbors on a triangular and on a hexagonal grid, respectively. This observation raises the following open problem.

Open Problem: Let G be a directed graph with outdegree at most one, maximum degree 6 and lmn vertices. What is the complexity of determining whether it is a LOG graph on a rectangular ($l \times m \times n$) unit grid in \mathbb{R}^3 or a LOG graph on a triangular grid in \mathbb{R}^2 ?

6 Labeling LOGs with Repetition

In this section, we introduce another subclass of LOG graphs: the *Greatest Increase with Repeated Labels Al-*

lowed Grid directed graphs, or GIRL graphs for short.

GIRL graphs are a modification of GIG graphs where we allow repeated use of labels, but at each vertex the labels of its neighbors must be distinct; the label of a vertex may appear among the labels of its neighbors.

Definition 6 A Greatest Increase with Repeated Labels Allowed Grid directed graph is a LOG graph in which the vertices can be labeled with integers $1, 2, \dots, mn$ such that the directed edge $(u, v) \in E$ if and only if $v \in N(u)$, $L(v) > L(u)$ and $L(v) = \max\{L(w) | w \in N(u)\}$; furthermore $\forall v \in V, \|N(v)\| = \|\{L(w) | w \in N(v)\}\|$, i.e. for each vertex the labels of its neighbors must be distinct.

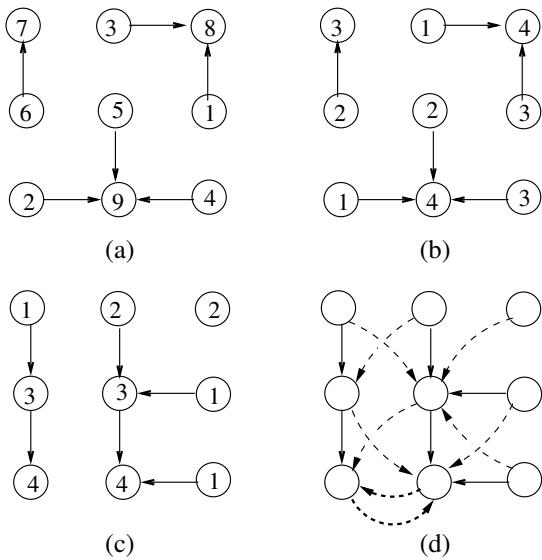


Figure 14: (a) A GIG graph, where each vertex has a unique label, (b) an equivalent GIRL representation, (c) A GIRL graph that is not a GIG graph and (d) the augmented graph of the underlying LOG graph for (c) which has a directed cycle shown in bold dashed lines.

Note that, since we can determine if a LOG graph is a GIG graph, we can determine if a GIRL graph is a GIG graph. For example, the GIRL graph in Figure 14(c) cannot be a GIG graph because its augmented graph contains a directed cycle as shown in Figure 14(d). Also note that there are LOG graphs that are not GIRL graphs: for example, a 3×3 LOG graph without any edges is not a GIRL graph. Thus the inclusions in Figure 15 are strict.

The possibility of repeated labels leads to several questions:

- Given a LOG graph, is it a GIRL graph?

It is easy to check that Figure 14(b) uses a minimum label set.

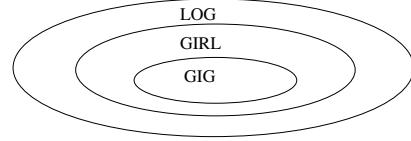


Figure 15: The inclusions are strict.

- Given a GIG graph, what is the minimum set of labels needed to represent it as a GIRL graph?

7 Conclusion

We have studied the LOG graphs and GIG graphs, introduced the augmented graphs of a GIG graph and characterized GIG graphs in terms of their augmented graphs. We show that LOG graph and GIG graph recognition is NP-complete. We have also introduced generalizations of LOG graphs and a significant superclass of GIG graphs called the GIRL graphs. We close with the following question, similar in spirit to graph decomposition problem such as linear arboricity [1].

Open problem: Given any directed acyclic graph G , can we decompose G into (a minimum number of) augmented graphs of GIG or GIRL graphs?

References

- [1] Noga Alon. The linear arboricity of graphs. *Israel Journal of Mathematics*, 62(3):311–325, 1988.
- [2] Joshua Chester, Linnea Edlin, Jonah Galeota-Sprung, Bradley Isom, Andrew Lantz, Alexander Moore, Virginia Perkins, E. Tucker Whitesides, A. Malcolm Campbell, Todd T. Eckdahl, Laurie J. Heyer, and Jeffrey L. Poet. On counting limited outdegree grid digraphs and greatest increase grid digraphs. Unpublished manuscript, 2012.
- [3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms* (3. ed.). MIT Press, 2009.
- [4] Erik D. Demaine and Joseph O’Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, NY, USA, 2007.
- [5] Danny Dolev, Tom Leighton, and Howard Trickey. Planar embedding of planar graphs. In *Advances in Computing Research*, pages 147–161, 1984.
- [6] Rahnuma Islam Nishat. Map folding. Master’s thesis, University of Victoria, BC, Canada, April 2013.
- [7] Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994.

Bounding the Locus of the Center of Mass for a Part with Shape Variation

Fateme Panahi

A. Frank van der Stappen *

Abstract

The shape and center of mass of a part are crucial parameters to algorithms for planning automated manufacturing tasks. As industrial parts are generally manufactured to tolerances, the shape is subject to variations, which, in turn, also cause variations in the location of the center of mass. Planning algorithms should take into account both types of variation to prevent failure when the resulting plans are applied to manufactured incarnations of a model part.

We study the relation between variation in part shape and variation in the location of the center of mass for a convex part with uniform mass distribution. We consider a general model for shape variation that only assumes that every valid instance contains a polygon P_I while it is contained in another polygon P_E . We characterize the worst-case displacement of the center of mass in a given direction in terms of P_I and P_E . The characterization allows us to determine an adequate polygonal approximation of the locus of the center of mass. We also show that the worst-case displacement is small if P_I is fat and the distance between the boundary of P_E and P_I is bounded.

1 Introduction

Many automated part manufacturing tasks involve manipulators that perform physical actions—such as pushing, squeezing [1], or pulling [2]—on the parts. Over the past two decades, researchers in robotics in general and algorithmic automation in particular have thoroughly studied the effect of physical actions as well as their potential role in accomplishing high-level tasks like orienting or sorting. It is evident that shape and—in many cases (see e.g. [1, 3, 4, 5, 6, 7])—location of the center of mass are important parameters in determining the effect of a physical action on a part.

Industrial parts are always manufactured to tolerances as no production process is capable of delivering parts that are perfectly identical. Tolerance models [12, 13] are therefore used to specify the admitted variations with respect to the CAD model. A consequence of

these variations [8, 9] is that actions that are computed on the basis of a CAD model of a part may easily lead to different behavior when executed on a manufactured incarnation of that part, and thus to failure to accomplish the higher-level task. It is important to note that the shape variations not only directly affect the behavior of the part but indirectly as well because they also cause a displacement of the center of mass of the part.

To extend the planning algorithms to imperfect manufactured incarnations, it is important to understand the effects of variations and take them into account during planning. Larger variations in part shape and center-of-mass location inevitably result in a larger range of possible part behaviors, which reduces the likeliness that a manufacturing task can be accomplished. Therefore we will study how variations in part shape influence the location of the center of mass. (Note that variations in shape and center of mass are not the only sources of uncertainty in robotics. Additional uncertainty can result from the inaccuracy of the actuators and manipulators [11] and sensors [10].)

Several geometric approaches have been proposed to overcome the problems occurring in the presence of uncertainty and to smooth the effects of errors. Among the existing approaches are the model of ϵ -geometry [14], tolerance and interval geometry [15, 16] and region-based models [17]. Generally, in all these models an uncertain point is represented by a region in which it may vary. The model of ϵ -geometry assumes that a point can vary within a disk of radius ϵ . Tolerance and interval-geometry take into account coordinate errors which results in an axis-aligned rectangular region in which a point can vary. In general, region-based models represent a point by any convex region. After modeling uncertainty as a point surrounded by a region, it is possible to study worst (and best) cases for a problem under the specific uncertainty model.

As observed before, variation of the shape causes variation of the center of mass of a part. The locus of the centroid of a set of points with approximate weights has been studied by Bern et al. [19]. Akella et al. [18] estimated the locus for a polygon under the ϵ -geometry model [18]. The problem of finding the locus of the center of mass of a part with shape variation and uniformly distributed mass has been mentioned as an open problem [9, 18]. Akella et al. [18] studied rotating a convex polygon whose vertices and the center of mass lie inside predefined circles centered at their nominal loca-

*Department of Information and Computing Sciences, Utrecht University, PO Box 80089, 3508 TB Utrecht, The Netherlands, email:{F.Panahi,A.F.vanderStappen}@uu.nl F. Panahi is supported by the Netherlands Organization for Scientific Research (NWO).

tions. The problem of orienting a part by fence has been studied by Chen et al. [9]. They define disk and square regions for the vertices of a part and proposed a method for computing the maximum allowable uncertainty radius for each vertex. They also discussed in a more general way the key role of the center of mass and the successfullness of part feeding (or orienting) algorithms in a setting of shape variation. Chen et al. [20] presented algorithms for squeezing and pushing problems. Kehoe et al. [21] explored cloud computing in a context of grasping and push-grasping under shape variation.

All the previous models for shape variation only allow the vertices to vary. In this paper we use a more general model for shape variation. For given convex shapes P_I and P_E such that $P_I \subseteq P_E$ we consider the family of shapes P satisfying $P_I \subseteq P \subseteq P_E$. In the practical setting of toleranced parts the shapes P_I and P_E will be fairly similar. We will show in Section 3 that the valid instance that yields the largest displacement of the center of mass in a given direction is a shape that combines a part of P_I with a part of P_E . The corresponding displacement is computable in $O(n)$ time where n is the complexity of P_I and P_E ; it can be used to obtain a k -vertex outer approximation of the set of all possible loci of the center of mass in $O(kn)$ time.

In Section 4, we will study the size of the set of possible center-of-mass loci. Fatness of the objects under consideration has led to lower combinatorial complexities and more efficient algorithms for various problems, including union complexities [24], motion planning [22], hidden surface removal [25], and range searching [26]. We show that fatness of P_I together with the assumption that no point in P_E has a distance larger than ϵ to P_I leads to a bound on the distance between the centers of mass of any two valid instances of a part which is proportional to ϵ and the fatness of P_I .

2 Preliminaries

In this section, we first present a general model for shape variations, then review the notion of a center of mass, and finally introduce a few notions that allow us to characterize the shapes that maximize the displacement of the center of mass. Let P_M be the model part. The part P_M has a uniform mass distribution.

No production process ever delivers parts that are perfectly identical to the model part P_M and therefore industrial parts are manufactured to tolerances. We use a very general model for permitted shape variations that only requires that any manufactured instance of P_M contains a given convex subshape P_I of P_M while it is contained in a convex supershape P_E of P_M . As a result, the set of acceptable instances of P_M is a family of shapes $V(P_I, P_E) = \{P \in \mathbb{R}^2 | P_I \subseteq P \subseteq P_E\}$ for given P_I and P_E satisfying $P_I \subseteq P_M \subseteq P_E$. In other

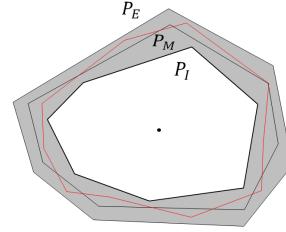


Figure 1: A family of shapes specified by a subshape P_I and a supershape P_E of a model part P_M , along with a valid instance $P \in V(P_I, P_E)$.

words, the boundary ∂P of an instance $P \in V(P_I, P_E)$ should be entirely contained in $Q = P_E - \text{int}(P_I)$ where $\text{int}(P)$ denotes the interior of the set P . The region Q is referred to as the *tolerance zone*. The objects P_I and P_E are assumed to be closed polygons with a total of n vertices. (Figure 1 shows and example of a model part P_M , shapes P_I and P_E , and a valid instance $P \in V(P_I, P_E)$.) We denote by $\text{COM}(P_I, P_E)$ the set of all centers of mass of instances $P \in V(P_I, P_E)$.

We let $X_c(P)$ denote the x -coordinate of the center of mass and $A(P)$ be the area of the object P . The x -coordinate of the center of mass of an object with uniform mass distribution satisfies

$$X_c(P) = \frac{1}{A} \int_A x dA,$$

where A is the area of the object. A similar equality holds for the y -coordinate of the center of mass. In the case of uniform mass distribution the center of mass corresponds to the centroid of the object. We will often decompose an object P into sub-objects P_i ($1 \leq i \leq n$) and then express its center of mass as a function of the centers of mass of its constituents, through the equation

$$X_c(P) = \frac{\sum_{i=1}^n X_c(P_i) A(P_i)}{\sum_{i=1}^n A(P_i)}. \quad (1)$$

We conclude this section by defining useful objects. Disks play a prominent role in Section 4. We denote by $D_r(p)$ the closed disk with radius r centered at p , and use the abbreviation $D_r = D_r(O)$ where O is the origin.

For an object P and a value m we define its right portion $P^+[m]$ with respect to m by $P^+[m] = \{(x, y) \in P | x \geq m\}$. Similarly, we define its left portion $P^-[m]$ with respect to m by $P^-[m] = \{(x, y) \in P | x \leq m\}$. With these portions we can define *minmax* objects, which allow us to capture the intuition that the largest displacement of the center of mass in a given direction is achieved by the object from $V(P_I, P_E)$ that ‘maximizes mass’ in that direction and ‘minimizes mass’ in the opposite direction. The minmax object $P^*[m]$ consists of a left portion of P_I and a right portion of P_E with respect to the same m , so $P^*[m] = P_I^-[m] \cup P_E^+[m]$

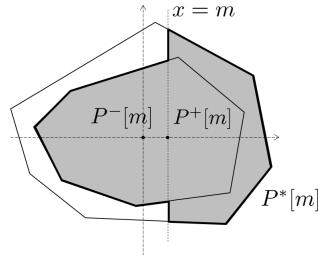


Figure 2: A minmax object

(see Figure 2). Note that an alternative way to describe $P^*[m]$ is by the equation $P^*[m] = P_I \cup Q^+[m]$

3 Displacement of the center of mass

In this section, we find an upper bound on the displacement of the center of mass in a given direction. The resulting bound allows us to determine a good polygonal outer approximation of the set $COM(P_I, P_E)$ of possible loci of the center of mass.

3.1 Bounding the displacement in one direction

Without loss of generality we assume that P_I and P_E are positioned and oriented in such a way that the center of mass of P_I coincides with the origin (so $X_c(P_I) = 0$) and that the direction in which we want to bound the displacement aligns with the positive x -axis. This assumption and the equation $P^*[m] = P_I \cup Q^+[m]$ allow us to simplify Equation 1 to

$$X_c(P^*[m]) = \frac{X_c(Q^+[m])A(Q^+[m])}{A(P_I) + A(Q^+[m])}. \quad (2)$$

Although we will bound the displacement with respect to the center of mass of P_I we observe that the result also induces a bound with respect to the center of mass of P_M as $P_M \in V(P_I, P_E)$ by definition. We let $X_r = \max_{(x,y) \in P_E} x$.

Our first lemma establishes a connection between the minmax objects $P^*[x]$ for $0 \leq x \leq X_r$ and the location of their centers of mass.

Lemma 1 *There is exactly one minmax object $P^*[m]$ ($0 \leq m \leq X_r$) that satisfies $X_c(P^*[m]) = m$. Moreover $x < X_c(P^*[x]) \leq m$ for all $0 \leq x < m$ and $X_c(P^*[x]) < m$ for all $m < x \leq X_r$.*

Proof. From $X_c(P_I) = 0$ and $X_c(Q^+[0]) \geq 0$ and the fact that $P^*[0] = P_I \cup Q^+[0]$ it follows that $X_c(P^*[0]) \geq 0$; moreover, it is clear that $X_c(P^*[X_r]) \leq X_r$. As the center of mass of $P^*[x]$ moves continuously as x increases from 0 to X_r there must be at least one x such that $X_c(P^*[x]) = x$. It remains to show that there is also at most one such x . Let m be such that

$X_c(P^*[m]) = m$. We consider a minmax object $P^*[x]$ for $x \neq m$ and distinguish two cases: (i) $0 \leq x < m$ and (ii) $m < x \leq X_r$.

Consider case (i). Using the notation $Q' = Q^+[m]$ and $Q'' = P^*[x] - P^*[m] = Q^+[x] - Q^+[m]$ we have that $P^*[m] = P_I \cup Q'$ and $P^*[x] = P_I \cup Q' \cup Q''$. Note that $Q'' \subset [x, m] \times \mathbb{R}$ and thus

$$x \leq X_c(Q'') \leq m.$$

As $x < X_c(P^*[m]) = X_c(P_I \cup Q') = m$ it follows from Equation 1 that

$$x(A(P_I) + A(Q')) < X_c(Q')A(Q') \leq m(A(P_I) + A(Q')).$$

After observing that $X_c(P_I) = 0$ we apply Equation 1 to $P^*[x] = P_I \cup Q' \cup Q''$ and use the aforementioned inequalities to obtain

$$\begin{aligned} X_c(P^*[x]) &= \frac{X_c(Q')A(Q') + X_c(Q'')A(Q'')}{A(P_I) + A(Q') + A(Q'')} \\ &> \frac{x(A(P_I) + A(Q')) + xA(Q'')}{A(P_I) + A(Q') + A(Q'')} = x \end{aligned}$$

and

$$\begin{aligned} X_c(P^*[x]) &= \frac{X_c(Q')A(Q') + X_c(Q'')A(Q'')}{A(P_I) + A(Q') + A(Q'')} \\ &\leq \frac{m(A(P_I) + A(Q')) + mA(Q'')}{A(P_I) + A(Q') + A(Q'')} = m. \end{aligned}$$

Consider case (ii). Using the notation $Q' = Q^+[x]$ and $Q'' = P^*[m] - P^*[x] = Q^+[m] - Q^+[x]$ we have that $P^*[x] = P_I \cup Q'$ and $P^*[m] = P_I \cup Q' \cup Q''$. Note that $Q'' \subset [m, x] \times \mathbb{R}$ and thus

$$m \leq X_c(Q'') \leq x.$$

As $X_c(P^*[m]) = X_c(P_I \cup Q' \cup Q'') = m$ it follows from Equation 1 that

$$X_c(Q')A(Q') = m(A(P_I) + A(Q')) + (m - X_c(Q''))A(Q'').$$

We apply Equation 1 to $P^*[x] = P_I \cup Q'$ and use the aforementioned equations and inequality to obtain

$$\begin{aligned} X_c(P^*[x]) &= \frac{X_c(Q')A(Q')}{A(P_I) + A(Q')} \\ &\leq \frac{m(A(P_I) + A(Q')) - (X_c(Q'') - m)}{A(P_I) + A(Q')} \\ &\leq \frac{m(A(P_I) + A(Q'))}{A(P_I) + A(Q')} = m. \end{aligned}$$

Combining both cases we find that there is no $x \neq m$ that satisfies $X_c(P^*[x]) = x$. \square

In addition to the fact that there is only one minmax object $P^*[m]$ that satisfies $X_c(P^*[m]) = m$, Lemma

1 also reveals that $X_c(P^*[x]) > x$ for $x < m$ and $X_c(P^*[x]) < x$ for $x > m$. Moreover, it shows that $X_c(P^*[x]) < m$ for all $x \neq m$ which means that the minmax object $P^*[m]$ with $X_c(P^*[m]) = m$ achieves larger displacement of the center of mass in the direction of the positive x -axis than any other minmax object $P^*[x]$ with $x \neq m$. The following theorem shows that $P^*[m]$ in fact achieves the largest displacement of the center of mass among *all* objects in $V(P_M)$.

Theorem 2 Let $P^*[m]$ ($0 \leq m \leq X_r$) be the unique minmax object that satisfies $X_c(P^*[m]) = m$. Then $X_c(P) < X_c(P^*[m])$ for all $P \in V(P_I, P_E), P \neq P^*[m]$.

Proof. Let $P \in V(P_I, P_E), P \neq P^*[m]$ be the object that yields the largest displacement $m' \geq m$ of the center of mass, so $X_c(P) = m'$. If $P = P^*[m']$ then it follows immediately from Lemma 1 that $m' = m$. Now assume for a contradiction that $P \neq P^*[m'] = P_I^-[m'] \cup P_E^+[m']$ which implies that (i) $P_E^+[m'] - P^+[m'] \neq \emptyset$ or (ii) $P^-[m'] - P_I^-[m'] \neq \emptyset$.

Consider case (i) and let R be a closed connected subset with $A(R) > 0$ of $P_E^+[m'] - P^+[m']$. Observe that $P \cup R \in V(P_I, P_E)$. Note that $R \subset (m', \infty) \times \mathbb{R}$ and thus $X_c(R) > m'$. We get

$$\begin{aligned} X_c(P \cup R) &= \frac{X_c(P)A(P) + X_c(R)A(R)}{A(P) + A(R)} \\ &> \frac{m'A(P) + m'A(R)}{A(P) + A(R)} = m' \end{aligned}$$

which contradicts the assumption that P is the object in $V(P_I, P_E)$ that achieves the largest displacement of the center of mass.

Consider case (ii) and let R be a closed connected subset with $A(R) > 0$ of $P^-[m'] - P_I^-[m']$. Observe that $P - R \in V(P_M)$. Note that $R \subset (-\infty, m') \times \mathbb{R}$ and thus $X_c(R) < m'$. We get

$$\begin{aligned} X_c(P - R) &= \frac{X_c(P)A(P) - X_c(R)A(R)}{A(P) - A(R)} \\ &> \frac{m'A(P) - m'A(R)}{A(P) - A(R)} = m \end{aligned}$$

which again contradicts the assumption that P is the object in $V(P_I, P_E)$ that achieves the largest displacement of the center of mass. As a result we find that $P^*[m]$ with $X_c(P^*[m]) = m$ is the unique object in $V(P_I, P_E)$ that achieves the largest displacement of the center of mass. \square

The theorem shows that the set $COM(P_I, P_E)$ does not extend beyond (i.e., to the right of) the line $x = m$ where m is such that $X_c(P^*[m]) = m$. The bound is tight because $P^*[m] \in V(P_I, P_E)$. In fact, the theorem shows that $P^*[m]$ is the only instance in $V(P_I, P_E)$ that has its center of mass on that line. Since the result holds in any direction, this implies that $COM(P_I, P_E)$ is convex.

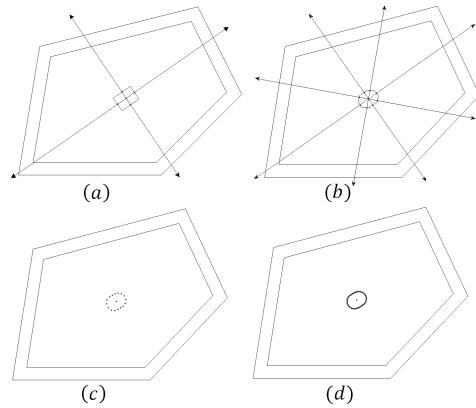


Figure 3: Outer approximations of $COM(P_I, P_E)$ with (a) 4, (b) 8, (c) 16, and (d) 64 vertices.

3.2 A k -vertex approximation for $COM(P_I, P_E)$

The results in the previous subsection suggest an easy approach to determine an outer approximation of the set $COM(P_I, P_E)$ of possible centers of mass of instances in $V(P_I, P_E)$. If we select k different directions that positively span the plane and apply Theorem 2 in each of these directions then we obtain a bounded polygon with k edges enclosing $COM(P_I, P_E)$. To find the largest displacement in the positive x -direction, we sweep a vertical line from $x = 0$ to $x = X_r$ while maintaining $X_c(P^*[m])$ using Equation 2. The mathematical descriptions of $A(Q^+[m])$ and $X_c(Q^+[m])$ only change when the line hits a vertex of P_I or P_E , as the boundary of these two shapes determine the boundary of Q . The corresponding update of these mathematical descriptions at such a vertex can be accomplished in constant time. Moreover, the check to decide whether the equation $X_c(P^*[m]) = m$ has a solution between the current vertex and the next vertex hit by the line also takes constant time as it requires (for polygonal P_I and P_E) computing the roots of a polynomial function of degree four. Since P_I and P_E have n vertices we obtain the following theorem.

Theorem 3 A polygonal k -vertex outer approximation of $COM(P_I, P_E)$ can be computed in $O(kn)$ time.

Lemma 1 suggests that we can use binary search to identify the m satisfying $X_c(P^*[m]) = m$. Unfortunately it seems impossible to evaluate $A(Q^+[m])$ and $X_c(Q^+[m])$ for an arbitrary m in $o(n)$ time, which would be crucial to improve the time bound reported in Theorem 3.

Figure 3 shows 4-, 8-, 16-, and 64-vertex outer approximations of $COM(P_I, P_E)$ for a given P_I and P_E . Recall that every edge of the polygonal approximation contains one point of the convex set $COM(P_I, P_E)$, so $COM(P_I, P_E)$ strongly resembles its approximation.

4 Bounding the size of $COM(P_I, P_E)$

The admitted shape variation for a manufactured part is usually small compared to the dimensions of the part itself. As a result, the enclosed shape P_I and enclosing shape P_E do not deviate much from the model shape P_M , and therefore also not from each other. To capture this similarity we will assume that $P_I \subseteq P_E \subseteq P_I \oplus D_\epsilon$, where D_ϵ is the disk of radius ϵ centered at the origin and \oplus denotes the Minkowski sum. Note that this means that every point in P_E is within a distance of at most ϵ from some point in P_I . In Subsection 4.1 we will see that this distance constraint alone is not enough to obtain a bound on the diameter of $COM(P_I, P_E)$ that is independent of the size of P_I and P_E . In Subsection 4.2 we show that the additional assumption that P_I is fat leads to a bound on the diameter of $COM(P_I, P_E)$ that depends on ϵ and the fatness.

4.1 A thin part

When P_I is a sufficiently long and narrow box the set $V(P_I, P_E)$ contains shapes whose centers of mass are a distance proportional to the diameter of P_I apart. Let $L \gg \epsilon$ and pick δ such that $0 < \delta < \epsilon^2/(2L - \epsilon)$. We define $P_I = [-L/2, L/2] \times [-\delta/2, \delta/2]$ and $P_E = [-(L+\epsilon)/2, (L+\epsilon)/2] \times [-(\delta+\epsilon)/2, (\delta+\epsilon)/2]$, and note that $P_E \subseteq P_I \oplus D_\epsilon$. Now consider the object $P^*[L/2] = P_I^-[L/2] \cup P_E^+[L/2] = P_I \cup P_E^+[L/2]$. We observe that $A(P_I) = \delta L$, $X_c(P_I) = 0$, $A(P_E^+[L/2]) = \epsilon(\epsilon + \delta)/2$, and $X_c(P_E^+[L/2]) = L/2 + \epsilon/4 > L/2$. The upper bound on δ implies that $A(P_E^+[L/2]) > A(P_I)$. From Equation 1 it follows that $X_c(P^*[L/2]) > L/4$ showing that the diameter of $COM(P_I, P_E)$ is not proportional to ϵ in this case.

4.2 Fat parts

In this subsection we add the assumption that P_I is fat to deduce a bound on the diameter of $COM(P_I, P_E)$ that depends on ϵ and the fatness. There are many different definitions of fatness and we will use the one by De Berg et al. [23], which is based on a similar definition presented in the thesis of van der Stappen [22].

Definition 1 Let $P \subseteq \mathbb{R}^2$ be an object and let β be a constant with $0 < \beta \leq 1$. Define $U(P)$ as the set of all disks centered inside P whose boundary intersects P . We say that the object P is β -fat if for all disks $D \in U(P)$ we have $A(P \cap D) \geq \beta \cdot A(D)$. The fatness of P is defined as the maximal β for which P is β -fat.

For bounded objects the value of β is at most $1/4$; larger values only occur for unbounded objects [22].

In the remainder of this section we assume that P_I is β -fat ($0 < \beta \leq 1$). The main implication of this

assumption is that it provides us with a lower bound on the area of P_I in terms of its diameter.

The following lemma is not strictly necessary yet it leads to a better bound in our main theorem. We omit the proof and immediately apply it in the theorem.

Lemma 4 Let P be a convex polygon with diameter d . Then no point in P has distance larger than $\frac{2d}{3}$ to the center of mass of P .

Theorem 5 Let P_I be a bounded convex β -fat object ($0 < \beta \leq 1$) and let P_E be a bounded object satisfying $P_I \subseteq P_E \subseteq P_I \oplus D_\epsilon$. Then the diameter of $COM(P_I, P_E)$ is bounded by $\frac{5}{2}\beta^{-1}\epsilon$.

Proof. We use d to denote the diameter of P_I and once again assume without loss of generality that the center of mass of P_I coincides with the origin. Theorem 2 shows that it suffices to consider objects $P^*[m]$ ($0 \leq m \leq X_r$) to bound the size of $COM(P_I, P_E)$. Lemma 4 says that P_I lies completely inside the disk $D(2d/3)$. As a consequence, the object $P^*[m]$ must lie entirely inside $D(2d/3 + \epsilon)$, which implies that $X_c(P^*[m]), X_c(Q^+[m]) \leq 2d/3 + \epsilon$. We distinguish two cases based on the ratio of ϵ and d .

If $\epsilon \geq d/6$ then $X_c(P^*[m]) \leq 2d/3 + \epsilon \leq 5\epsilon$. Since $P^*[m]$ is bounded we know that $\beta \leq 1/4$ and thus $X_c(P^*[m]) \leq 5\epsilon \leq 5\beta^{-1}\epsilon/4$.

If $\epsilon \leq d/6$ we use Equation 2 to obtain an upper bound $X_c(P^*[m])$ by combining the upper bound $X_c(Q^+[m]) \leq 2d/3 + \epsilon$ with a lower bound on $A(P_I)$ and upper and lower bounds on $A(Q^+[m])$. The lower bound on $A(P_I)$ follows from the fatness of P_I . As d is the diameter of P_I there must be two points $p_1, p_2 \in P_I$ that are d apart. The boundary of the disk $D_d(p_1)$ contains p_2 and thus belongs to the set $U(P_I)$. The β -fatness of P_I implies that $A(P_I) \geq \beta \cdot A(D_d(p_1)) = \beta\pi d^2$.

It remains to bound $A(Q^+[m])$. Recall that $Q = P_E - int(P_I)$. Due to our assumption $P_E \subseteq P_I \oplus D_\epsilon$ we obtain that $Q \subseteq Q' = (P_I \oplus D_\epsilon) - int(P_I)$, from which it follows that $A(Q^+[m]) \leq A(Q) \leq A(Q')$. We observe that the convexity of P_I implies that $A(Q') = l\epsilon + \pi\epsilon^2$, where l denotes the perimeter of P_I . As P_I is contained in $D(2d/3)$ the perimeter l of P_I is bounded by $4\pi d/3$. Combining these observations with a trivial lower on $A(Q^+[m])$ we get $0 \leq A(Q^+[m]) \leq 4\pi\epsilon d/3 + \pi\epsilon^2$.

Plugging all the inequalities into Equation 2 and using $\epsilon/d \leq 1/6$ yields

$$\begin{aligned} X_c(P^*[m]) &= \frac{X_c(Q^+[m])A(Q^+[m])}{A(P_I) + A(Q^+[m])} \\ &\leq \frac{(\frac{2}{3}d + \epsilon)(\frac{4}{3}\pi\epsilon d + \pi\epsilon^2)}{\beta\pi d^2} \\ &= \beta^{-1}\epsilon(\frac{8}{9} + 2(\frac{\epsilon}{d}) + (\frac{\epsilon}{d})^2) \\ &\leq \frac{5}{4}\beta^{-1}\epsilon. \end{aligned}$$

which shows $COM(P_I, P_E) \subseteq D(\frac{5}{4}\beta^{-1}\epsilon)$. \square

Theorem 5 confirms the intuition that the variation of the center of mass grows if the admitted shape variation increases or the fatness decreases.

5 Conclusion

We have considered a very general model for admitted shape variations of a model part, based on enclosed convex shape P_I and an enclosing convex shape P_E . We have identified the valid instance that maximizes the displacement of the center of mass in a given direction, and used this result to find a k -vertex polygonal outer approximation of the set of all possible center-of-mass loci in $O(kn)$ time, where n is the number of vertices of P_I and P_E . If P_I is β -fat and every point of P_E is within a distance ϵ of P_I then the diameter of the set of all center-of-mass loci can be shown to be $O(\beta^{-1}\epsilon)$.

We expect that our results will generalize to three-dimensional objects. It is interesting to see under which circumstances the results can be extended to non-convex shapes P_I (and P_E), and to parts with non-uniform mass distribution.

References

- [1] K. Goldberg, Orienting polygonal parts without sensors. *Algorithmica*, 10(2), pp. 201–225, 1993.
- [2] R.-P. Berretty, M.H. Overmars, A.F. van der Stappen, Orienting polyhedral parts by pushing. *Computational Geometry: Theory and Applications*, 21, pp. 21–38, 2002.
- [3] S. Akella, M.T. Mason, Posing polygonal objects in the plane by pushing. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2255–2262, 1992.
- [4] M.A. Erdmann, M.T. Mason, An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4, pp. 367–379, 1988.
- [5] K.M. Lynch, Locally controllable manipulation by stable pushing. *IEEE Transactions on Robotics and Automation*, 15, pp. 314–323, 1999.
- [6] R.-P. Berretty, K. Goldberg, M.H. Overmars, A.F. van der Stappen, Computing fence designs for orienting parts. *Computational Geometry: Theory and Applications*, 10(4), pp. 249–262, 1998.
- [7] R.C. Brost, Automatic grasp planning in the presence of uncertainty. *International Journal of Robotics Research*, 7(1), pp. 3–17, 1988.
- [8] B.R. Donald, Error Detection and Recovery in Robotics. *Springer-Verlag*, 1987.
- [9] J. Chen, K. Goldberg, M.H. Overmars, D. Halperin, K.-F. Böhringer, Y. Zhuang, Shape tolerance in feeding and fixturing. *Robotics: the algorithmic perspective*, pp. 297–311. A.K. Peters, 1998.
- [10] S.M. LaValle, Planning Algorithms, chapter 12: Planning Under Sensing Uncertainty. *Cambridge University press*, 2006.
- [11] M. Dogar, S.S. Srinivasa, A Framework for Push-grasping in Clutter. *Robotics: Science and Systems* 2, 2011.
- [12] A.A.G. Requicha, Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2, 1983.
- [13] H. Voelker. A current perspective on tolerancing and metrology. *Manufacturing Review*, 6(4), 1993.
- [14] L. Guibas, D. Salesin, J. Stolfi, Epsilon geometry: Building robust algorithms for imprecise computations. *Proc. of the 5th Annual ACM Symp. on Computational Geometry*, pp. 208–217, 1989.
- [15] U. Roy, C. Liu, T. Woo. Review of dimensioning and tolerancing. *Computer-Aided Design*, 23(7), 1991.
- [16] Y. Ostrovsky-Berman, L. Joskowicz, Tolerance envelopes of planar mechanical parts with parametric tolerances. *Computer-Aided Design*, pp. 531–534, 2005.
- [17] M. Löffler, Data Imprecision in Computational Geometry. *PhD Thesis, Utrecht University*, 2009.
- [18] S. Akella, M.T. Mason, Orienting Toleranced Polygonal Parts. *International Journal of Robotics Research*, 19(12), pp. 1147–1170, 2000.
- [19] M. Bern, D. Eppstein, L.J. Guibas, J.E. Hershberger, S. Suri, J. Wolter, The centroid of points with approximate weights. *Proc. 3rd Eur. Symp. Algorithms*, LNCS 979, pp. 460–472, 1995.
- [20] Y.-B. Chen, D.J. Ierardi, The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algorithmica*, 14, pp. 367–397, 1995.
- [21] B. Kehoe, D. Berenson, K. Goldberg, Toward Cloud-Based Grasping with Uncertainty in Shape: Estimating Lower Bounds on Achieving Force Closure with Zero-Slip Push Grasps. *Proc. IEEE Int. Conf. on Robotics and Automation*, 2012.
- [22] A.F. van der Stappen, Motion Planning amidst Fat Obstacles. *Ph.D. Thesis, Utrecht University*, 1994.
- [23] M. de Berg, M.J. Katz, A.F. van der Stappen, J. Vleugels, Realistic input models for geometric algorithms, *Algorithmica*, 34, pp. 81–97, 2002.
- [24] B. Aronov and M. de Berg, Unions of fat convex polytopes have short skeletons. *Discrete and Computational Geometry*, 48(1): 53–64, 2012.
- [25] M.J. Katz, M.H. Overmars, M. Sharir, Efficient hidden surface removal for objects with small union size, *Computational Geometry: Theory and Applications*, 2, pp. 223–234, 1992.
- [26] M.H. Overmars, A.F. van der Stappen, Range searching and point location among fat objects, *Journal of Algorithms*, pp. 626–656, 1996.

Geometric Separators and the Parabolic Lift

Donald R. Sheehy*

Abstract

A geometric separator for a set U of n geometric objects (usually balls) is a small (sublinear in n) subset whose removal disconnects the intersection graph of U into roughly equal sized parts. These separators provide a natural way to do divide and conquer in geometric settings. A particularly nice geometric separator algorithm originally introduced by Miller and Thurston has three steps: compute a centerpoint in a space of one dimension higher than the input, compute a conformal transformation that “centers” the centerpoint, and finally, use the computed transformation to sample a sphere in the original space. The output separator is the subset of S intersecting this sphere. It is both simple and elegant. We show that a change of perspective (literally) can make this algorithm even simpler by eliminating the entire middle step. By computing the centerpoint of the points lifted onto a paraboloid rather than using the stereographic map as in the original method, one can sample the desired sphere directly, without computing the conformal transformation.

1 Geometric Separators

A spherical geometric separator of a collection of n balls in \mathbb{R}^d is a sphere S that has at least $\frac{n}{d+2}$ balls centered inside, at least $\frac{n}{d+2}$ centered outside, and intersects at most $O(n^{1-\frac{1}{d}})$ of them (see Section 2 for a formal definition). The existence of such separators in two and three dimensions was established by Miller and Thurston, though their method was quickly adapted to higher dimensions. Across a series of papers, Miller, Thurston, Teng, and Vavasis laid out the theory of geometric separators and their applications to scientific computing [14, 15, 13, 10, 12]. This line of work is a treasure trove for computational geometers as it hinges on a novel trick that combines projective and combinatorial geometry to solve an important algorithmic problem, solving linear systems arising in finite element analysis.

More generally, geometric separators give a natural way to do divide and conquer for geometric problems. They have been applied to various nearest neighbor search problems [11] as well as to mesh compression [1]. Other variations of geometric separators have been used

for the Traveling Salesman and Minimum Steiner Tree problems in geometric settings [18], or for packing and piercing problems [2].

The Miller-Thurston algorithm for computing a geometric separator maps the n points (the centers of the balls) to a unit d -sphere in \mathbb{R}^{d+1} via a stereographic map. It then computes a centerpoint, which is a geometric generalization of a median (a formal definition is given in Section 2). There exists a conformal transformation of the points in \mathbb{R}^{d+1} so that this centerpoint will lie exactly at the origin. To output a separator, one samples a random unit vector in \mathbb{R}^{d+1} . The hyperplane through the origin normal to this vector intersects the unit d -sphere at a $(d-1)$ -sphere. The output is just the stereographic projection of this $(d-1)$ -sphere back to \mathbb{R}^d . With high probability, such a sphere will be a geometric separator.

The one aspect of this algorithm that was left to the reader, was the linear algebra required to compute the desired conformal transformation. In the original paper, it was simply asserted that it exists. Later papers explained that it can be computed via Householder transformations and cited a textbook on matrix computations. In this paper, we show that this phase of the algorithm is entirely unnecessary. By working initially with the parabolic lifting

$$\mathbf{p} \mapsto \begin{bmatrix} \mathbf{p} \\ \|\mathbf{p}\|^2 \end{bmatrix},$$

rather than the stereographic map, the desired sphere can be sampled directly.

Related work In addition to the previously mentioned work on sphere separators and their applications by various combinations of Miller, Thurston, Teng, and Vavasis, the generalization to other types of contact graphs and other shapes of separators (in particular hypercubes) was performed by Smith and Wormald [18]. This method was refined slightly by Chan to apply separators to geometric hitting set problems [2]. Eppstein et al. gave a linear-time, deterministic algorithm for finding geometric separators based on core sets [5]. Har-Peled gave a simple proof of the existence of geometric separators for interior-disjoint disks in the plane (that easily extends to higher dimensions) [6].

*Department of Computer Science and Engineering, University of Connecticut, don.r.sheehy@gmail.com

2 Definitions

Points We treat points in Euclidean space as column vectors. As the algebra will be in both \mathbb{R}^d as well as \mathbb{R}^{d+1} , we adopt the convention that a boldface vector \mathbf{p} is a vector of \mathbb{R}^d and an overline such as in $\bar{\mathbf{p}}$ indicates a vector in \mathbb{R}^{d+1} . In particular, we write $\mathbf{0}$ to indicate the zero vector in \mathbb{R}^d . Scalars are italic and when it is useful, we add a subscript $d+1$ as in p_{d+1} to indicate a scalar that is the $(d+1)$ st coordinate of a vector in \mathbb{R}^{d+1} . So, for example, we will write $\bar{\mathbf{p}} = \begin{bmatrix} \mathbf{p} \\ p_{d+1} \end{bmatrix}$ to indicate that $\bar{\mathbf{p}} \in \mathbb{R}^{d+1}$ with its first d coordinates matching those of \mathbf{p} and last coordinate equal to p_{d+1} . Whenever we speak of \mathbb{R}^d as a subspace of \mathbb{R}^{d+1} , it is always assumed that we mean the hyperplane $\{\bar{\mathbf{p}} \mid p_{d+1} = 0\}$ in \mathbb{R}^{d+1} .

Projections Given a point $\bar{\mathbf{f}} = \begin{bmatrix} \mathbf{f} \\ f_{d+1} \end{bmatrix}$, we define $\Pi_{\bar{\mathbf{f}}}^1$ to be the stereographic map from \mathbb{R}^d to the sphere centered at $\bar{\mathbf{f}}$ with radius f_{d+1} . That is, $\Pi_{\bar{\mathbf{f}}}^1(\mathbf{p})$ is the intersection of the sphere with the line through $\begin{bmatrix} \mathbf{p} \\ 0 \end{bmatrix}$ and the north pole, $\begin{bmatrix} \mathbf{f} \\ 2f_{d+1} \end{bmatrix}$ (other than the pole itself). Similarly, we define $\Pi_{\bar{\mathbf{f}}}^\infty$ to be the parabolic lifting map from \mathbb{R}^d to \mathbb{R}^{d+1} that lifts \mathbf{p} to $\bar{\mathbf{p}} = \begin{bmatrix} \mathbf{p} \\ p_{d+1} \end{bmatrix}$ so that $\bar{\mathbf{p}}$ lies on the paraboloid with focal point $\bar{\mathbf{f}}$ and directrix $\{p_{d+1} = -f_{d+1}\}$. The reason for the notation comes from the fact that the parabola is the limiting case of an ellipse formed by moving one focal point to infinity. We will consider more general stereographic projections in Section 4. In all cases, these maps are invertible (except at the north pole).

We abuse notation slightly and let $\Pi_{\bar{\mathbf{f}}}^1(S)$ denote the set $\{\Pi_{\bar{\mathbf{f}}}^1(\mathbf{p}) \mid \mathbf{p} \in S\}$ and similarly for $\Pi_{\bar{\mathbf{f}}}^\infty$. In particular, $\Pi_{\bar{\mathbf{f}}}^1(\mathbb{R}^d)$ denotes the sphere centered at $\bar{\mathbf{f}}$ with radius f_{d+1} and $\Pi_{\bar{\mathbf{f}}}^\infty(\mathbb{R}^d)$ denotes the paraboloid with focal point $\bar{\mathbf{f}}$ and directrix $\{p_{d+1} = -f_{d+1}\}$.

Centerpoints Given n points in \mathbb{R}^d , a centerpoint is a point $\mathbf{c} \in \mathbb{R}^d$ such that any closed halfspace containing more than $\frac{nd}{d+1}$ points also contains \mathbf{c} . The existence of centerpoints follows from Helly's Theorem and the pigeonhole principle. Let $\text{CENTERPOINT}(P)$ denote the set of all centerpoints of the set P .

It is not known how to find a centerpoint deterministically in polynomial time for point sets in \mathbb{R}^d . However, there is an efficient randomized algorithm [3] known for at least twenty years and some recent work on deterministic approximation algorithms [9, 16].

Sphere Separators A **graph separator** is a subset of vertices in a graph whose removal disconnects the graph. Usually, the goal is to find a small separator that separates the graph into roughly equal sized pieces. The most famous example of graphs admitting small separators is the Planar Separator Theorem of Lipton

and Tarjan [8], which states that a separator of size $O(\sqrt{n})$ is always possible for planar graphs that has at least $\frac{n}{3}$ vertices in each of the resulting components.

The early work on separators was directed at solving linear systems by generalized nested dissection, a method for ordering pivots in Cholesky decomposition of sparse, symmetric, positive definite matrices [7]. Of particular interest were those linear systems arising in the finite element method. These systems reflected the underlying geometric structure of the problem domain and thus it was natural to look to the geometry to find small separators [12]. In particular, it often sufficed to consider various definitions of intersection graphs of systems of balls.

Let $\mathcal{B} = \{B_1, \dots, B_n\}$ be a collection of interior disjoint balls in \mathbb{R}^d . Let S be a sphere in \mathbb{R}^d and let $\mathcal{B}_I(S)$, $\mathcal{B}_E(S)$, and $\mathcal{B}_O(S)$ be the subsets of \mathcal{B} that are interior to, exterior to, and intersecting S respectively. The following theorem is a combination of the main existence result for geometric separators with the algorithm used to prove existence. We state it this way, to make clear that the geometric challenge for computing a separator this way lies in finding a stereographic projection to a sphere that has a centerpoint at the center of the sphere.

Theorem 1 (Sphere Separator Thm.[14, 11, 12])
Let \mathcal{B} be a collection of n interior disjoint balls with centers P . Let $\bar{\mathbf{v}} \in \mathbb{R}^{d+1}$ be chosen uniformly from the unit d -sphere. If $\bar{\mathbf{f}}$ is a point in \mathbb{R}^{d+1} such that $\bar{\mathbf{f}}$ is a centerpoint of $\Pi_{\bar{\mathbf{f}}}^1(P)$ and $H = \{\bar{\mathbf{p}} \mid \bar{\mathbf{v}}^\top(\bar{\mathbf{p}} - \bar{\mathbf{f}}) = 0\}$ is the hyperplane through $\bar{\mathbf{f}}$ normal to $\bar{\mathbf{v}}$, then the sphere

$$S = (\Pi_{\bar{\mathbf{f}}}^1)^{-1}(\Pi_{\bar{\mathbf{f}}}^1(\mathbb{R}^d) \cap H)$$

has the property that

$$|\mathcal{B}_O(S)| = O(n^{1-1/d}), \text{ and}$$

$$|\mathcal{B}_I(S)|, |\mathcal{B}_E(S)| \leq \frac{(d+1)n}{d+2}$$

with probability at least $\frac{1}{2}$.

For ease of exposition, we only give this simplest version of the theorem. However, it has also been proven for k -ply neighborhood systems where the balls are permitted to have up to k -wise interior intersections [11] as well as for α -overlap graphs where two balls are considered neighbors if for both balls increasing the radius of one by a factor of α causes them to intersect [12]. In these cases, the bounds depend on k and α respectively, but the algorithm is the same and so the results of this paper apply to these versions of the theorem as well. The version stated above, when combined with the Koebe-Andreev-Thurston embedding theorem for planar graphs is strong enough to prove the Planar Separator Theorem.

3 The Algorithm

When Archimedes quipped that he could move the earth if given a sufficiently long lever, he transferred the majority of the work to the lever builders of his day. We will do likewise and assume that we are given a long lever in the form of an algorithm to efficiently compute a centerpoint of n points in \mathbb{R}^{d+1} . Given such an algorithm, the heavy lifting will be quite easy, and as with Archimedes, that is entirely the point.

Let $P = \{\mathbf{p}_1, \dots, \mathbf{p}_n\} \subset \mathbb{R}^d$ be a set of points. We first present the Miller-Thurston algorithm for computing a separator and then present the new simplified version.

3.1 The Miller-Thurston Algorithm

Let Π be the stereographic map from \mathbb{R}^d to the unit d -sphere centered at the origin in \mathbb{R}^{d+1} . First, compute

$$\bar{\mathbf{c}} \in \text{CENTERPOINT}(\Pi(P)).$$

Find an orthogonal transformation Q such that

$$Q(\bar{\mathbf{c}}) = \begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix} \text{ for some } \theta \in \mathbb{R}.$$

Let $D = \sqrt{\frac{1-\theta}{1+\theta}}I$, where I is the identity on \mathbb{R}^d .

Choose a random unit vector $\bar{\mathbf{v}} \in \mathbb{R}^{d+1}$ and let S_0 be the d -sphere formed by intersecting the hyperplane $\{\bar{\mathbf{p}} \mid \bar{\mathbf{v}}^\top \bar{\mathbf{p}} = 0\}$ with the unit d -sphere centered at the origin. **Output** $S = \Pi^{-1}(Q^{-1}\Pi(D\Pi^{-1}(S_0)))$.

3.2 A Simpler Algorithm Using the Parabolic Lift

First, compute

$$\bar{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ c_{d+1} \end{bmatrix} \in \text{CENTERPOINT}\left(\left[\frac{\mathbf{p}_1}{\|\mathbf{p}_1\|^2}\right], \dots, \left[\frac{\mathbf{p}_n}{\|\mathbf{p}_n\|^2}\right]\right).$$

Next, choose a random unit vector $\bar{\mathbf{v}} = \begin{bmatrix} \mathbf{v} \\ v_{d+1} \end{bmatrix} \in \mathbb{R}^{d+1}$. Let

$$r = \frac{\sqrt{c_{d+1} - \|\mathbf{c}\|^2}}{|v_{d+1}|}.$$

Output the sphere S with center $(\mathbf{c} - r\mathbf{v})$ and radius r . In the improbable case that $v_{d+1} = 0$, the output is just the hyperplane $\{\mathbf{p} \mid \mathbf{v}^\top (\mathbf{p} - \mathbf{c}) = 0\}$.

3.3 Correctness of the Algorithm

The remainder of this section will prove that the algorithm works. According to the results of Miller et al. [12], all we need is a stereographic projection of \mathbb{R}^d to a d -sphere such that the projected points have a centerpoint at the center of the sphere. The trick presented here is that we will instead show how to find a parabolic lifting that has a centerpoint at the focal point of the paraboloid. Then, we show that if we have a point $\bar{\mathbf{f}}$ such that $\Pi_{\bar{\mathbf{f}}}^\infty(P)$ has a centerpoint at $\bar{\mathbf{f}}$, then $\Pi_{\bar{\mathbf{f}}}^1(P)$

also has a centerpoint at $\bar{\mathbf{f}}$, thus giving the desired map. For any vector $\bar{\mathbf{v}} \in \mathbb{R}^{d+1}$ and any $\mathbf{p} \in \mathbb{R}^d$,

$$\bar{\mathbf{v}}^\top (\Pi_{\bar{\mathbf{f}}}^1(\mathbf{p}) - \bar{\mathbf{f}}) = 0 \text{ iff } \bar{\mathbf{v}}^\top (\Pi_{\bar{\mathbf{f}}}^\infty(\mathbf{p}) - \bar{\mathbf{f}}) = 0.$$

That is, for sampling spheres, there is no difference between using the stereographic map or the parabolic lifting. In fact, we prove a much more general statement about the equivalence of various stereographic projections in Theorem 3.

Theorem 2 *Let \mathcal{B} be a collection of n interior disjoint balls with centers $P = \{\mathbf{p}_1, \dots, \mathbf{p}_n\} \subset \mathbb{R}^d$ and let S be the sphere output by the algorithm in Section 3.2. Then,*

$$|\mathcal{B}_O(S)| = O(n^{1-1/d}), \text{ and}$$

$$|\mathcal{B}_I(S)|, |\mathcal{B}_E(S)| \leq \frac{(d+1)n}{d+2}$$

with probability at least $\frac{1}{2}$.

Proof. Using Theorem 1, it will suffice to show there exists $\bar{\mathbf{f}}$ such that $S = (\Pi_{\bar{\mathbf{f}}}^1)^{-1}(\Pi_{\bar{\mathbf{f}}}^1(\mathbb{R}^d) \cap H)$ where

$$H = \{\bar{\mathbf{p}} \mid \bar{\mathbf{v}}^\top (\bar{\mathbf{p}} - \bar{\mathbf{f}}) = 0\}$$

and $\bar{\mathbf{f}} \in \text{CENTERPOINT}(\Pi_{\bar{\mathbf{f}}}^1(P))$. The equivalence of different stereographic projections proven in Theorem 3 implies that it will suffice to prove the same facts replacing the stereographic map $\Pi_{\bar{\mathbf{f}}}^1$ with the parabolic lift $\Pi_{\bar{\mathbf{f}}}^\infty$. We will show that the focal point that makes this true is

$$\bar{\mathbf{f}} = \begin{bmatrix} \mathbf{c} \\ \frac{1}{2}\sqrt{c_{d+1} - \|\mathbf{c}\|^2} \end{bmatrix},$$

where $\bar{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ c_{d+1} \end{bmatrix}$ is the centerpoint of $\Pi_{\bar{\mathbf{f}}}^\infty(P)$ computed in the first step.

First, we show that $S = (\Pi_{\bar{\mathbf{f}}}^\infty)^{-1}(\Pi_{\bar{\mathbf{f}}}^\infty(\mathbb{R}^d) \cap H)$. We will assume without loss of generality that $v_{d+1} > 0$ since $\begin{bmatrix} \mathbf{v} \\ v_{d+1} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{v} \\ -v_{d+1} \end{bmatrix}$ are sampled with equal probability and both yield the same sphere. We need only check that S is the orthogonal projection of $\Pi_{\bar{\mathbf{f}}}^\infty(\mathbb{R}^d) \cap H$ to \mathbb{R}^d . The equation for $\Pi_{\bar{\mathbf{f}}}^\infty(\mathbb{R}^d)$ is

$$p_{d+1} = \frac{\|\mathbf{p} - \mathbf{c}\|^2}{2\sqrt{c_{d+1} - \|\mathbf{c}\|^2}} = \frac{\|\mathbf{p} - \mathbf{c}\|^2}{2rv_{d+1}},$$

and the equation for H can be rewritten as

$$\begin{aligned} p_{d+1} &= \frac{\mathbf{v}^\top (\mathbf{c} - \mathbf{p})}{v_{d+1}} + \frac{1}{2}\sqrt{c_{d+1} - \|\mathbf{c}\|^2} \\ &= \frac{\mathbf{v}^\top (\mathbf{c} - \mathbf{p})}{v_{d+1}} + \frac{1}{2}rv_{d+1}. \end{aligned}$$

So, for a point $\bar{\mathbf{p}} = \begin{bmatrix} \mathbf{p} \\ p_{d+1} \end{bmatrix}$ in the intersection,

$$\frac{\|\mathbf{p} - \mathbf{c}\|^2}{2rv_{d+1}} = \frac{\mathbf{v}^\top (\mathbf{c} - \mathbf{p})}{v_{d+1}} + \frac{1}{2}rv_{d+1}.$$

Multiplying by -2 and completing the square twice gives

$$\|\mathbf{p} - (\mathbf{c} - r\mathbf{v})\|^2 = r^2 v_{d+1}^2 + \|r\mathbf{v}\|^2 = r^2,$$

which is precisely the equation for S .

Now, we show that the focal point $\bar{\mathbf{f}}$ is a centerpoint of $\Pi_{\bar{\mathbf{f}}}^\infty(P)$. There are several different ways to show this, but one nice approach is to observe that for any hyperplane normal to $\bar{\mathbf{v}}$ passing through $\bar{\mathbf{f}}$ there is a corresponding hyperplane H normal to $[\frac{\mathbf{c}-r\mathbf{v}}{-\frac{1}{2}}]$ passing through $\bar{\mathbf{c}}$. The hyperplane H intersects the paraboloid $\Phi = \{[\frac{\mathbf{p}}{p_{d+1}}] \mid p_{d+1} = \|\mathbf{p}\|^2\}$ in an ellipse that projects orthogonally to the very same sphere of radius r centered at $\mathbf{c} - r\mathbf{v}$. It will suffice to show for any $\bar{\mathbf{p}} = [\frac{\mathbf{p}}{\|\mathbf{p}\|^2}] \in H \cap \Phi$, that $\|\bar{\mathbf{p}} - (\mathbf{c} - r\mathbf{v})\|^2 = r^2$. By the definition of H ,

$$(\mathbf{c} - r\mathbf{v})^\top \bar{\mathbf{p}} - \frac{1}{2} \|\bar{\mathbf{p}}\|^2 = (\mathbf{c} - r\mathbf{v})^\top \mathbf{c} - \frac{1}{2} c_{d+1}.$$

Multiplying by -2 and completing the square yields

$$\begin{aligned} \|\bar{\mathbf{p}} - (\mathbf{c} - r\mathbf{v})\|^2 &= \|\mathbf{c} - r\mathbf{v}\|^2 - 2(\mathbf{c} - r\mathbf{v})^\top \mathbf{c} + c_{d+1} \\ &= r^2 \|\mathbf{v}\|^2 - \|\mathbf{c}\|^2 + c_{d+1} \\ &= r^2. \end{aligned}$$

The last equality follows from the definition of r and the fact that $\|\bar{\mathbf{v}}\|^2 = 1$.

This implies that $\bar{\mathbf{f}}$ is a centerpoint of the points $\Pi_{\bar{\mathbf{f}}}^\infty(P)$ because every hyperplane passing through $\bar{\mathbf{f}}$ separates the same set of points as the corresponding hyperplane through $\bar{\mathbf{c}}$, which is a centerpoint by definition. \square

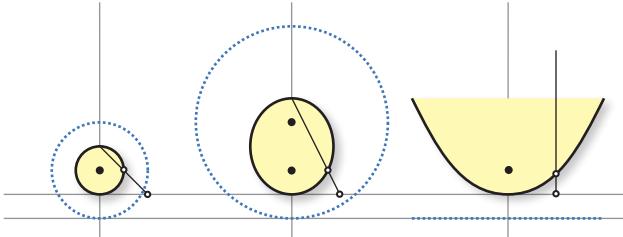


Figure 1: Three examples of E_R are illustrated for $R = 1$, $R = 2$, and $R = \infty$ from left to right. In each case, the stereographic map of a single point is illustrated. For the last case, the pole is at infinity, so the result is a vertical projection.

4 Equivalence of Stereographic Maps

Let $\bar{\mathbf{f}} = [\frac{0}{1}]$. Let R be a real number and consider the ellipsoid $E_R \subset \mathbb{R}^{d+1}$ defined as the points $\bar{\mathbf{p}} = [\frac{\mathbf{p}}{p_{d+1}}]$ such that

$$\frac{\|\mathbf{p}\|^2}{2R-1} + \frac{(p_{d+1}-R)^2}{R^2} = 1.$$

It is tangent to $[\frac{\mathbb{R}^d}{0}]$ at the origin, has major radius R and all minor radii all equal to $\sqrt{2R-1}$. The stereographic map $\Pi_{\bar{\mathbf{f}}}^R$ through $[\frac{0}{2R}]$, the north pole of this ellipse, from \mathbb{R}^d to E_R is well-defined as is the inverse map (except at $[\frac{0}{2R}]$).

As R goes to infinity, E_R converges to the paraboloid $p_{d+1} = \|\mathbf{p}\|^2/4$. This is perhaps easier to see when one observes that E_R is the set of points equidistant from $\bar{\mathbf{f}}$ and the sphere centered at $[\frac{0}{2R-1}]$ with radius $2R$. As R goes to infinity, this sphere becomes a plane and the paraboloid is the set of points equidistant from a point and a plane. See Figure 1.

If we intersect the ellipsoid E_R with a hyperplane through one of its focal points and then stereographically project that intersection to \mathbb{R}^d from the pole of the ellipse, then the result is a sphere. The following theorem says that for a fixed hyperplane, it doesn't matter which ellipsoid E_R we started with, we always get the same sphere as illustrated in Figure 2.

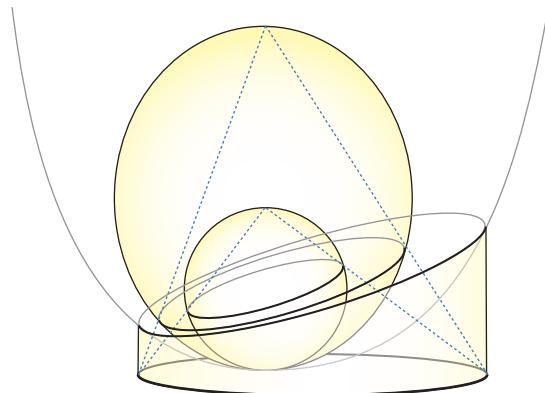


Figure 2: The stereographic projection of the intersection of the ellipse and a plane through the focal point is the same as the second focal point is moved up to infinity.

Theorem 3 Let α and β be real numbers in $[1, \infty]$ and let $\bar{\mathbf{f}} \in \mathbb{R}^{d+1}$ be any point. If H is a hyperplane containing $\bar{\mathbf{f}}$, then

$$(\Pi_{\bar{\mathbf{f}}}^\alpha)^{-1}(\Pi_{\bar{\mathbf{f}}}^\alpha(\mathbb{R}^d) \cap H) = (\Pi_{\bar{\mathbf{f}}}^\beta)^{-1}(\Pi_{\bar{\mathbf{f}}}^\beta(\mathbb{R}^d) \cap H).$$

Proof. The sphere $S^\alpha = (\Pi_{\bar{\mathbf{f}}}^\alpha)^{-1}(\Pi_{\bar{\mathbf{f}}}^\alpha(\mathbb{R}^d) \cap H)$ can also be written as

$$S^\alpha = \{\mathbf{p} \in \mathbb{R}^d \mid \bar{\mathbf{v}}^\top (\Pi_{\bar{\mathbf{f}}}^\alpha(\mathbf{p}) - \bar{\mathbf{f}}) = 0\},$$

where $\bar{\mathbf{v}}$ is the normal vector of H . By translating the points in \mathbb{R}^d and scaling the space uniformly, we may assume that $\bar{\mathbf{f}} = [\frac{0}{1}]$. Let R be any real number in the range $[1, \infty]$ and let $\mathbf{q} \in \mathbb{R}^d$ be any point. We will show that $\Pi_{\bar{\mathbf{f}}}^R(\mathbf{q})$ lies on a line through $\bar{\mathbf{f}}$ that does not

depend on R . Thus, $\bar{\mathbf{v}}^\top(\Pi_{\bar{\mathbf{f}}}^\alpha(\mathbf{q}) - \bar{\mathbf{f}}) = 0$ if and only if $\bar{\mathbf{v}}^\top(\Pi_{\bar{\mathbf{f}}}^\beta(\mathbf{q}) - \bar{\mathbf{f}}) = 0$ and therefore, the spheres S^α and S^β must be equal for all values of α and β .

Let $\bar{\mathbf{p}} = [p_{d+1}]$ be the projection $\Pi_{\bar{\mathbf{f}}}^R(\mathbf{q})$. All of the relevant points $\mathbf{0}$, $\bar{\mathbf{f}}$, \mathbf{p} , and \mathbf{q} are contained in a plane, so we proceed in two dimensions, letting $x = \|\mathbf{p}\|$, $y = p_{d+1}$, and $a = \|q\|$.

Since $\bar{\mathbf{p}}$ lies on E_R , we have $\frac{x^2}{2R-1} + \frac{(y-R)^2}{R^2} = 1$, or equivalently,

$$x^2 R^2 + y(y - 2R)(2R - 1) = 0. \quad (1)$$

Since $\bar{\mathbf{p}}$ is also on the line from $[0]$, the north pole of E_R , to $[q]$, its planar coordinates x and y satisfy the equation

$$y = \frac{-2R}{a}x + 2R.$$

It follows that

$$R = \frac{ay}{2(a-x)}, \quad 2R - 1 = \frac{a(y-1) + x}{a-x},$$

$$\text{and } y - 2R = \frac{-xy}{a-x}$$

Plugging these values into (1) gives the following.

$$\frac{x^2(ay)^2}{4(a-x)^2} + \frac{y(-xy)(a(y-1) + x)}{(a-x)^2} = 0.$$

Multiplying by $\frac{4(a-x)^2}{xy^2}$ and collecting terms yields the line

$$(a^2 - 4)x - 4a(y - 1) = 0.$$

We observe that this is the equation of a line through $\bar{\mathbf{f}} = (0, 1)$ as desired. \square

5 Concluding Remarks

The main story of this paper is not entirely new to computational geometry. When the parabolic lifting map was first introduced in the problem of computing Voronoi diagrams by Edelsbrunner and Seidel [4], they replaced previous methods based on the stereographic map. Today, the parabolic lifting is the preferred method for reducing Delaunay triangulations to convex hulls in one higher dimension.

Throughout, we have worked directly with the Euclidean coordinates. This gave the benefit of making the computations more immediately clear, but came at the cost of considering several special cases at infinity. An alternative approach would be to exploit the power of projective geometry, which has been shown to be the natural language for stereographic projections. Even the parabola is just an ellipse in the projective plane. For more on projective geometry, I highly recommend the book by Richter-Gebert [17].

I would like to thank Marc Glisse for helpful conversations and advice on high school algebra.

References

- [1] D. K. Blandford, G. E. Blelloch, D. E. Cardoze, and C. Kadow. Compact representations of simplicial meshes in two and three dimensions. *Int. J. Comput. Geometry Appl.*, 15(1):3–24, 2005.
- [2] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003.
- [3] K. Clarkson, D. Eppstein, G. Miller, C. Sturtivant, and S.-H. Teng. Approximating center points with iterated Radon points. *International Journal of Computational Geometry and Applications*, 6(3):357–377, Sep 1996. invited submission.
- [4] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1(1):25–44, 1986.
- [5] D. Eppstein, G. L. Miller, and S.-H. Teng. A deterministic linear time algorithm for geometric separators and its applications. *Fundamenta Informatica*, 22(4):309–329, 1995.
- [6] S. Har-Peled. A simple proof of the existence of a planar separator, July 10 2011.
- [7] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, 1979.
- [8] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.
- [9] G. L. Miller and D. R. Sheehy. Approximate centerpoints with proofs. *Computational Geometry: Theory and Applications*, 43(8):647–654, 2010.
- [10] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Graphs Theory and Sparse Matrix Computation*, The IMA Volumes in Mathematics and its Application, pages 57–84. Springer-Verlag, 1993. Vol 56.
- [11] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighborhood graphs. *Journal of the ACM*, 44(1):1–29, 1997.
- [12] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Geometric separators for finite-element meshes. *SIAM J. Comput.*, 19(2):364–386, 1998.
- [13] G. L. Miller, S.-H. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 538–547, 1991.
- [14] G. L. Miller and W. Thurston. Separators in two and three dimensions. In *Proceedings of the 22th Annual ACM Symposium on Theory of Computing*, pages 300–309. ACM, 1990.
- [15] G. L. Miller and S. A. Vavasis. Density graphs and separators. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 331–336. ACM-SIAM, 1991.

- [16] W. Mulzer and D. Werner. Approximating Tverberg points in linear time for any fixed dimension. In *Proceedings of the 28th ACM Symposium on Computational Geometry*, pages 303–310, 2012.
- [17] J. Richter-Gebert. *Perspectives on Projective Geometry*. Springer, 2011.
- [18] W. D. Smith and N. C. Wormald. Geometric separator theorems and applications. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 232–243, 1998.

How To Place a Point to Maximize Angles

Boris Aronov*
aronov@poly.edu

Mark V. Yagnatinsky†
myag@cis.poly.edu

Polytechnic Institute of NYU, Brooklyn, New York

Abstract

We describe a randomized algorithm that, given a set of points in the plane, computes the best location to insert a new point, such that the Delaunay triangulation of the resulting point set has the largest possible minimum angle. The expected running time of our algorithm is at most cubic on any input, improving the roughly quartic time of the best previously known algorithm.

1 Introduction

The subject of meshing and specifically constructing “well behaved” triangulations has been researched extensively [B04]. One of the problems extensively addressed in the literature is that of refining or improving an existing mesh by incremental means. Motivated by this, Aronov *et al.* [AAF10] considered the following problem: Given a set of points in the plane, where would you place one additional point, so as to maximize the smallest angle in a good triangulation of the point set. Since Delaunay triangulations are known to maximize the smallest angle over all possible triangulations with a given vertex set [S78], the question can be rephrased as: “Given a point set, where do we place an additional point, so as to maximize the minimum angle in the Delaunay triangulation of the resulting set?” In the rest of the paper we always picture the new point as lying within the convex hull of the existing points, but the algorithm is essentially the same without this simplification. (Another variant of the problem mentioned in [AAF10] involved incrementally improving an existing triangulation by “tweaking” the position of an existing interior vertex, one at a time, so that, again, the smallest angle is maximized.) They also discuss the more challenging question of how to position several points in the best possible coordinated way; we do not address this variant of the problem in the current paper.

The previous algorithm [AAF10] for placing an additional point runs in worst-case $O(n^{4+\varepsilon})$ time, for any $\varepsilon > 0$, with the constant of proportionality depending on ε . We propose a randomized algorithm whose expected running time is roughly an order of

magnitude lower. Somewhat surprisingly, Aronov *et al.* considered and rejected the approach we use in this paper [AAF10, page 96].

We present our algorithm in the following section. The analyses of this and the precursor algorithm [AAF10] are misleading in that they reflect situations unlikely to happen for “reasonable” inputs. We discuss how to quantify reasonableness of the inputs and the resulting behavior of both algorithms in section 3 and conclude in section 4.

2 The Algorithm

Our algorithm takes a set P of n points in the plane and computes the best location for a new point p , such that the Delaunay triangulation of $P \cup \{p\}$ has the largest possible minimum angle; for ease of presentation we will assume that the points of P are in *general position*, that is no three points of P lie on a line and no four on a circle. We start by recalling an argument detailed in [AAF10] which duplicates the insertion step of the standard incremental Delaunay triangulation algorithm [GS85]. Let T be the Delaunay triangulation of P . We begin by computing the arrangement \mathcal{A} induced by the Delaunay circles of P , i.e., of the circumcircles of the triangles of T . Although there are only a linear number of such circles, in the worst case every pair of them intersect, so that \mathcal{A} has quadratic complexity. We examine how the Delaunay triangulation T_p of $P \cup \{p\}$ differs from T . Let c be the face of \mathcal{A} containing p . Recall that a triangle is present in a Delaunay triangulation if and only if its Delaunay disk is empty of vertices. Point p invalidates some triangles of T by appearing in the interior of the corresponding disks. After we have inserted p , we no longer have a triangulation; instead we have a star-shaped polygonal hole H in T containing p ; see Figure 1. Since the insertion of p only invalidates previously valid triangles, but cannot make an invalid triangle valid (since insertion of p can not turn nonempty disks into empty ones), new edges of T_p must have p as an endpoint. So, connecting p to all vertices of H (Figure 1, right) is the way to complete T_p . This suggests this algorithm outline:

1. Compute the Delaunay triangulation T .
2. Build the arrangement \mathcal{A} of Delaunay circles of T .

*Supported by NSF grants CCF-11-17336 and CCF-12-18791.

†Supported by NSF grant CCF-11-17336.

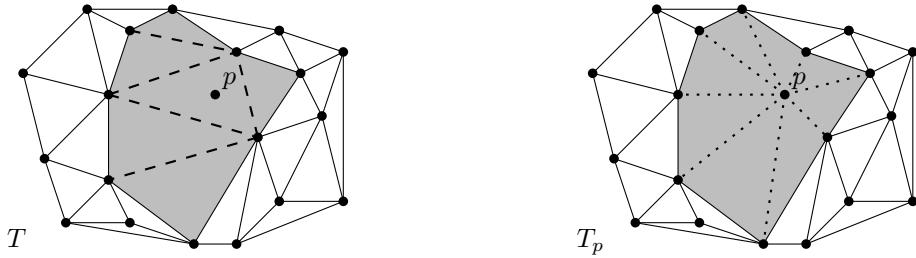


Figure 1: The new point p is in the kernel of the shaded star-shaped polygonal hole H . Removed edges of T are shown dashed (left) and added edges are dotted (right).

3. For each of the $O(n^2)$ cells $c \in \mathcal{A}$:
 - (a) Find the set of $O(n)$ triangles invalidated by placing p in c , the union of which forms the hole H .
 - (b) Optimize the placement of p in c .
4. Return the best placement of p found.

This outline was in fact used in [AAF10]. The main contribution of this paper is to use a different approach for step 3b. Specifically, in [MSW96], it was shown that the following is an LP-type problem.¹

Given a star-shaped polygon H , find the point p in its kernel that maximizes the smallest angle in the triangulation that results by connecting p to all vertices of H .

Being an LP-type problem, it can be solved in expected time linear in the number of polygon vertices, while the approach from [AAF10], based on explicitly computing lower envelopes of bivariate functions, takes time roughly quadratic in the number of vertices. However, this LP-type problem is not quite the problem we actually wish to solve, as we need the optimal placement of p *within the current cell* c , which is why this idea was rejected in [AAF10]. Fortunately, there is a conceptually simple fix. In the *region search* stage of our procedure, for each cell c , we run the algorithm from [MSW96] discarding the result if the returned optimum lies outside c . A simple argument (see Lemma 1 below) shows that if the solution to the unconstrained problem results in a point not in c , then the optimum within c must lie on its boundary. So in a separate *boundary search* step detailed below, we find the best placement of p on any cell boundary. Combining the results from the two steps we obtain the globally optimal placement for p .

Lemma 1 *If the optimum solution to the unconstrained LP-type problem corresponding to cell c is not in c , then the optimum solution for c lies on its boundary.*

Proof. Consider the locus $R(x)$ of points p such that the smallest angle in the new triangulation of H is at

¹In [ABE99], this and related problems are presented in a unified framework.

least x . It was shown in [MSW96] that $R(x)$ is a convex region; it is easy to see that it varies continuously with x , when non-empty. Clearly, $R(x) \subset R(y)$ for $y < x$. As x decreases from its optimum unconstrained value, $R(x)$ will gradually grow from a single point outside c and eventually intersect c ; as it is connected and changes continuously with x , the first intersection must occur along the boundary of c . \square

It remains to find the best placement for p on each cell boundary. A cell boundary has two sides, and we process each separately. First consider each edge of \mathcal{A} separately. For a fixed side of a fixed edge e , we know which cell of \mathcal{A} we are in, and thus the hole H . If H has h vertices, the triangulation has $3h$ angles. The measure of each of these angles is a function of the position of p . To maximize the smallest of these functions, find the maximum of their lower envelope by computing the envelope explicitly. We will show that the graphs of any pair of these functions intersect at most 16 times. A well-known result from the theory of Davenport-Schinzel sequences immediately implies that the maximum complexity $E(n)$ of the lower envelope is $O(\lambda_{16}(n))$, which is $o(n \log^* n)$, where $\lambda_s(n)$ is the maximum length of a DS(s, n) sequence [AS00]. If the worst-case complexity of the lower envelope of h functions from some class is $E(h)$, then we can compute the lower envelope of n functions from that class in $O(E(n) \log n)$ time using a simple divide-and-conquer algorithm [AS00].

Lemma 2 *The complexity of the lower envelope of n angle functions is $O(\lambda_{16}(n))$.*

Proof. There are two kinds of angles to consider: angles at the boundary of H , and angles at the new point p . We consider first angles at p . Let $p = (x, y)$, and let q and r be two consecutive vertices of H . Note that the coordinates of q and r are known constants. We are interested in the angle $\angle qpr$ at which p sees the segment qr ; see Figure 2 (left). Let s, t be another pair of consecutive vertices. The angle that p makes with the segment st is $\angle spt$. Now consider the locus of points p specified by the equation $\angle qpr = \angle spt$; a point p

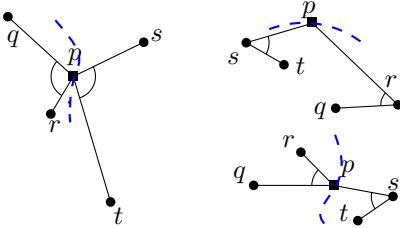


Figure 2: “Artist’s impression” of the curves defined by the three types of angle equality constraints.

satisfying this equation will see qr and st at the same angle; refer to Figure 2 (left). An intersection between this curve and an edge of \mathcal{A} corresponds precisely to an intersection of the graphs of two angle functions. Once we prove that there are at most 16 such intersections, we are done. For convenience, we will equate the cosines of the angles instead of the angles themselves. Using $|\cdot|$ to denote distances, the law of cosines gives $|qr|^2 = |pr|^2 + |pq|^2 - 2|pr||pq| \cos \angle qpr$. Solving for $\cos \angle qpr$ gives

$$\cos \angle qpr = \frac{|pr|^2 + |pq|^2 - |qr|^2}{2|pr||pq|}.$$

Setting $\cos \angle qpr$ equal to $\cos \angle spt$ produces

$$\frac{|pr|^2 + |pq|^2 - |qr|^2}{|pr||pq|} = \frac{|ps|^2 + |pt|^2 - |st|^2}{|ps||pt|}.$$

After squaring both sides and reshuffling, we get

$$(|pr|^2 + |pq|^2 - |qr|^2)^2 |ps|^2 |pt|^2 = \\ (|ps|^2 + |pt|^2 - |st|^2)^2 |pr|^2 |pq|^2.$$

Now each side of the equation is a polynomial in x and y of total degree eight. So, the question now is: how many times can a curve of degree eight intersect an edge of \mathcal{A} ? An edge is an arc of a circle, which is the zero set of a polynomial of degree two. According to Bézout’s theorem [B1779], the number of intersection points is at most the product of the degrees, so there can be at most 16 intersection points. So, the complexity of the envelope is $O(\lambda_{16}(n))$, and we are done. A similar argument is needed for $\angle qpr = \angle pst$ and also $\angle pqr = \angle pst$ (refer to Figure 2 (right)), but they also result in polynomial equations of degree at most eight; we omit the entirely analogous calculation. \square

The approach outlined above is inefficient in that there may be a quadratic number of arcs, and since we spend more than linear time on each, this would become this bottleneck of the algorithm. However, we are duplicating much work: if we follow a Delaunay circle as it crosses another circle, very little changes when we cross: either one triangle of T ceases to be valid, or else one triangle becomes valid. (This assumes that we only cross one

circle at at time. At a vertex, we may cross many circles at once, so the total change is large, but it is still true that each circle we cross does only one triangle’s worth of damage.) Suppose that a triangle becomes valid when we cross (the other case is symmetric). Then H loses a boundary vertex, and our triangulation of H loses two old triangles and gains one new one, which means our set of angle functions gains 3 new angles and loses 6 old ones. The other angle functions remain unchanged. Thus, we can define the functions over the an entire circle (provided we are consistent whether we are on the inside or outside of the circle.) On a given arc, there are at most $3n$ functions. If there are m circles, then the boundary of a fixed circle can only have $2(m - 1) < 2m$ intersections with other circles, and for each of those intersections, at most 6 new functions appear. The number of circles equals the number of triangles, which is less than $2n$. Thus for the entire circle, there are at most a linear number of functions ($2n \times 2 \times 6 + 3n \leq 27n$). It is still the case that any pair of function graphs intersect at most 16 times, but because each is not defined over the entire circle, but only a contiguous arc on it, the complexity of the lower envelope can increase slightly, up to $\lambda_{18}(n)$ [AS00]. Thus, the running time of the boundary search stage is $O(n\lambda_{18}(n)\log n)$ and the total (expected) running time is dominated by the $O(n^3)$ region search time. (The boundary search can be sped up slightly to $O(n\lambda_{17}(n)\log n)$ by using the algorithm of Hershberger [H89].)

3 Realistic inputs

In the long tradition in computational geometry, exemplified by [BKSV02], we would like to be able to analyze our problem in non-worst-case situations. To this end, we introduce several parameters, besides n that measures the number of input points, that quantify the “badness” of the input point set and express the running time of the algorithms in terms of them.

Consider the arrangement \mathcal{A} of Delaunay disks of P and let k be its complexity, that is the total number of vertices, edges, and faces; let d be the maximum *depth* of the arrangement, that is the maximum, over all points in the plane, of the number of disks covering the point. In the worst case k is $\Theta(n^2)$ and d is $\Theta(n)$. In well-behaved point sets, such as those corresponding to uniformly distributed points, k is $\Theta(n)$; one would also expect d to be near-constant, however, somewhat surprisingly, an unfortunate, but arbitrarily small perturbation of the $\sqrt{n} \times \sqrt{n}$ grid can cause d to be $\Theta(\sqrt{n})$ (we omit the details in this version).

We now express the running times in terms of n , k , and d . Our algorithm starts by computing the Delaunay triangulation, which can be done in $O(n \log n)$ time. We then compute the arrangement of circles in $O(k \log n)$

time using a standard sweepline algorithm (better running times are possible using more involved techniques). Our algorithm and that of [AAF10] share the first two steps of the outline. Their analog of the region search runs in time $O(kd^{2+\varepsilon})$, for any positive ε , since for every cell $c \in \mathcal{A}$, it performs an independent bivariate lower envelope calculation on $O(d)$ functions, for a total time of $O(kd^{2+\varepsilon} + k \log n)$. We analyze the region search and the boundary search stages of our proposed algorithm separately. The region search runs in expected time $O(kd)$, as its bottleneck is solving k LP-type problems of size at most d each. (Note that this requires that we quickly determine the set of constraints that correspond to a cell. This is easy to arrange if we visit adjacent cells in order.)

We now turn our attention to the boundary search. Our analysis here needs stronger general position assumptions than the algorithm itself does. In particular, we require that if two circles intersect in some point not in P , no third circle goes through that point.

The running time for one circle is affected by how many functions we need to take the lower envelope of along that circle. We earlier derived a bound of $27n$ for the number of functions on a given circle. We now make this more precise. Let f_i denote the number of functions along circle C_i . If C_i intersects x_i other circles, and further is not adjacent to any cell having depth more than d_i , then by our previous analysis $f_i \leq 3(d_i + 1) + 12x_i$. Note that since these circles are Delaunay, no disk fully contains another. Hence, any circle containing a cell of large depth must intersect many other circles. In particular, $x_i \geq d_i - 1$. Thus, we have $f_i \leq 3(d_i + 1) + 12x_i \leq 3(x_i + 2) + 12x_i = 15x_i + 6$, which is $O(x_i)$.

We now show that the sum of x_i over all circles is at most proportional to the arrangement complexity k . Note first that this sum is simply twice the number of pairs of intersecting circles. Our approach will thus be to show that most pairs of intersecting circles contribute a vertex of degree 4 to the arrangement \mathcal{A} , that is, a vertex that no third circle goes through. Indeed, consider a pair of intersecting circles such that both intersection points, call them p and q , have degree at least 6 (in a circle arrangement, all vertices have even degree). By our stronger general position assumption, both p and q are from the original point set P . We now have a pair of points with two Delaunay circles passing through it: hence pq must be a Delaunay edge! But there are only a linear number of such edges, so we are done: all but $O(n)$ pairs of intersecting circles contribute a new vertex to the arrangement.

Finally, let m be the number of circles, X be the number of pairs of intersecting circles, u be the number of vertices of degree 4, and e be the number of edges of the Delaunay triangulation. We now bound the sum of x_i over all circles: $\sum_{i=1}^m x_i = 2X \leq 2(u + e) = 2u + 2e <$

$$2k + 2e \leq 2k + 2(n + m - 2) \leq 2k + 2(m + 2 + m - 2) = 2k + 4m < 2k + 4k = 6k, \text{ which is } O(k).$$

Lastly, the total running time of the boundary search stage is at most proportional to:

$$\begin{aligned} \sum_{i=1}^m \lambda_{18}(x_i) \log x_i &\leq \sum_{i=1}^m \lambda_{18}(x_i) \log m \\ &= \log m \cdot \sum_{i=1}^m \lambda_{18}(x_i) \\ &\leq \log m \cdot \lambda_{18}(\sum_{i=1}^m x_i) \\ &\leq \lambda_{18}(6k) \log m, \end{aligned}$$

which is $O(\lambda_{18}(k) \log n)$, and thus the running time of our entire algorithm is $O(kd + \lambda_{18}(k) \log n)$. Therefore, our algorithm outperforms (in expectation) that of [AAF10] for all values of k and d .

We can slightly refine the above analysis in another direction: Recall that we defined d to be the *maximum* depth of the arrangement \mathcal{A} . If we let \bar{d} be the *average* depth, over all the cells, the running time of the region search can then be bounded by $O(k\bar{d})$, while the running time of the analogous part of algorithm of [AAF10] is $O(\sum_{c \in \mathcal{A}} d_c^{2+\varepsilon})$, where d_c is the depth of cell c ; the latter quantity is, roughly, k times the average *squared* depth. The running time of the boundary search is not easily expressed in terms of \bar{d} , but it is less likely to dominate the running time of our algorithm.

It would be interesting to connect the parameters d and k (and ultimately the behavior of the algorithms) to the more commonly used measures of how well-behaved a point set in the plane is, such as its *spread*, which is the ratio between the largest and the smallest interpoint distances.

4 Conclusions and open problems

We believe our algorithm can be easily modified to work with constrained Delaunay triangulations, which was the original setting of [AAF10]; we omit the extension in this version. (The key differences are that the set of invalidated triangles depends on the constraints, and that the arrangement \mathcal{A} is formed by circles and the constraining segments.)

It would be interesting to see if our algorithm can be derandomized using the results of Chazelle and Matoušek [CM93]; the LP-type problem needs to meet some technical requirements the discussion of which is omitted here.

Can the algorithm be sped up by roughly another order of magnitude by observing that there is generally very little difference between LP-type problems corresponding to adjacent cells of \mathcal{A} ?

Is there any hope of generalizing our approach to multiple Steiner points as in [AAF10]?

References

- [AS00] P.K. Agarwal and M. Sharir. Davenport-Schinzel sequences and their geometric applications. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds., 1–47. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
- [ABE99] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *J. Algorithms* 30(2), 302–322, 1999.
- [AAF10] B. Aronov, T. Asano, and S. Funke. Optimal triangulations of points and segments with Steiner points. *Int. J. Comput. Geom. Appl.*, 20(1) 89–104, 2010.
- [BKSv02] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34:81–97, 2002.
- [B04] M. Bern. Triangulations and mesh generation. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, J. E. Goodman and J. O’Rourke, Eds., 563–582. CRC Press LLC, Boca Raton, FL, April 2004.
- [B1779] Bézout theorem. Wikipedia. From http://en.wikipedia.org/wiki/B%C3%A9zout%27s_theorem; retrieved 11 May 2013.
- [CM93] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Proc. Fourth Annu. ACM-SIAM Symp. Discr. Algorithms*, pp. 281–290, 1993.
- [GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2) 74–123, 1985.
- [H89] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inf. Proc. Letters*, 33(4) 169–174, 1989.
- [MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4–5) 498–516, 1996.
- [S78] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3) 243–245, 1978.

Spanning Colored Points with Intervals

Payam Khanteimouri * Ali Mohades* Mohammad Ali Abam † Mohammad Reza Kazemi*

Abstract

We study a variant of the problem of spanning colored objects where the goal is to span colored objects with two similar regions. We dedicate our attention in this paper to the case where objects are points lying on the real line and regions are intervals. Precisely, the goal is to compute two intervals together spanning all colors. As the main ingredient of our algorithm, we first introduce a kinetic data structure to keep track of minimal intervals spanning all colors. Then we present a novel algorithm using the proposed KDS to compute a pair of intervals which together span all the colors with the property that the largest one is as small as possible. The algorithm runs in $O(n^2 \log n)$ using $O(n)$ space where n is the number of points.

1 Introduction

Background. In the view of location planning, suppose there are n facilities with k types like banks, network access points, etc. in the plane. Each type t can be represented by a unique color $c(t)$, i.e., each facility of type t is colored with $c(t)$. In some applications, accessing to one representative of each type suffices which means location planning is defined based on types rather than facilities in these applications. One basic problem arises here is to find a location where there is at least one representative of each type in its nearby. This suggests the problems of computing the *smallest area/perimeter color-spanning objects*. A region is said to be *color-spanning* if it contains at least one point from each color.

The other motivation specially for 1D colored points comes from planning a toolpath in layered manufacturing [4, 11]. In this application, a 3D object is defined by its layers in the plane and each layer consists of contours or polygons. To construct an object, a toolpath like a laser beam cuts the boundary of contours one by one. In practice, the laser beam moves in a straight line from one contour to other and each contour is traced only once. A trace path contains all contours and the line segments connecting them. Since the straight lines

indicate the wasted time which is significant, toolpath planning is computing the trace path which minimize the total wasted time. In particular, when each contour is almost a single point, the problem is computing the TSP and clearly is NP-Hard. To obtain a heuristic method, Tang and Pang [11] proposed an algorithm in which they compute the *smallest color-spanning interval* for a set of colored points on the real line. Beside these two motivations, studying on spanning colored points has other applications in imprecise data, statistical clustering, pattern recognition and generalized range searching [6, 7, 9, 10].

Since the main ingredient of algorithms is a KDS for maintaining the minimal color-spanning intervals, we here sketch an overview of Kinetic Data Structures. A *kinetic data structure (KDS)* is a structure for keeping the trajectory of an *attribute* e.g. the sorting, for moving objects. We define a set of *certificates* for a KDS which are some boolean functions. Indeed, the set of certificates is a *proof scheme* for the attribute which means the validity of all certificates leads to correctness of the attribute. Therefore, when an event occurs, i.e., a certificate fails, we should update the KDS. Thus, we use a priority queue like a min heap to store the failure times of the events. A KDS is analysed with the following concepts. We say a KDS is *compact* if it totally uses $O(n \text{ polylog}(n))$ space and is *local* if each object participates in $O(\text{polylog}(n))$ certificates. In addition, a KDS is *responsive* if it can be updated in $O(\text{polylog}(n))$ time when an event occurs. The event which changes the attribute is an *external event* and any KDS should be updated in its failure time. Moreover, there may be an *internal event* which means our KDS should be updated while the attribute remains unchanged. A KDS is said to be *efficient* if the ratio of the number of all events and the number of external events is $O(\text{polylog}(n))$.

Related works. In the view of imprecise data, for a set of n points with k colors, the main problem is computing exactly k points with different colors in which a property like diameter, closest pair, and etc. is minimized/maximized. C. Fan et al. [6] showed that the problem of computing the *largest closest pair* is NP-Hard under the L_p metric ($1 \leq p < \infty$) even for 1D points. They [6] also proposed an algorithm with $O(n \log n)$ expected time for computing the *maximum diameter*.

In the view of location planning, for a given set of

*Laboratory of Algorithms and Computational Geometry, Department of Mathematics and Computer Science, Amirkabir University of Technology (Tehran Polytechnic), {p.khanteimouri,mohades,mr.kazemi}@aut.ac.ir

†Department of Computer Engineering, Sharif University of Technology, abam@sharif.edu

n colored points with k colors in the plane, computing *the smallest color-spanning axes parallel rectangle* is the most studied problem. Abellanas et al. [1] proposed an algorithm of $O((n - k)^2 \log^2 k)$ time and $O(n)$ space while they showed there are $\Theta((n - k)^2)$ minimal color-spanning rectangles in the worst case. Das et al. [5] have recently improved the running time to $O(n^2 \log n)$. They [5] also present an algorithm in $O(n^3 \log k)$ time and $O(n)$ space to solve the arbitrary oriented case. Another studied problem by these papers is computing *the smallest color-spanning strip*. Das et al. [5] propose an algorithm in $O(n^2 \log n)$ time and $O(n)$ space using the dual of the given points to solve the problem. The results are near efficient with respect to testing all minimal objects. Recall that, a *minimal color-spanning object* contains at least one point from each color and any sub-region of it does not contain all colors.

In addition, Abellanas et al. [2] defined *the farthest colored Voronoi diagram (FCVD)*. For a set of n colored sites with k colors in the plane, the FCVD is the subdivision of the plane in which for any region R there is a unique site p such that any color-spanning circle centered at a point in R must contain p . Therefore, to compute the *smallest color-spanning circle* a simple algorithm is to compute the FCVD and test circles centered at the vertices of FCVD. They [2] proposed an algorithm with $O(n^2 \alpha(k) \log k)$ time to compute the FCVD and the smallest color-spanning circle. The other approach mentioned by Abellanas et al. [1] is to obtain the smallest color-spanning circle and the *smallest color-spanning axes-parallel square* in $O(kn \log n)$ time and $O(n)$ space using the upper envelope of Voronoi surfaces [8].

Computing the smallest color-spanning interval has been widely studied by Chen and Misolek [4]. For a set of n points with k colors on the real line they [4] showed that minimal color-spanning intervals form a strictly increasing sequence. Due to this property they proposed two algorithms for computing the smallest color-spanning interval. The first algorithm simply compute the smallest color-spanning interval by a left to right sweeping in $O(n)$ time and space apart from sorting. Next, they assumed that points are given one by one in a sorted order and each point must be processed only once which is suitable for an online processing. They [4] proposed an algorithm of $O(n)$ time and $O(k)$ space.

Our results. In this paper, we study on spanning a set of n points with k colors on the real line by two intervals. We first assume points are moving on \mathbb{R}^1 . We design a kinetic data structure for keeping the track of all minimal color-spanning intervals. We show our KDS is efficient, responsive, local and compact. Next, we use this result to propose an algorithm to compute two intervals which together span all colors and the largest one is as small as possible. The algorithm runs in $O(n^2 \log n)$ time and $O(n)$ space.

This paper is organized as following. In Section 2 we show how to keep track of all minimal color-spanning intervals for a set of moving colored points on \mathbb{R}^1 . Next, in Section 3 we propose an almost efficient algorithm to compute two intervals together spanning all colors and the largest one is as small as possible. Finally we conclude in Section 4.

2 Minimal Color-Spanning Intervals for Moving Points

We first pay our attention to static points on the real line and then switch to moving points where the trajectory of each colored point is a polynomial function with degree at most s . For moving points, we are interested in maintaining all minimal color-spanning intervals. Since the problem is trivial for $k = 2$ we assume $k > 2$.

Static Points. Let $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ be a set of n colored points with k colors on the real line. We assume points are in general position which means no two points coincide. A *minimal color-spanning interval (MCSI)* is an interval containing all colors and any sub-interval of it, does not contain all colors. From the definition, we can immediately deduce that the colors of the start point and the end point of an MCSI are different and unique in the interval. We first present the following lemma.

Lemma 1 *For a set of n points with k colors on the real line, there are at most $n - k + 1$ minimal color-spanning intervals.*

Proof. Let p_i be the start point of an MCSI $[p_i, p_j]$. Obviously, p_i cannot be the start point of another MCSI $[p_i, p'_j]$ due to the minimality of both intervals. Therefore, each MCSI can be uniquely charged to its start point. On the other hand, the start point cannot be among the $k - 1$ rightmost points as MCSI must contain at least k points. These together show the number of MCSIs is at most $n - k + 1$. To give a tight lower bound, consider the sequence $1, 2, \dots, k$ repeated $\frac{n}{k}$ times. The number of MCSIs in this example obviously is $n - k + 1$. \square

To compute the MCSIs, it suffices to sweep the points from left to right with two sweep lines. The sweep lines stop at the start and the end points of an interval. To recognize if the sweep lines indicate an MCSI, we use an array for keeping the number of points from each color and a variable for the number of different colors between the two sweep lines. From the fact that both sweep lines go forward in each step, the algorithm runs in $O(n)$ time and space apart from sorting. We avoid the details due to the simplicity and conclude the following lemma.

Lemma 2 For a set of n points with k colors on the real line, all minimal color spanning intervals can be computed in $O(n)$ time and space apart from sorting.

Moving Points. We now concentrate on maintaining MCSIs while points are moving continuously on the real line. Let $p(t)$ be the position of point p at time t . We define the following ordered sets:

- $\mathcal{P}(t) = \{p_{i_1}, \dots, p_{i_n}\} ; p_{i_1}(t) < \dots < p_{i_n}(t),$
- $M(t) = \{[p_i, p_j] \mid [p_i, p_j] \text{ is an MCSI at time } t\},$
- $\mathcal{S}(t) = \{p_i \mid [p_i, p_j] \in M(t)\},$
- $\mathcal{E}(t) = \{p_j \mid [p_i, p_j] \in M(t)\}.$

Indeed, $\mathcal{P}(t)$ is the increasingly ordered list of the moving points according to their positions at time t . Moreover, $M(t)$ is the set of all MCSIs $[p_i, p_j]$ at time t . Since for any two MCSIs $[p_i, p_j]$ and $[p_s, p_t]$, we have either $p_i < p_s$ and $p_j < p_t$ or $p_s < p_i$ and $p_t < p_j$ due to the minimality, $M(t)$ can be recognized as an ordered list based on MCSI's start points. We have also stored the start and respectively the end points of MCSIs at time t in distinct sets $\mathcal{S}(t)$ and $\mathcal{E}(t)$. In addition, let $c(p)$ denotes the color of point p , $\text{pred}(p)$ and $\text{suc}(p)$ be the previous and respectively the next point of p with color $c(p)$.

Now, we show how $M(t)$ changes while the points are moving. It is obvious while the sorted list of points, $\mathcal{P}(t)$, does not change which means there is no swap between two consecutive points in $\mathcal{P}(t)$, $M(t)$ remains unchanged. Therefore, we maintain a kinetic sorting for moving points in \mathcal{P} and explain how to handle an event in the kinetic sorting where two consecutive points p and q swap.

To handle events, we start with a useful lemma. Suppose there are two MCSIs $I_i = [p_i, p]$ and $I_j = [p_j, q]$ in $M(t)$ such that p and q are consecutive points in $\mathcal{P}(t)$ ($p < q$) —see Figure 1. Therefore, I_i and I_j should also be consecutive in $M(t)$. Since the intersection of I_i and I_j contains exactly $k - 1$ colors (all colors except $c(q)$), the color of point p_i should be the same as the color of q , i.e., $c(p_i) = c(q)$. In addition, $p_i = \text{pred}(q)$ which means p_i is the previous point of q with color $c(q)$. Put together we obtain the following result.

Lemma 3 Suppose there are two intervals $I_i = [p_i, p]$ and $I_j = [p_j, q]$ in $M(t)$ such that p and q are consecutive in \mathcal{P} , then $c(p_i) = c(q)$ and $p_i = \text{pred}(q)$.

Now, we concentrate on cases in which two consecutive points p and q swap their positions in $\mathcal{P}(t)$. In all cases, when $M(t)$ changes we update the sets $\mathcal{S}(t)$ and $\mathcal{E}(t)$. The arising cases are as following.

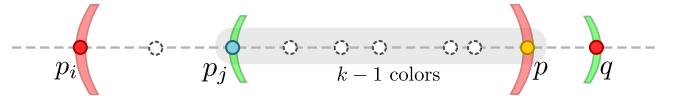


Figure 1: MCSIs $I_i = [p_i, p]$ and $I_j = [p_j, q]$ for two consecutive points p and q .

1. $p \notin \mathcal{S}(t) \cup \mathcal{E}(t)$ and $q \notin \mathcal{S}(t) \cup \mathcal{E}(t).$

In this case, $M(t)$ does not change. If $c(p) = c(q)$, we update pred and suc of p , q and the adjacent points.

2. $p \notin \mathcal{S}(t) \cup \mathcal{E}(t)$ and $q \in \mathcal{S}(t) \cup \mathcal{E}(t).$

Let I be the MCSI whose one of endpoints is q . We can distinguish 4 sub-cases. In sub-cases (a) and (b) p is inside I before the event and in sub-cases (c) and (d) p is outside I before the event.

(a) p is inside $I = [p_i, q]$ and is the only point with color $c(p)$ in I . Since p is not the end point of an MCSI, according to Lemma 3 there is no point v in the left of p_i such that $c(v) = c(q)$. In this case, if there is a point u with color $c(p)$ in the left of p , precisely $u = \text{pred}(p)$, we first update $[p_i, q]$ to $[p_i, p]$. Then, we insert MCSI $[u, q]$ in $M(t)$ —see Figure 2(a). If u does not exist no new MCSI is inserted.

(b) p is inside I and there is a point u with color $c(u) = c(p)$ in I . In this case $M(t)$ does not change —see Figure 2(b).

(c) p is outside $I = [p_i, q]$ and $c(p) \neq c(q)$. since I is an MCSI, there should be point u inside I with color $c(u) = c(p)$ —see Figure 2(c). Recall that $u \neq p_i$ according to Lemma 3. Therefore, $M(t)$ remains unchanged.

(d) $c(p) = c(q)$ —see Figure 2(d). In this case, we first update pred and suc of p , q and adjacent points. Then, we change the interval $[p_i, q]$ to $[p_i, p]$ in $M(t)$.

3. $p \in \mathcal{S}(t) \cup \mathcal{E}(t)$ and $q \notin \mathcal{S}(t) \cup \mathcal{E}(t).$

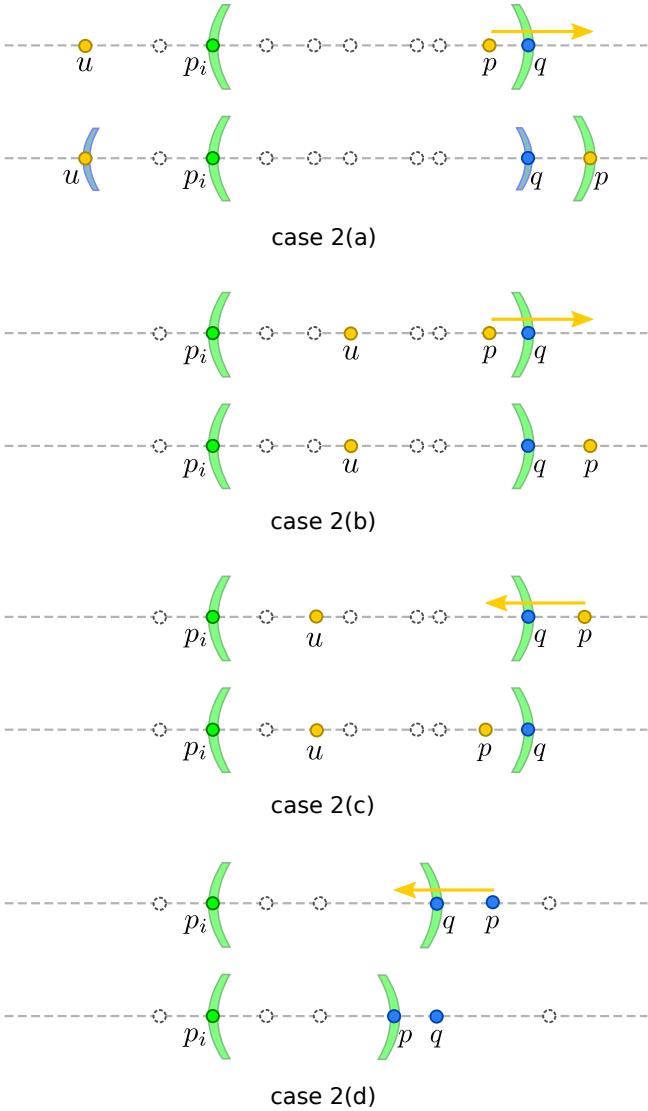
This case can be handled in the same manner in case 2.

4. $p \in \mathcal{S}(t) \cup \mathcal{E}(t)$ and $q \in \mathcal{S}(t) \cup \mathcal{E}(t).$

We can distinguish two cases based on whether $p, q \in \mathcal{E}(t)$ or $p \in \mathcal{S}(t)$ and $q \in \mathcal{E}(t)$.

(a) $p, q \in \mathcal{E}(t)$. As Figure 3(a) illustrates, since $[p_i, p]$ is not an MCSI after the swap, we first remove the interval $[p_i, p]$ from $M(t)$ and then we update the interval $[p_j, q]$ to $[p_j, p]$.

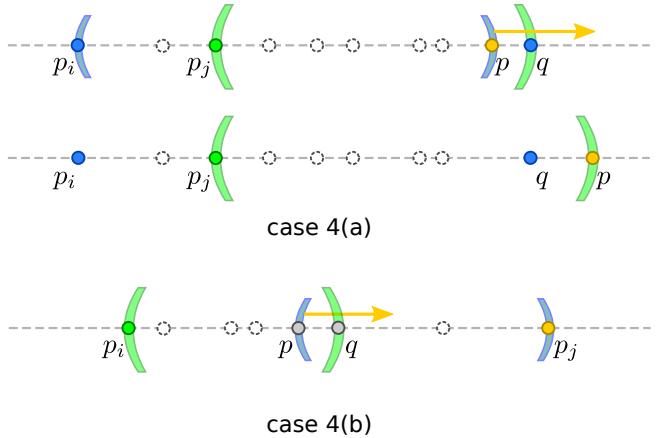
(b) $p \in \mathcal{S}(t), q \in \mathcal{E}(t)$. To handle this case, p affects the interval $[p_i, q]$ as an ordinary point.

Figure 2: $p \notin \mathcal{S}(t) \cup \mathcal{E}(t)$ and $q \in \mathcal{S}(t) \cup \mathcal{E}(t)$.

Thus, we consider p as a point inside the interval $[p_i, q]$ and handle the event based on case 2. This happens similarly for q which is inside the interval $[p, p_j]$ —see Figure 3(b). Therefore, we handle two events in the type of case 2 instead of this case. The case $p \in \mathcal{E}(t), q \in \mathcal{S}(t)$ can be handled similarly.

Note that since a point can simultaneously be start and end point of MCSIs, more than one of the above cases may be handled when one event happens. As each point can appear once as the start or the end point of MCSIs, there are constant arising cases in an event and all of them can be independently handled as described in the above cases without priority.

To test whether $p \in \mathcal{S}(t)$ in $O(\log n)$ time, we maintain a dynamic search tree (like a red-black tree) over

Figure 3: $p \in \mathcal{S}(t) \cup \mathcal{E}(t)$ and $q \in \mathcal{S}(t) \cup \mathcal{E}(t)$.

$\mathcal{S}(t)$ supporting deletion and insertion in $O(\log n)$ time. In each event-handling we update this tree by performing a constant number of deletions and insertions. We also need a similar search tree over $\mathcal{E}(t)$ to test whether $p \in \mathcal{E}(t)$.

Since we just use a kinetic sorting over \mathcal{P} , our KDS is obviously compact and local. Moreover, as described above, each event can be handled in $O(\log n)$ time. In the worst cast, we handle $O(n^2)$ events under the assumption that the trajectory of each point is a bounded degree polynomial. Putting all together we conclude the following theorem.

Theorem 4 *For a set of n moving colored points with k colors on the real line, there is an efficient, responsive, local and compact KDS which keeps the track of all minimal color-spanning intervals.*

Proof. In order to show our KDS is efficient, we give a configuration of moving points where MCSIs change $\Omega(n^2)$ times when trajectories are bounded degree polynomials. Consider $\frac{n}{2}$ static points with color 1 and $\frac{n}{2}$ moving points with color 2 passing through all static points. When two points of different colors swap, definitely an MCSI changes. Therefore, the total number of external events in this example is $\Omega(n^2)$ as any point with color 1 swaps with any point with color 2. This shows our KDS is efficient as it handles $O(n^2)$ events in the worst case. \square

3 Computing the Smallest Color-Spanning Two Intervals

We now present our novel algorithm to compute *the smallest color-spanning two intervals (SCS2I)* which is two intervals together spanning all colors with the largest one as small as possible.

We first give a naive algorithm to compute SCS2I. For each interval $[p_i, q_j]$ we can compute the smallest interval $[p_s, p_t]$ which spans the colors that are not appeared in $[p_i, p_j]$ in $O(n)$ time using Lemma 2. As there are $O(n^2)$ different intervals, this algorithm runs in $O(n^3)$ time which is far from being efficient.

We next present our main algorithm that uses the KDS described in the previous section. Suppose \mathcal{P} is a set of n points on the real line, each associated with one of the given k colors. Let \mathcal{P}' be obtained by translating \mathcal{P} by a vector \vec{d} to the right such that all points of \mathcal{P} are left to all points of \mathcal{P}' — see Figure 4. Now, consider the kinetic maintenance of MCSIs of the set $\mathcal{P} \cup \mathcal{P}'$ where points in \mathcal{P} are static and points in \mathcal{P}' move all with the same speed to the left.

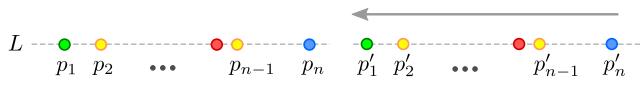


Figure 4: The copied points, \mathcal{P}' , move through the static original points.

Let $I(t)$ be the smallest color-spanning interval of $\mathcal{P}(t) \cup \mathcal{P}'(t)$ at time t and set I to be the smallest $I(t)$ for all t . We first show the length of I , the difference of its two endpoints denoted by $|I|$, is the solution to SCS2I and then we explain how to compute I during the kinetic maintenance of MCSIs of $\mathcal{P} \cup \mathcal{P}'$.

Suppose intervals $I_1 = [p_i, p_j]$ and $I_2 = [p_s, p_t]$ are the solution to the problem of SCS2I such that $|I_1| > |I_2|$ — see Figure 5(a). We first give the following lemma.

Lemma 5 *The length of the smallest color-spanning interval over all time, I , is equal to the length of I_1 .*

Proof. Let $|I|$ be the length of the smallest color-spanning interval I during the movement of points. Since $\mathcal{P}'(t)$ is the same as $\mathcal{P}(t)$, the intervals $I'_1 = [p'_i, p'_j]$ and $I'_2 = [p'_s, p'_t]$ in $\mathcal{P}'(t)$ are also the solution to SCS2I — see Figure 5(b). Now, consider the time in which p_j and p'_t are swapped. Since the color of the endpoints of I_1 and I'_2 are unique in both intervals and together span all colors, the interval $I_1 = [p_i, p_j]$ becomes an MCSI after the swap — see Figure 5(c) for more illustration. Therefore, we conclude $|I| \leq |I_1|$.

To prove $|I| \geq |I_1|$, for the sake of contradiction assume $|I| < |I_1|$. Now, consider the time t when the length of the smallest color-spanning interval is $|I|$. Since I consists of points in $\mathcal{P}(t) \cup \mathcal{P}'(t)$ we can define two sub-intervals over the points in $\mathcal{P}(t)$ and respectively $\mathcal{P}'(t)$ which together span all colors and the length of the largest one is I which is smaller than $|I_1|$. This leads us to a better solution which is a contradiction. Therefore, we have $|I| = |I_1|$. \square

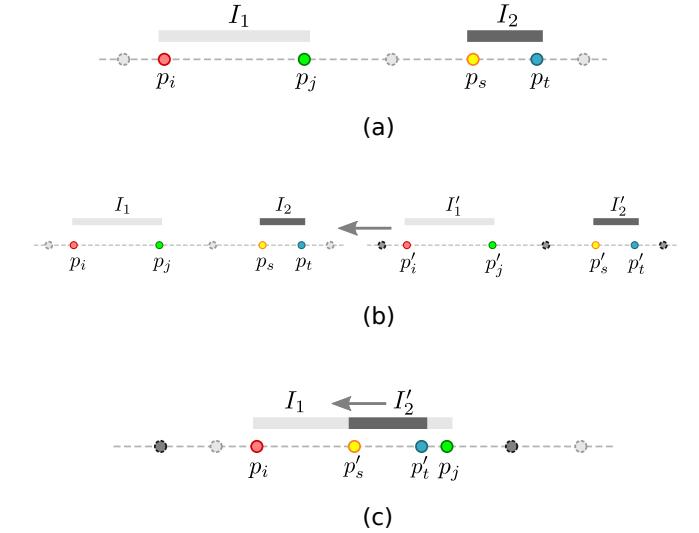


Figure 5: Intervals I_1 and I'_2 becomes an MCSI after swapping the points p_j and p'_t .

Note that the smallest color-spanning interval for all time, I , is appeared at least two times; precisely at times $I'_2 \subset I_1$ and $I_2 \subset I'_1$.

We now explain how to compute I , minimum over all $I(t)$. In general, we can maintain I using a kinetic tournament over all MCSIs at current time. Note that even there is no swap in the kinetic sorting, interval I can combinatorially change. The kinetic maintenance of $I(t)$ can be simply done by putting a kinetic tournament over intervals in $M(t)$. It is straightforward to show the kinetic sorting together with the kinetic tournament handle $O(\lambda_{s+2}(n^2) \log^2 n)$ events where s is the maximum degree of the polynomials describing the motions [3]. As we just need I , the minimum interval over all times, we can do faster as follows. Fortunately, in our setting where points of \mathcal{P} are static and points of \mathcal{P}' move with the same speed to the left, an MCSI reaches its minimum length when it appears or disappears from $M(t)$. Appearance or disappearance of MCSIs happens at event times where two points swap. Therefore, it suffices to take the minimum over all MCSIs' lengths at their appearance or disappearance times. This obviously can be done in $O(n^2)$ time.

Theorem 6 *For a given set of n colored points with k colors on the real line, the smallest color-spanning two intervals can be computed in $O(n^2 \log n)$ time and $O(n)$ space.*

In addition, We can show if the color of given points is a sequence of $1, 2, \dots, k$ repeated $\frac{n}{k}$ times, there are $\Omega(n^2)$ minimal color-spanning two intervals. So, our algorithm is near efficient with respect to testing all minimal color-spanning two intervals.

4 Conclusion

For a set of n colored points with k colors on the real line, first the problem of keeping the track of minimal color-spanning intervals is studied in this paper. We present a kinetic data structure which it is efficient, responsive, local and compact. Then, we use this result to compute two intervals which they span all colors together and the length of the largest one is minimum. This is a novel idea which solves a static problem from the kinetic interpretation of it. We propose an algorithm which compute the smallest color-spanning two intervals in $O(n^2 \log n)$ time and $O(n)$ space.

References

- [1] M. Abellanas and F. Hurtado and C. Icking and R. Klein and E. Langetepe and L. Ma and B. Palop and V. Sacristán. Smallest Color-Spanning Objects. *ESA, Springer-Verlag*, 278–289, 2001.
- [2] M. Abellanas and F. Hurtado and C. Icking and R. Klein and E. Langetepe and L. Ma and B. Palop and V. Sacristán. The Farthest Color Voronoi Diagram and Related Problems. *tech. report. University of Bonn.*, 2006.
- [3] J. Basch. *Kinetic Data Structures*. PhD thesis, 1999.
- [4] D. Z. Chen and E. Misiolek. Algorithms for interval structures with applications. *Proceedings of the 5th joint international frontiers in algorithmics, FAW-AAIM'11, Springer-Verlag*, 196–207, 2011.
- [5] S. Das and P. P. Goswami and S. C. Nandy. Smallest Color-Spanning Object Revisited. *Int. J. Comput. Geometry Appl.*, 19:457–478, 2009.
- [6] C. Fan and W. Ju and J. Luo and B. Zhu. On some geometric problems of color-spanning sets. *Proceedings of the 5th joint international frontiers in algorithmics, and 7th international conference on Algorithmic aspects in information and management. FAW-AAIM'11. Springer-Verlag*, 113–124, 2011.
- [7] P. Gupta and R. Janardan and M. Smid. Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization. *J. Algorithms.*, 19(2):282–317, 1995.
- [8] D. P. Huttenlocher and K. Kedem and M. Sharir. The Upper Envelope of voronoi Surfaces and Its Applications. *Discrete Computational Geometry*, 9:267–291, 1993.
- [9] J. Matoušek. On enclosing k points by a circle. *Information Processing Letters*, 53(4):217–221, 1995.
- [10] M. Smid. Finding k points with a smallest enclosing square. *MPI-I-92-152, Max-Planck-Institut Inform., Saarbrücken, Germany*, 1992.
- [11] K. Tang and A. Pang. Optimal connection of loops in laminated object manufacturing. *Computer-Aided Design*, 35(11):1011-1022, 2003.

On Fence Patrolling by Mobile Agents

Ke Chen* Adrian Dumitrescu† Anirban Ghosh‡

Abstract

Suppose that a fence needs to be protected by k mobile agents with maximum speeds v_1, \dots, v_k so that each point on the fence is visited by some agent within every duration of a predefined time. The problem is to determine if this requirement can be met, and if so, to design a suitable schedule for the agents. Alternatively, one would like to find a schedule that minimizes the *idle time*, that is, the longest time interval during which some point is not visited by any agent. The problem was introduced by Czyzowicz et al. (2011). We revisit this problem and discuss several strategies for the cases of open and respectively closed fence.

1 Introduction

A set of mobile agents with predefined (possibly distinct) maximum speeds are in charge of *guarding* or in other words *patrolling* a given region of interest. Two interesting uni-dimensional variants where the agents move along a curve (e.g., the boundary of the region), have been introduced by Czyzowicz et al. [1]: (i) only part of the boundary, that is, an open curve, or *open fence*, needs to be guarded; (ii) the entire boundary, that is, a closed curve (cycle), or *closed fence*, needs to be guarded. For simplicity (and without loss of generality) it can be assumed that the open curve is a segment and the closed curve is a circle.

Given a schedule of the agents over some time interval $[0, t]$, the *idle time* I is the longest time interval during which a point of the fence remains unvisited, taken over all points. We are interested in guarding over an unlimited time interval, i.e., over the interval $[0, \infty)$. If the schedule of the agents is such that the positions of the agents during the time intervals $[it_0, (i+1)t_0]$, $i = 0, 1, \dots$, are the same functions of t , we say that the schedule is *periodic* with period t_0 .

Given k agent speeds $v_1, \dots, v_k > 0$, the goal is to find a schedule for which the idle time is minimum. A straightforward volume argument from [1] yields the

*Dept. of Comp. Sci., Univ. of Wisconsin–Milwaukee, USA.
Email: kechen@uwm.edu.

†Dept. of Comp. Sci., Univ. of Wisconsin–Milwaukee, USA.
Email: dumitres@uwm.edu. Supported in part by NSF grant DMS-1001667.

‡Dept. of Comp. Sci., Univ. of Wisconsin–Milwaukee, USA.
Email: anirban@uwm.edu. Supported by NSF grant DMS-1001667.

lower bound $I \geq 1 / \sum_{i=1}^k v_i$. This lower bound applies for both the segment and the circle variant of the problem, and for any speed setting.

For the segment variant, Czyzowicz et al. [1] proposed a simple partitioning strategy, algorithm \mathcal{A}_1 , where each agent moves back and forth in a segment whose length is proportional with its speed. Algorithm \mathcal{A}_1 is *universal* in the sense that is applicable for any speed setting $v_1, \dots, v_k > 0$ for the agents. \mathcal{A}_1 has been proved to be optimal for uniform speeds [1], i.e., when all maximum speeds are equal. It has been conjectured [1] that it is optimal for any speed setting, however this was recently disproved by Kawamura and Kobayashi [2] with two examples (periodic schedules) that only barely invalidate the conjecture. It is worth mentioning that the idle time achieved by \mathcal{A}_1 is $2 / \sum_{i=1}^k v_i$ and thereby \mathcal{A}_1 yields a 2-approximation algorithm for the shortest idle time. The current best lower bound examples have an idle time of about $0.98 \left(2 / \sum_{i=1}^k v_i\right)$.

For the circle variant, no universal algorithm has been proposed to be optimal. However, if the maximum speeds of the agents are the same, i.e., $v_1 = \dots = v_k = v$, then placing the agents uniformly around the circle and moving in the same direction yields the minimum idle time for this setting. Indeed, the idle time is $1 / (kv) = 1 / \sum_{i=1}^k v_i$, matching the lower bound mentioned earlier.

Under the restriction that all agents must move in the same, say clockwise direction, Czyzowicz et al. [1] conjectured that the following algorithm \mathcal{A}_2 is optimal: Let $v_1 \geq v_2 \geq \dots \geq v_k$. Let r be such that $\max_{1 \leq i \leq k} iv_i = rv_r$. Place the agents a_1, a_2, \dots, a_r at equal distances of $1/r$ around the unit circle, each moving clockwise at the same speed v_r . Discard the remaining agents, if any. Since all agents move in the same direction, we also refer to \mathcal{A}_2 as the “runners” algorithm. Observe that \mathcal{A}_2 is also universal. Its idle time is $1 / \max_{1 \leq i \leq k} iv_i$ [1, Theorem 2]. The conjectured optimality of \mathcal{A}_2 is still open.

Notation and terminology. Write $H_n = \sum_{i=1}^n 1/i$. A *unit* circle (resp., segment) is one of unit length. For a given patrolling algorithm \mathcal{A} , using maximum speeds $v_1, \dots, v_k > 0$, let $\text{idle}(\mathcal{A}, v_1, \dots, v_k)$, or just $\text{idle}(\mathcal{A})$ if there is no danger of confusion, denote its idle time.

Given k agents with maximum speeds $v_1, \dots, v_k > 0$, and a patrolling algorithm \mathcal{A} , let $L(\mathcal{A}, v_1, \dots, v_k)$ denote the maximum length of a segment patrolled

by these agents using algorithm \mathcal{A} . Since the partition-based algorithm was conjectured to be optimal for a segment, it is natural to define the ratio of performance for any other algorithm \mathcal{A}' over the existing partition-based algorithm \mathcal{A}_1 as $\rho = \rho(\mathcal{A}', \mathcal{A}_1) = L(\mathcal{A}', v_1, \dots, v_k)/L(\mathcal{A}_1, v_1, \dots, v_k)$, where $L(\mathcal{A}_1, v_1, \dots, v_k) = (\sum_{i=1}^k v_i)/2$. This ratio can be used to evaluate strategies for patrolling—higher ratio implies better strategy. More generally one can compare two arbitrary strategies $\mathcal{A}', \mathcal{A}''$ via their lengths $L(\mathcal{A}', v_1, \dots, v_k)$ and $L(\mathcal{A}'', v_1, \dots, v_k)$. It is worth to keep in mind the equivalence between comparing different strategies via either their ratio or their idle time: if two algorithms compare with each other with ratio ρ in the length measure, the ratio of their idle times is $1/\rho$.

We use *distance-time diagrams* to plot the agent trajectories with respect to time. The x -coordinate represents distance along the fence and the y -coordinate represents time. For a constant-speed trajectory connecting (x_1, y_1) and (x_2, y_2) in the diagram, construct a shaded parallelogram with vertices, $(x_1, y_1), (x_1, y_1+I), (x_2, y_2), (x_2, y_2+I)$, where I denotes the idle time (in most of our cases, $I = 1$) and the shaded region represents the covered (guarded) region. A schedule for the agents ensures idle time I if and only if all area of the diagram in the time interval $[I, \infty)$ is covered.

General observations. 1. *Strategy scalability.* Suppose we have a patrolling strategy with k agents for a fence (open or closed) of length l with ratio ρ (relative to the partition strategy). Then, we can *scale* this strategy for every $l' \neq l$ using k agents as follows. Let $l'/l = c$, then $v'_i = cv_i, 1 \leq i \leq k$, where v'_i is the scaled speed of a_i . The waiting times used in the strategy at specific positions for agents need not to be scaled. One can check that the ratio ρ remains unchanged.

2. *Strategy extension.* Suppose we have a patrolling strategy with k agents for a fence (open or closed) of length l with ratio $\rho > 1$ (relative to the partition strategy). Then for any $k' > k$, there exists a patrolling strategy with k' agents for a fence of length $l' > l$ with ratio $\rho' > 1$: use $m = k' - k$ additional agents with $\sum_{i=k+1}^{k'} v_i = 2(l' - l)$ to patrol $l' - l$ using the partition strategy. Now if $\rho = \frac{a}{b} > 1$, then one can check that $\rho' = \frac{a+2(l'-l)}{b+2(l'-l)} > 1$. It follows from the results of Kawamura and Kobayashi [2] and the above observation that the partition based algorithm is not optimal for a segment for any $k \geq 6$, and k suitable speeds.

Our results.

- For every integer $x \geq 2$ there exist $k = 4x+1$ agents with $\sum_{i=1}^k v_i = 48x+3$ and a guarding schedule for a segment of length $25x/3$. Alternatively, for every integer $x \geq 2$ there exist $k = 4x+1$ agents with suitable speeds v_1, \dots, v_k , and a guarding schedule

for a unit segment that achieves idle time at most $\frac{48x+3}{50x} \frac{2}{\sum_{i=1}^k v_i}$. In particular, for every $\varepsilon > 0$, there exist k agents with suitable speeds v_1, \dots, v_k , and a guarding schedule for a unit segment that achieves idle time at most $(\frac{24}{25} + \varepsilon) \frac{2}{\sum_{i=1}^k v_i}$. See Theorem 3, Section 2.

- For every $k \geq 4$, there exist maximum speeds $v_1 \geq v_2 \geq \dots \geq v_k$ and a new patrolling algorithm \mathcal{A}_3 that yields an idle time better than that achieved by both \mathcal{A}_1 and \mathcal{A}_2 . In particular, for large k , the idle time of \mathcal{A}_3 with these speeds is about $2/3$ of that achieved by \mathcal{A}_1 and \mathcal{A}_2 . See Proposition 1, Section 3.
- Consider the unit circle, where all agents are required to move in the same direction. For every $t > 0$, there exists $k = k(t) = O(e^t)$ and a schedule for the system of agents with maximum speeds $v_i = 1/i, i = 1, \dots, k$, that ensures an idle time < 1 during the time interval $[0, t]$. See Proposition 2, Section 4.
- For every $k \geq 2$, there exist maximum speeds $v_1 \geq v_2 \geq \dots \geq v_k$ so that there exists an optimal schedule (patrolling algorithm) for the circle that does not use up to $k-1$ of the agents a_2, \dots, a_k . In contrast, for a segment, any optimal schedule must use all agents. See Proposition 3, Section 4.
- There exist settings in which if all k agents are used by a patrolling algorithm, then some agent(s) need overtake (pass) other agent(s). This follows from Proposition 3 and partially answers a question left open by Czyzowicz et al. [1, Section 3].
- When agents have some radius of visibility, there exists instances in which a zero “speed budget” suffices for guarding. E.g., k stationary agents with radii of visibility r_1, \dots, r_k , can guard a segment of length $2 \sum_{i=1}^k r_i$. This partially answers another question left open by Czyzowicz et al. [1, Section 3].

2 An improved idle time for open fence patrolling

In the paper by Kawamura and Kobayashi [2], the first example with 6 agents has $\rho = 42/41$ and the second example with 9 agents has $\rho = 100/99$. By repeating the strategy from the second example (with 9 agents) with a larger number of agents we improve the ratio to $25/24 - \varepsilon$ for any $\varepsilon > 0$. We need two technical lemmas.

Lemma 1 Consider a segment of length $L = \frac{25}{3}$ such that three agents a_1, a_2, a_3 are patrolling perpetually each with speed of 5 and generating an alternating sequence of uncovered triangles $T_2, T_1, T_2, T_1, \dots$, as shown in the distance-time diagram in Fig. 1. Denote the vertical distances between consecutive occurrences of T_1 and T_2 by δ_{12} and between consecutive occurrences of

T_2 and T_1 by δ_{21} . Denote the bases of T_1 and T_2 by b_1 and b_2 respectively, and the heights of T_1 and T_2 by h_1 and h_2 respectively . Then

- (i) $\frac{10}{3}$ is a period of the schedule.
- (ii) T_1 and T_2 are congruent; further, $b_1 = b_2 = \frac{1}{3}$, $\delta_{12} = \delta_{21} = \frac{4}{3}$, and $h_1 = h_2 = \frac{5}{6}$.

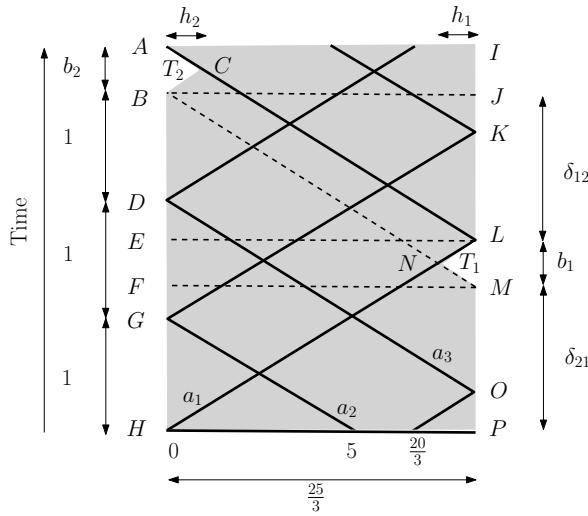


Figure 1: Three agents each with a speed of 5 patrolling a fence of length $25/3$; their start positions are 0, 5, and $20/3$, respectively. Figure is not to scale.

Proof. (i) Observe that a_1 , a_2 and a_3 reach the left endpoint of the segment at times $2(25/3)/5 = 10/3$, $5/5 = 1$, and $(25/3 + 5/3)/5 = 2$, respectively. During the time interval $[0, 10/3]$, each agent traverses the distance $2L$ and the positions and directions of the agents at time $t = 10/3$ are the same as those at time $t = 0$. Hence $10/3$ is a period for their schedule.

(ii) Since $AL \parallel BM$ and $AB \parallel LM$, we have $b_1 = b_2$. Since L is the midpoint of IP , we have $\delta_{12} + b_2 = \delta_{21} + b_1$, thus $\delta_{12} = \delta_{21}$. Since all the agents have same speed, 5, all the trajectory line segments in the distance-time diagram have the same slope, $1/5$. Hence $\angle BAC = \angle ABC = \angle MLN = \angle LMN$. Thus, T_1 is similar to T_2 . Since $b_1 = b_2$, T_1 is congruent to T_2 , hence $h_1 = h_2$.

Put $b = b_1$, $h = h_1$, and $\delta = \delta_{12}$. Recall from (i) that $|AH| = 10/3$. By construction, we have $|BD| = 1$, thus $|BH| = |BD| + |DG| + |GH| = 1 + 1 + 1 = 3$. We also have $|AH| = b + |BH|$, thus $b = 10/3 - 3 = 1/3$. Since L is the midpoint of IP , we have $\delta + b = 5/3$, thus $\delta = 5/3 - b = 4/3$.

Let $x(N)$ denote the x -coordinate of point N ; then $x(N) + h = 25/3$. To compute $x(N)$ we compute the intersection of the two segments HL and BM . We have $H = (0, 0)$, $L = (25/3, 5/3)$, $B = (0, 3)$, and $M = (25/3, 4/3)$. The equations of HL and BM are $HL : x =$

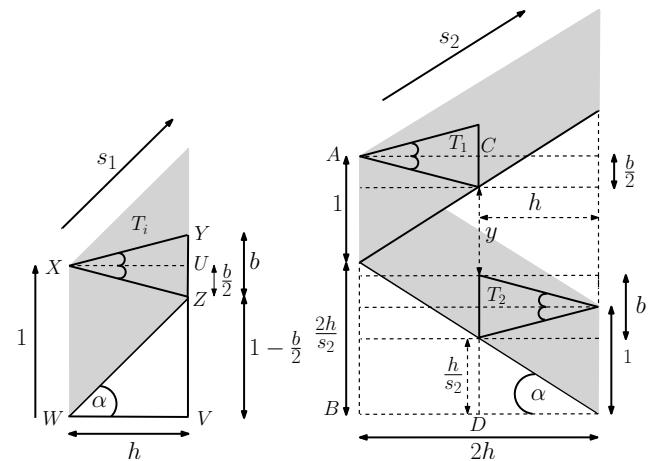


Figure 2: Left: agent covering an uncovered triangle T_i . Right: agent covering an alternate sequence of congruent triangles T_1, T_2 , with collinear bases.

$5y$ and $BM : x + 5y = 15$, and solving for x yields $x = 15/2$, and consequently $h = 25/3 - 15/2 = 5/6$. \square

Lemma 2 (i) Let s_1 be the speed of an agent needed to cover an uncovered isosceles triangle T_i ; refer to Fig. 2(left). Then $s_1 = \frac{h}{1-b/2}$, where $b < 1$ and h are the base and height of T_i , respectively.

(ii) Let s_2 be the speed of an agent needed to cover an alternate sequence of congruent isosceles triangles T_1, T_2 with bases on same vertical line; refer to Fig. 2(right). Then $s_2 = \frac{h}{3b/2+y-1}$ where y is the vertical distance between the triangles, $b < 1$ is the base and h is the height of the congruent triangles.

Proof. (i) In Fig. 2(left), $\tan \alpha = 1/s_1$, $|UZ| = b/2$, hence $|VZ| = 1 - b/2$. Also, $\frac{|VZ|}{|WV|} = \tan \alpha = \frac{1-b/2}{h} = \frac{1}{s_1}$, which yields $s_1 = \frac{h}{1-b/2}$.

(ii) In Fig. 2(right), $|AB| = 1 + \frac{2h}{s_2}$. Also, $|CD| = \frac{b}{2} + y + b + \frac{h}{s_2}$. Equating $1 + \frac{2h}{s_2} = \frac{3b}{2} + y + \frac{h}{s_2}$ and solving for s_2 , we get $s_2 = \frac{h}{3b/2+y-1}$. \square

Theorem 3 For every integer $x \geq 2$, there exist $k = 4x + 1$ agents with $\sum_{i=1}^k v_i = 48x + 3$ and a guarding schedule for a segment of length $25x/3$. Alternatively, for every integer $x \geq 2$ there exist $k = 4x + 1$ agents with suitable speeds v_1, \dots, v_k , and a guarding schedule for a unit segment that achieves idle time at most $\frac{48x+3}{50x} \frac{2}{\sum_{i=1}^k v_i}$. In particular, for every $\varepsilon > 0$, there exist k agents with suitable speeds v_1, \dots, v_k , and a guarding schedule for a unit segment that achieves idle time at most $(\frac{24}{25} + \varepsilon) \frac{2}{\sum_{i=1}^k v_i}$.

Proof. Refer to Fig. 3. We use a long fence divided into x blocks; each block is of length $25/3$. Each block has 3 agents each of speed 5 running in zig-zag fashion.

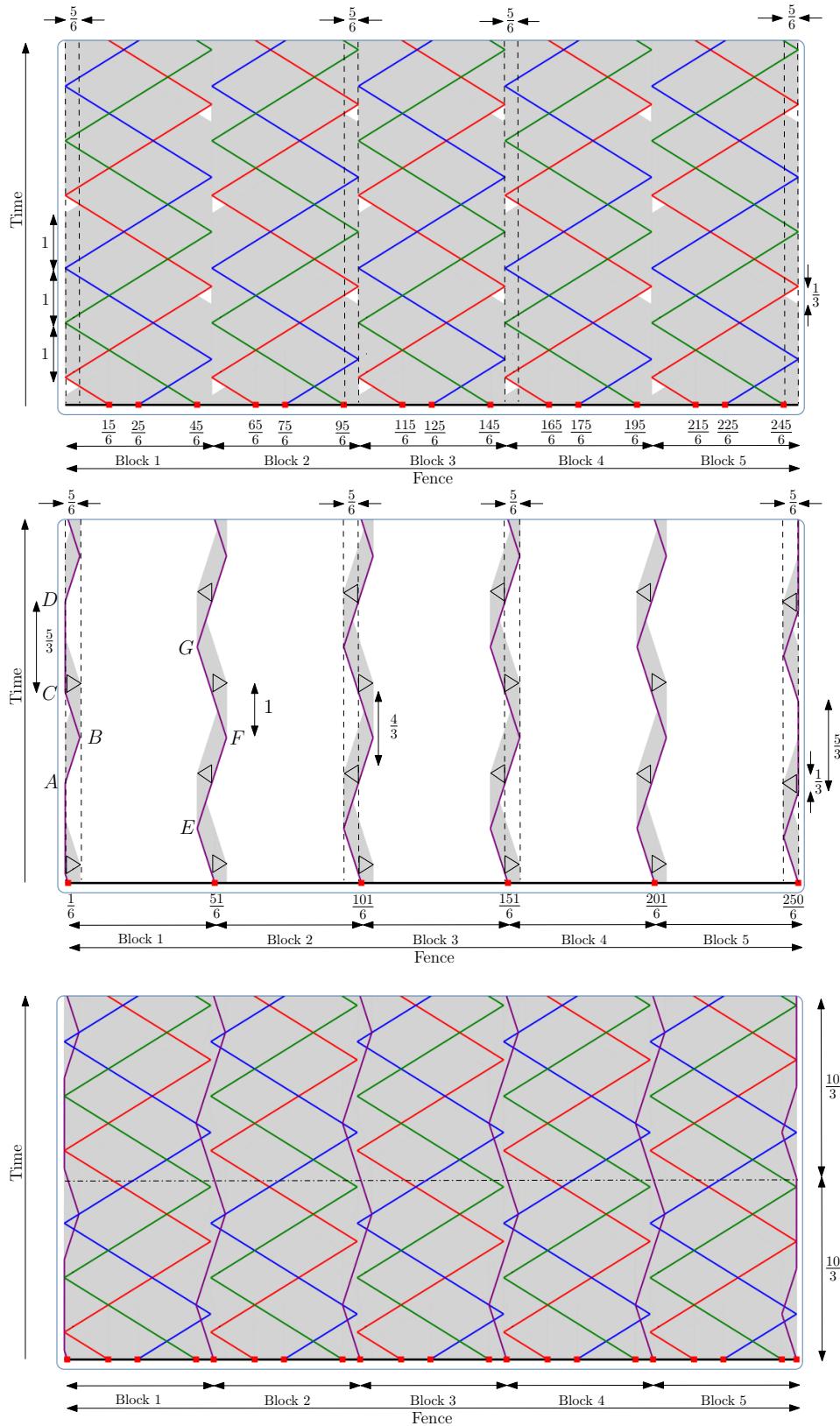


Figure 3: Top: iterative construction with 5 blocks; each block has three agents with speed 5. Middle: six agents with speed 1. Bottom: patrolling strategy for 5 blocks using 21 agents for two time periods (starting at $t = 1/3$ relative to Fig. 1); the block length is $25/3$ and the time period is $10/3$.

Consecutive blocks share one agent of speed 1 which covers the uncovered triangles from the trajectories of the zig-zag agents in the distance-time diagram. The first and the last block use two agents of speed 1 not shared by any other block. The setting of these speeds is explained below.

From Lemma 1(ii), we conclude that all the uncovered triangles generated by the agents of speed 5 are congruent and their base is $b = 1/3$ and their height is $h = 5/6$. By Lemma 2(i), we can set the speeds of the agents not shared by consecutive blocks to $s_1 = \frac{5/6}{1-1/6} = 1$. Also, in our strategy, Lemma 1(ii) yields $y = \delta = 4/3$. Hence, by Lemma 2(ii), we can set the speeds of the agents shared by consecutive blocks to $s_2 = \frac{5/6}{1/2+4/3-1} = 1$.

In our strategy, we have 3 types of agents: agents running with speed 5 as in Fig. 3(top), unit speed agents not shared by 2 consecutive blocks and unit speed agents shared by two consecutive blocks as in Fig. 3(middle). By Lemma 1(i), the agents of first type have period $10/3$. In Fig. 3(middle), there are two agents of second type and both have a similar trajectory. Thus, it is enough to verify for the leftmost unit speed agent. It takes $5/6$ time from A to B and again $5/6$ time from B to C . Next, it waits for $5/3$ time at C . Hence after $5/6 + 5/6 + 5/3 = 10/3$ time, its position and direction at D is same as that at A . Hence, its time period is $10/3$. For the agents of third type, refer to Fig. 3(middle): it takes $10/6$ time from E to F and $10/6$ time from F to G . Thus, arguing as above, its time period is $10/3$. Hence, overall the time period of the strategy is $10/3$.

For x blocks, we use $3x + (x + 1) = 4x + 1$ agents. The sum of all speeds is $5(3x) + 1(x + 1) = 16x + 1$ and the total fence length is $\frac{25x}{3}$. The resulting ratio is $\rho = \frac{25x}{3}/\frac{16x+1}{2} = \frac{50x}{48x+3}$. For example, when $x = 2$ we reobtain the bound of Kawamura and Kobayashi [2], when $x = 39$, $\rho = \frac{104}{100}$ and further on, $\rho \xrightarrow{x \rightarrow \infty} \frac{25}{24}$. \square

3 A new algorithm for closed fence patrolling

Czyzowicz et al. [1, Theorem 5] showed that for $k = 3$ there exist speed settings and an algorithm that achieves an idle time better than both \mathcal{A}_1 and \mathcal{A}_2 in this case: $35/36$ versus $12/11$ and 1. We extend this result for any $k \geq 4$.

Proposition 1 *For every $k \geq 4$, there exist maximum speeds $v_1 > v_2 \geq \dots \geq v_k$ so that a new patrolling algorithm \mathcal{A}_3 (we refer to as the “train algorithm”) yields an idle time better than that achieved by both \mathcal{A}_1 and \mathcal{A}_2 . In particular, for large k , the idle time of \mathcal{A}_3 with these speeds is about $2/3$ of that achieved by \mathcal{A}_1 and \mathcal{A}_2 .*

Proof. We will need $v_1 > v_2$ in this algorithm. Place the $k - 1$ agents a_2, \dots, a_k at equal distances, x on the unit circle, and let them move all clockwise at the same

speed v_k ; we say that a_2, \dots, a_k make a “train”. Let a_1 move back and forth (i.e., clockwise and counterclockwise) on the moving segment of length $1 - (k - 2)x$, i.e., between the start and the end of the train. Refer to Fig. 4. Consider the speed setting: $v_1 = a$,

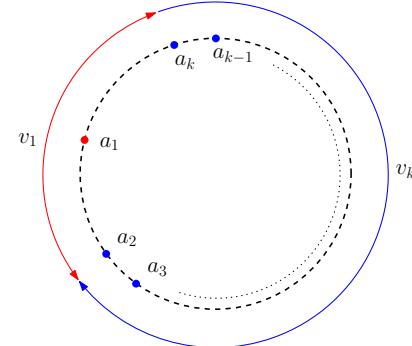


Figure 4: Train algorithm: the train a_2, \dots, a_k moving unidirectionally with speed v_k and the bidirectional agent a_1 with speed v_1 .

$v_2 = \dots = v_k = b$, where $a > b$, and $\max_{1 \leq i \leq k} iv_i = kb$ (i.e., $a \leq kb$). Put $y = 1 - (k - 2)x$. To determine the idle time, x/b , write: $[1 - (k - 2)x] \left(\frac{1}{a-b} + \frac{1}{a+b} \right) = \frac{x}{b}$, or equivalently, $\frac{2ay}{a^2-b^2} = \frac{1-y}{(k-2)b}$. Solving for x/b yields

$$\text{idle}(\mathcal{A}_3) = \frac{2a}{a^2 - b^2 + 2(k - 2)ab}.$$

For our setting, we also have

$$\text{idle}(\mathcal{A}_1) = \frac{2}{a + (k - 1)b}, \text{ and } \text{idle}(\mathcal{A}_2) = \frac{1}{kb}.$$

Write $t = a/b$. It can be checked that for $k \geq 4$, $\text{idle}(\mathcal{A}_3) \leq \text{idle}(\mathcal{A}_1)$ and $\text{idle}(\mathcal{A}_3) \leq \text{idle}(\mathcal{A}_2)$ when $a^2 - b^2 - 4ab \geq 0$, i.e., $t \geq 2 + \sqrt{5}$. In particular, for $a = 1$, and $b = 1/k$ (note that $a \leq kb$), we have

$$\text{idle}(\mathcal{A}_3) = \frac{2}{1 - 1/k^2 + 2(k - 2)/k} \xrightarrow{k \rightarrow \infty} \frac{2}{3},$$

$$\text{while } \text{idle}(\mathcal{A}_1) = \frac{2}{1+(k-1)/k} \xrightarrow{k \rightarrow \infty} 1 \text{ and } \text{idle}(\mathcal{A}_2) = \frac{1}{k(1/k)} = 1. \quad \square$$

4 Remarks

Finite time circle patrolling. While we cannot confirm the conjectured optimality of \mathcal{A}_2 —in particular, for the system of agents with maximum speeds $v_i = 1/i$, $i = 1, \dots, k$, acting on the unit circle, we would have $\text{idle}(\mathcal{A}_2) = 1$ —we can achieve an idle time below 1 in this setting for an arbitrarily long time, provided we choose k large enough. Obviously for this setting we have $\text{idle}(\mathcal{A}_2) \leq 1$, which is already achieved by the agent a_1 with the highest (here unit) speed, and the conjecture says that $\text{idle}(\mathcal{A}_2) < 1$ does not hold.

Proposition 2 Consider the unit circle, where all agents are required to move in the same direction. For every $t > 0$, there exists $k = k(t) = O(e^t)$ and a schedule for the system of agents with maximum speeds $v_i = 1/i$, $i = 1, \dots, k$, that ensures an idle time < 1 during the time interval $[0, t]$.

Proof. We construct a schedule with an idle time smaller than 1. Let $a_1(t) = t \bmod 1$ denote the position of agent a_1 at time t ; in particular $a_1(0) = 0$ with a_1 moving clockwise at maximum (unit) speed. We ensure that for each $t \geq 1$ there exists an agent that covers the interval $[t - \delta_1, t + \delta_2]$, for suitable $\delta_1, \delta_2 > 0$ before a_1 reaches this interval at time $t - \delta_1$. (We ignore any other contribution of this agent to the overall coverage.) We use many different agents to cover all time instances $t' \in [1, t]$. To this end we use the well-known fact that the harmonic series $\sum_1^\infty 1/i$ is divergent, more precisely that $H_k \geq \ln(k+1)$.

To start with, put $u_1 = 1$ as the first uncovered time instant t' , and $i = 2$ as the index of the next unused agent. Having defined u_{i-1} , initiate the movement of the next agent a_i at time $u_{i-1} - 1/2$ from the position $u_{i-1} - 1/(8i)$. Its speed is $1/i$ and during a time interval of $1/2$, the agent will traverse a distance equal to $1/(2i)$. Hence the agent's position at time u_{i-1} will be $u_{i-1} - 1/(8i) + 1/(2i) = u_{i-1} + 3/(8i)$. Now set $u_i = u_{i-1} + 3/(8i)$. In particular, $u_2 = 1 + 3/(8 \cdot 2)$ is the second uncovered time (to be covered by another agent), and $u_3 = 1 + 3/(8 \cdot 2) + 3/(8 \cdot 3)$ is the next such term. The solution of the recurrence is $u_k = \frac{5}{8} + \frac{3}{8}H_k$, and we need $u_k \geq t$. Since $H_k \geq \ln(k+1)$, it follows that $k = O(e^t)$ agents suffice to cover the time interval $[0, t]$ and ensure an idle time smaller than 1 in this way. \square

Useless agents for circle patrolling. Czyzowicz et al. [1] showed that for $k = 2$ there exist speed settings when an optimal schedule does not use one of the agents. Here we extend this result for all $k \geq 2$:

Proposition 3 (i) For every $k \geq 2$, there exist maximum speeds $v_1 \geq v_2 \geq \dots \geq v_k > 0$ and an optimal schedule (patrolling algorithm) for the circle with these speeds that does not use up to $k-1$ of the agents a_2, \dots, a_k . (ii) In contrast, for a segment, any optimal schedule must use all agents.

Proof. (i) Let $v_1 = 1$ and $v_2 = \dots = v_k = \varepsilon/k$, for a small positive $\varepsilon \leq 1/300$, and C be a unit length circle. Obviously by using agent a_1 alone (moving perpetually clockwise) we can achieve unit idle time. Assume for contradiction that there exists a schedule achieving an idle time less than 1. Let $a_1(t) = t \bmod 1$ denote the position of agent a_1 at time t . Assume without loss of generality that $a_1(0) = 0$ and consider the time interval $[0, 2]$. For $2 \leq i \leq k$, let J_i be the interval of points

visited by agent a_i during the time interval $[0, 2]$, and put $J = \cup_{i=2}^k J_i$. We have $|J_i| \leq 2\varepsilon/k$, thus $|J| \leq 2\varepsilon$. We make the following observations:

1. $a_1(1) \in [-2\varepsilon, 2\varepsilon]$. Indeed, if $a_1(1) \notin [-2\varepsilon, 2\varepsilon]$, then either some point in $[-2\varepsilon, 2\varepsilon]$ is not visited by any agent during the time interval $[0, 1]$, or some point in $C \setminus [-2\varepsilon, 2\varepsilon]$ is not visited by any agent during the time interval $[0, 1]$.
2. a_1 has done almost a complete (say, clockwise) rotation along C during the time interval $[0, 1]$, i.e., it starts at $0 \in [-2\varepsilon, 2\varepsilon]$ and ends in $[-2\varepsilon, 2\varepsilon]$, otherwise some point in $C \setminus [-2\varepsilon, 2\varepsilon]$ is not visited during the time interval $[0, 1]$.
3. $a_1(2) \in [-4\varepsilon, 4\varepsilon]$, by a similar argument.
4. a_1 has done almost a complete rotation along C during the time interval $[1, 2]$, i.e., it starts in $[-2\varepsilon, 2\varepsilon]$ and ends in $[-4\varepsilon, 4\varepsilon]$. Moreover this rotation must be in the same clockwise sense as the previous one, since otherwise there would exist points not visited for at least one unit of time.

Pick three points $x_1, x_2, x_3 \in C \setminus J$ close to $1/4, 2/4$, and $3/4$, respectively, i.e., $|x_i - i/4| \leq 1/100$, for $i = 1, 2, 3$. By Observations 2 and 4, these three points must be visited by a_1 in the first two rotations during the time interval $[0, 2]$ in the order $x_1, x_2, x_3, x_1, x_2, x_3$. Since a_1 has unit speed, successive visits to x_1 are at least one time unit apart, contradicting the assumption that the idle time of the schedule is less than 1.

(ii) Given $v_1 \geq v_2 \geq \dots \geq v_k > 0$, assume for contradiction that there is an optimal guarding schedule with unit idle time for a segment s of maximum length that does not use agent a_j (with maximum speed v_j), for some $1 \leq j \leq k$. Extend s at one end by a subsegment of length $v_j/2$ and assign a_j to this subsegment to move back and forth from one end to the other, perpetually. We now have a guarding schedule with unit idle time for a segment longer than s , which is a contradiction. \square

Acknowledgements. We sincerely thank Akitoshi Kawamura for generously sharing some technical details concerning their algorithms. We also express our satisfaction with the JavaScript library *JSXGraph*.

References

- [1] J. Czyzowicz, L. Gasieniec, A. Kosowski, and E. Kranakis, Boundary patrolling by mobile agents with distinct maximal speeds, *Proc. 19th European Sympos. on Algor. (ESA 2011)*, LNCS 6942, 2011, pp. 701–712.
- [2] A. Kawamura and Y. Kobayashi, Fence patrolling by mobile agents with distinct speeds, *Proc. 23rd International Sympos. on Algor. and Computation (ISAAC 2012)*, LNCS 7676, 2012, pp. 598–608.

Face-Guarding Polyhedra

Giovanni Viglietta*

Abstract

We study the Art Gallery Problem for face guards in polyhedral environments. The problem can be informally stated as: *how many (not necessarily convex) windows should we place on the external walls of a dark building, in order to completely illuminate it?*

We consider both closed and open face guards (i.e., faces with or without their boundary), and we give some upper and lower bounds on the minimum number of faces required to guard a given polyhedron, in terms of the total number of its faces, f . In some notable cases, our bounds are tight: $\lfloor f/6 \rfloor$ open face guards for *orthogonal* polyhedra, and $\lfloor f/4 \rfloor$ open face guards for *4-oriented* polyhedra (i.e., polyhedra whose faces have only four different orientations).

Then we show that it is **NP-hard** to approximate the minimum number of (closed or open) face guards within a factor of $\Omega(\log f)$, even for polyhedra that are orthogonal and simply connected.

Along the way we discuss some applications, arguing that face guards are *not* a reasonable model for guards *patrolling* on the surface of a polyhedron.

1 Introduction

Previous work. Art Gallery Problems have been studied in computational geometry for decades: given an *enclosure*, place a (preferably small) set of *guards* such that every location in the enclosure is seen by some guard. Most of the early research on the Art Gallery Problem focused on guarding 2-dimensional polygons with either point guards or segment guards [9, 10, 12].

Gradually, some of the attention started shifting to 3-dimensional settings, as well. Several authors have considered edge guards in 3-dimensional polyhedra, either in relation to the classical Art Gallery Problem or to its variations [2, 3, 4, 13, 14].

Recently, Souvaine et al. [11] introduced the model with *face guards* in 3-dimensional polyhedra. Ideally, each guard is free to roam over an entire face of a polyhedron, including the face’s boundary. They gave lower and upper bounds on g , the number of face guards that are required to guard a given polyhedron, in terms of f , the total number of its faces. For general polyhedra, they showed that $\lfloor f/5 \rfloor \leq g \leq \lfloor f/2 \rfloor$ and, for

the special case of orthogonal polyhedra (i.e., polyhedra whose faces meet at right angles), they showed that $\lfloor f/7 \rfloor \leq g \leq \lfloor f/6 \rfloor$. They also suggested several open problems, such as studying *open* face guards (i.e., face guards whose boundary is omitted), and the computational complexity of minimizing the number of face guards.

Subsequently, face guards have been studied to some extent also in the case of polyhedral terrains. In [8], a lower bound is obtained, and in [7] it is proved that minimizing face guards in terrains is **NP-hard**.

Our contribution. In this paper we solve some of the problems left open in [11], and we also expand our research in some new directions.

In Section 2 we discuss the face guard model, arguing that a face guard fails to meaningfully represent a guard “patrolling” on a face of a polyhedron. Essentially, there are cases in which the path that such a patrolling guard ought to follow is so complex (in terms of the number of turns, if it is a polygonal chain) that a much simpler path, striding from the face, would guard not only the region visible from that face, but the entire polyhedron. However, face guards are still a good model for illumination-related problems, such as placing (possibly non-convex) windows in a dark building.

In Section 3 we obtain some new bounds on g , for both closed and open face guards. Namely, we generalize the upper bounds given in [11] by showing that, for c -oriented polyhedra (i.e., whose faces have c distinct orientations), $g \leq \lfloor f/2 - f/c \rfloor$. We also provide some new lower bound constructions, which meet our upper bounds in two notable cases: orthogonal polyhedra with open face guards ($g = \lfloor f/6 \rfloor$), and 4-oriented polyhedra with open face guards ($g = \lfloor f/4 \rfloor$).

In Section 4 we provide an approximation-preserving reduction from **SET COVER** to the problem of minimizing the number of (closed or open) face guards in simply connected orthogonal polyhedra. It follows that the minimum number of face guards is **NP-hard** to approximate within a factor of $\Omega(\log f)$. We also discuss the membership in **NP** of the minimization problem.

2 Model and motivations

Definitions. Given a polyhedron in \mathbb{R}^3 , we say that a point x is *visible* to a point y if no point in the straight

*School of Computer Science, Carleton University, Ottawa ON, Canada, viglietta@gmail.com.

line segment xy lies in the exterior of the polyhedron. For any point x , we denote by $\mathcal{V}(x)$ the *visible region* of x , i.e., the set of points that are visible to x . In general, for any set $S \subset \mathbb{R}^3$, we let $\mathcal{V}(S) = \bigcup_{x \in S} \mathcal{V}(x)$. A set is said to *guard* a polyhedron if its visible region coincides with the entire polyhedron (including its boundary). The Art Gallery Problem for face guards in polyhedra consists in finding a (preferably small) set of faces whose union guards a given polyhedron. If such faces include their relative boundary, they are called *closed* face guards; if their boundary is omitted, they are called *open* face guards.

A polyhedron is *c-oriented* if there exist c unit vectors such that each face is orthogonal to one of the vectors. If these unit vectors form an orthonormal basis of \mathbb{R}^3 , the polyhedron is said to be *orthogonal*. Hence, a cube is orthogonal, a tetrahedron and a regular octahedron are both 4-oriented, etc.

Motivations. There is a straightforward analogy between *guarding* problems and *illumination* problems: placing guards in a polyhedron corresponds to placing light sources in a dark building, in order to illuminate it completely. For instance, a point guard would model a *light bulb* and a segment guard could be a *fluorescent tube*. Because face guards are 2-dimensional and lie on the boundary of the polyhedron, we may think of them as *windows*. A window may have any shape, but should be flat, and hence it should lie on a single face. It follows that, if our purpose is to illuminate as big a region as possible, we may assume without loss of generality that a window always coincides with some face.

Face guards were introduced in [11] to represent guards *roaming* over a face. This is in accordance with the traditional usage of segment guards as a model for guards that *patrol* on a line [9]. While this is perfectly sound in the case of segment guards, face guards pose additional problems, as explained next.

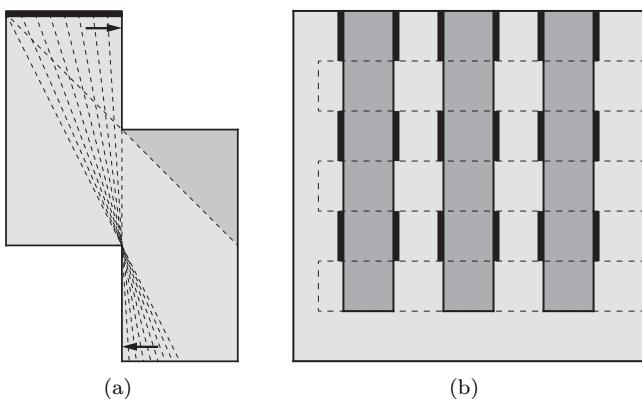


Figure 1: Constructing the polyhedron in Figure 2

We begin by observing that, even in 2-dimensional

polygons, there may be edge guards that cannot be locally “replaced” by finitely many point guards. Figure 1(a) shows an example: if a subset G of the top edge ℓ is such that $\mathcal{V}(G) = \mathcal{V}(\ell)$, then the right endpoint of ℓ must be a limit point of G .

We can exploit this fact to construct the class of polyhedra sketched in Figure 2. We cut long parallel *dents* on opposite faces of a cuboid, in such a way that the resulting polyhedron looks like an extruded “iteration” of the polygon in Figure 1(a). Then we stab this construction with a row of *girders* running orthogonally with respect to the dents.

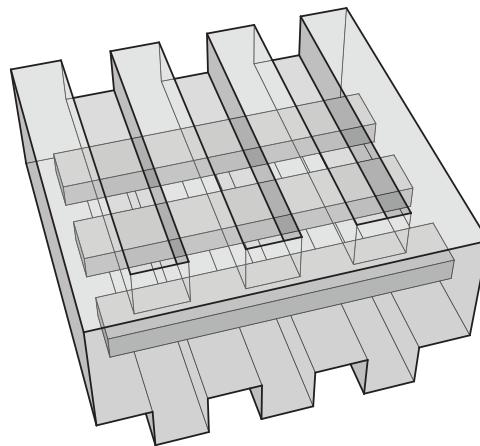


Figure 2: A guard patrolling on the top face must follow a path of quadratic complexity

Suppose that a guard has to patrol the top face of this construction, eventually seeing every point that is visible from that face. The situation is represented in Figure 1(b), where the light-shaded region is the top face, and the dashed lines mark the underlying girders. By the above observation and by the presence of the girders, each thick vertical segment must be approached by the patrolling guard from the interior of the face.

Suppose that the polyhedron has n dents and n girders. Then, the number of its vertices, edges, or faces is $\Theta(n)$. Now, if the guard moves along a polygonal chain lying on the top face, such a chain must have at least a vertex on each thick segment, which amounts to $\Omega(n^2)$ vertices. Similarly, if the face guard has to be substituted with segment guards lying on it, quadratically many guards are needed.

On the other hand, it is easy to show that a path of linear complexity is sufficient to guard any polyhedron: triangulate every face (thus adding linearly many new “edges”) and traverse the resulting 1-skeleton in depth-first order starting from any vertex, thus covering all edges. Because the set of edges is a guarding set for any polyhedron [13], the claim follows.

This defeats the purpose of having faces model guards patrolling on segments, as it makes little sense for a

face of “unit weight” to represent quadratically many guards. Analogously, a roaming guard represented by a face may have to follow a path that is overly complex compared to the guarding problem’s optimal solution.

Even if we are allowed to replace a face guard with guards patrolling any segment in the polyhedron (i.e., not necessarily constrained to live on that face), a linear number of them may be required. Indeed, consider a cuboid with very small height, and arrange n thin and long *chimneys* on its top, in such a way that no straight line intersects more than two chimneys. The complexity of the polyhedron is $\Theta(n)$, and a face guard lying on the bottom face must be replaced by $\Omega(n)$ segment guards. On the other hand, we know that a linear amount of segment guards is enough not only to “dominate” a single face, but to entirely guard any polyhedron.

Summarizing, a face guard appropriately models an entity that is naturally constrained to live on a single face, like a flat window, and unlike a team of patrolling guards. In the case of a single roaming guard, the model is insensitive to the complexity of the guard’s path.

3 Bounds on face guard numbers

Upper bounds. By generalizing the approach used in [11, Lemmas 2.1, 3.1], we provide an upper bound on face guard numbers, which becomes tight for open face guards in orthogonal polyhedra and open face guards in 4-oriented polyhedra. We emphasize that our upper bound holds for both closed and open face guards, and for polyhedra of any genus.

Theorem 1 Any c -oriented polyhedron with f faces is guardable by

$$\left\lfloor \frac{f}{2} - \frac{f}{c} \right\rfloor$$

closed or open face guards.

Proof. Let \mathcal{P} be a polyhedron whose faces are orthogonal to $c \geq 3$ distinct vectors. Let f_i be the number of faces orthogonal to the i -th vector v_i . We may assume that $i < j$ implies $f_i \geq f_j$. Then,

$$f_1 + f_2 \geq \left\lfloor \frac{2f}{c} \right\rfloor.$$

Let us stipulate that the direction of the cross product $v_1 \times v_2$ is *vertical*. Thus, there are at most

$$f - \left\lfloor \frac{2f}{c} \right\rfloor$$

non-vertical faces. Some of these are facing up, the others are facing down. Without loss of generality, at most half of them are facing down, and we assign a face guard to each of them. Therefore, at most

$$\left\lfloor \frac{f}{2} - \frac{f}{c} \right\rfloor$$

face guards have been assigned.

Let x be any point in \mathcal{P} . If x belongs to a face with a guard, x is guarded. Otherwise, consider an infinite circular cone \mathcal{C} with apex x and axis directed upward. Let \mathcal{G} be the intersection of $\mathcal{V}(x)$, \mathcal{C} , and the boundary of \mathcal{P} . If the aperture of \mathcal{C} is small enough, the relative interior of \mathcal{G} belongs entirely to faces containing guards and to at most two vertical faces containing x . Because these vertical faces obstruct at most one dihedral angle from x ’s view, the portion of \mathcal{G} not belonging to them has non-empty interior. If we remove from this portion the (finitely many) edges of \mathcal{P} , we still have a non-empty region. By construction, this region belongs to the interiors of faces containing a guard; hence x is guarded. \square

Our guarding strategy becomes less efficient as c grows. In general, if no two faces are parallel (i.e., $c = f$), we get an upper bound of $\lfloor f/2 \rfloor - 1$ face guards, which improves on the one in [11] by just one unit.

Lower bounds. In [11], Souvaine et al. construct a class of orthogonal polyhedra with f faces that need $\lfloor f/7 \rfloor$ closed face guards. In Figure 3 we give an alternative construction, with the additional property of having a 3-regular 1-skeleton. Indeed, each small L-shaped polyhedron that is attached to the big cuboid adds seven faces to the construction, of which at least one must be selected.

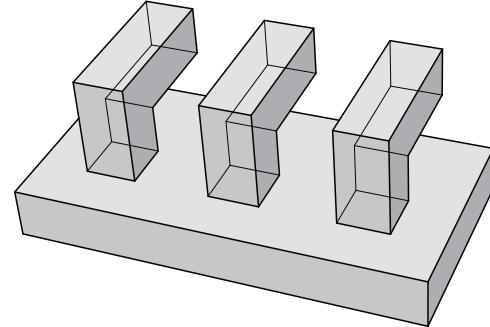


Figure 3: Orthogonal polyhedron that needs $\lfloor f/7 \rfloor$ closed face guards

For *open* face guards, we modify our previous construction by moving all the L-shaped pieces to the boundary of the top face, as in Figure 4. Thus, each piece adds just six faces to the construction (one face is shared by all of them), of which at least one must be selected. Moreover, no matter how these faces are selected, some portion of the big cuboid below remains unguarded, and needs one more face guard. This amounts to $\lfloor f/6 \rfloor$ open face guards in total.

Plugging $c = 3$ in Theorem 1 reveals that our lower bound is also tight.

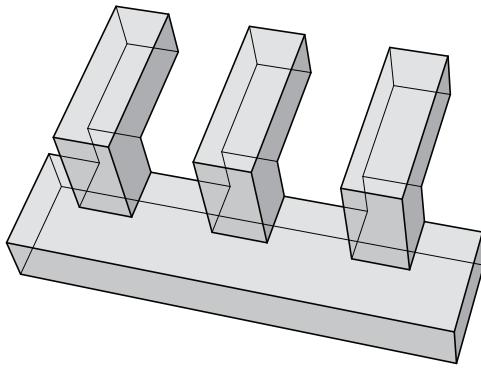


Figure 4: Orthogonal polyhedron that needs $\lfloor f/6 \rfloor$ open face guards

Theorem 2 *To guard an orthogonal polyhedron having f faces, $\lfloor f/6 \rfloor$ open face guards are always sufficient and occasionally necessary.* \square

Moving on to *closed* face guards in 4-oriented polyhedra, we propose the construction in Figure 5. Each closed face sees the tip of at most one of the k tetrahedral *spikes*, hence k guards are needed. Because there are $5k + 2$ faces in total, a lower bound of $\lfloor f/5 \rfloor$ closed face guards follows.

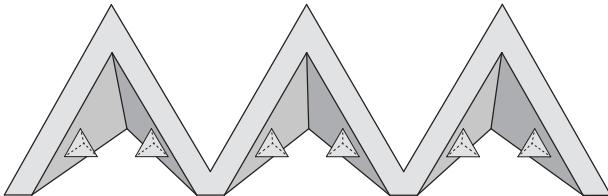


Figure 5: 4-oriented polyhedron that needs $\lfloor f/5 \rfloor$ closed face guards

For *open* face guards in 4-oriented polyhedra, we modify the previous example by carefully placing additional spikes on the other side of the construction, as Figure 6 illustrates. Once again, since each open face sees the tip of at most one of the k spikes and there are $4k + 2$ faces in total, a lower bound of $\lfloor f/4 \rfloor$ open face guards follows.

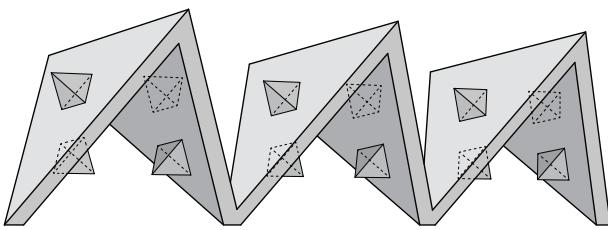


Figure 6: 4-oriented polyhedron that needs $\lfloor f/4 \rfloor$ open face guards

This bound is also tight, as easily seen by plugging $c = 4$ in Theorem 1.

Theorem 3 *To guard a 4-oriented polyhedron having f faces, $\lfloor f/4 \rfloor$ open face guards are always sufficient and occasionally necessary.* \square

4 Minimizing face guards

Hardness of approximation. In [11], Souvaine et al. ask for the complexity of minimizing face guards in a given polyhedron. We show that this problem is at least as hard as SET COVER, and we infer that approximating the minimum number of face guards within a factor of $\Omega(\log f)$ is NP-hard.

Theorem 4 *SET COVER is L-reducible to the problem of minimizing (closed or open) face guards in a simply connected orthogonal polyhedron.*

Proof. Let an instance of SET COVER be given, i.e., a universe $U = \{1, \dots, n\}$, and a collection $S \subseteq \mathcal{P}(U)$ of $m \geq 1$ subsets of U . We will construct a simply connected orthogonal polyhedron with $f = O(mn)$ faces that can be guarded by k (closed or open) faces if and only if U is the union of $k - 1$ elements of S .

Figure 7 shows our construction for $U = \{1, 2, 3, 4\}$ and $S = \{\{2, 4\}, \{1, 3\}, \{2\}\}$. Figure 8 illustrates the side view of a generic case in which $m = 4$.

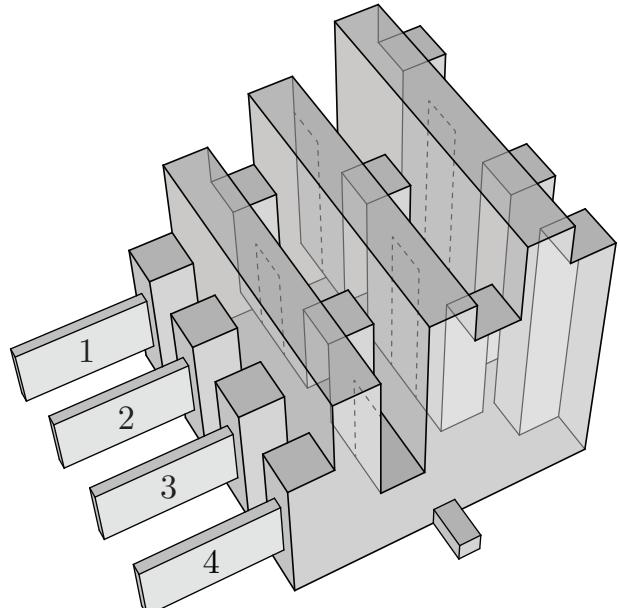


Figure 7: SET COVER reduction, 3D view

Each of the thin cuboids on the far left is called a *fissure*, and represents an element of U . In front of the fissures there is a row of m *mountains* of increasing height, separated by *valleys* of increasing depth. The

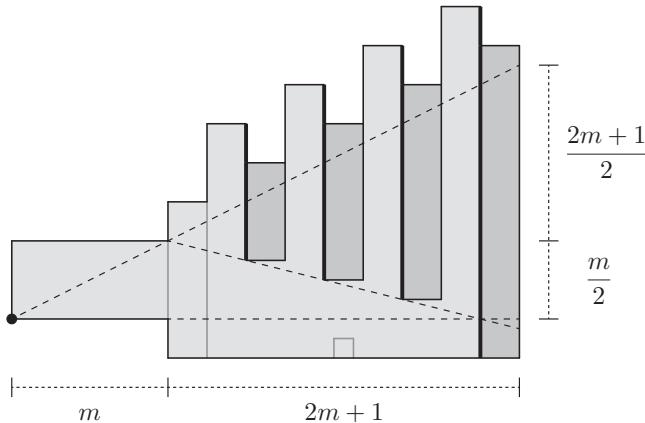


Figure 8: SET COVER reduction, side view

m vertical walls that are facing the fissures (drawn as thick lines in Figure 8) are called *set faces*, and each of them represents an element of S .

For each $S_i \in S$, we dig a narrow rectangular *dent* in the i -th set face in front of the j -th fissure, if and only if $j \notin S_i$. Each dent reaches the bottom of its set face, and almost reaches the top, so that it does not separate the set face into two distinct faces. Moreover, every dent (except those in the rightmost set face) is so deep that it connects two neighboring valleys. In Figure 8, dents are depicted as darker regions; in Figure 7, the dashed lines mark the areas where dents are *not* placed.

We want to fix the width of the fissures in such a way that only a restricted number of faces can see their bottom. Specifically, consider n *distinguished points*, located in the middle of the lower-left edges of the fissures (indicated by the thick dot in Figure 8). The j -th distinguished point definitely sees some portions of the i -th set face, provided that $j \in S_i$. If this is the case, and $i < m$, it also sees portions of two other faces (one horizontal, one vertical) surrounding the same valley. Moreover, if $j \notin S_m$, the j -th distinguished point also sees the bottom of a dent in the rightmost set face. We want no face to be able to see any distinguished point, except the faces listed above (plus of course the faces belonging to fissures or surrounding their openings). To this end, assuming that the dents have unit width, we set the width of the fissures to be slightly less than $1/4$. Indeed, referring to Figure 8, the width of the visible region of a distinguished point, as it reaches the far right of the construction, is strictly less than

$$\frac{(m) + (2m+1)}{m} \cdot \frac{1}{4} = \left(3 + \frac{1}{m}\right) \cdot \frac{1}{4} \leq 4 \cdot \frac{1}{4} = 1,$$

because $m \geq 1$.

Finally, a small *niche* is added in the lower part of the construction. Its purpose is to enforce the selection of a “dedicated” face guard, as no face can see both a distinguished point and the bottom of the niche.

Let a guarding set for our polyhedron be given, consisting of k face guards. We will show how to compute in polynomial time a solution of size at most $k-1$ for the given SET COVER instance, provided that it is solvable at all.

We first discard every face guard that is not guarding any distinguished point. Because at least one face must guard the niche, we are left with at most $k-1$ guards. Then, if any of the remaining face guards borders the i -th valley, with $i < m$, we replace it with the i -th set face. Indeed, it is easy to observe that such set face can see the same distinguished points, plus possibly some more. By construction, all the remaining guards can see exactly one distinguished point (they are either faces belonging to some fissure, or surrounding its opening, or bottom faces of the rightmost dents). We replace each of these face guards with any set face that guards the same distinguished point (which exists, otherwise the SET COVER instance would be unsolvable). As a result, we have at most $k-1$ set faces guarding all the distinguished points. These immediately determine a solution of equal size to the given SET COVER instance.

Conversely, if the SET COVER instance has a solution of size k , it is easy to see that our polyhedron has a guarding set of $k+1$ guards: all the set faces corresponding to the SET COVER’s solution, plus the bottom face. \square

Corollary 5 *Given a simply connected orthogonal polyhedron with f faces, it is NP-hard to approximate the minimum number of (closed or open) face guards within a factor of $\Omega(\log f)$.*

Proof. The polyhedra constructed in the L-reduction of Theorem 4 have $f = O(mn)$ faces. It was proved in [1] that SET COVER is NP-hard to approximate within a ratio of $\Omega(\log n)$ and, by inspecting the reduction employed, it is apparent that all the hard SET COVER instances generated are such that $m = O(n^c)$, for some constant $c \geq 1$. As a consequence, we may assume that $\Omega(\log n) = \Omega(\log n^{c+1}) = \Omega(\log(mn)) = \Omega(\log f)$, and our claim follows. \square

Computing visible regions. The next natural question is whether the minimum number of face guards can be computed in NP, and possibly approximated within a factor of $\Theta(\log f)$. Usually, when finitely many possible guard locations are allowed (such as with vertex guards and edge guards), this is established by showing that the visible region of any guard can be computed efficiently, as well as the intersection of two visible regions, etc. As a result, the environment is partitioned into polynomially many regions such that, for every region R and every guard g , either $R \subseteq \mathcal{V}(g)$ or $R \cap \mathcal{V}(g) = \emptyset$. This immediately leads to a reduction to SET COVER,

which implies an approximation algorithm with logarithmic ratio, via a well-known greedy heuristic [6].

With face guards (and also with edge guards in polyhedra) the situation is complicated by the fact that the visible region of a guard may not be a polyhedron, but in general its boundary is a piecewise quadric surface.

For example, consider the orthogonal polyhedron in Figure 9. It is easy to see that the visible region of the bottom face (and also the visible region of edge a) is the whole polyhedron, except for a small region bordered by the thick dashed lines.

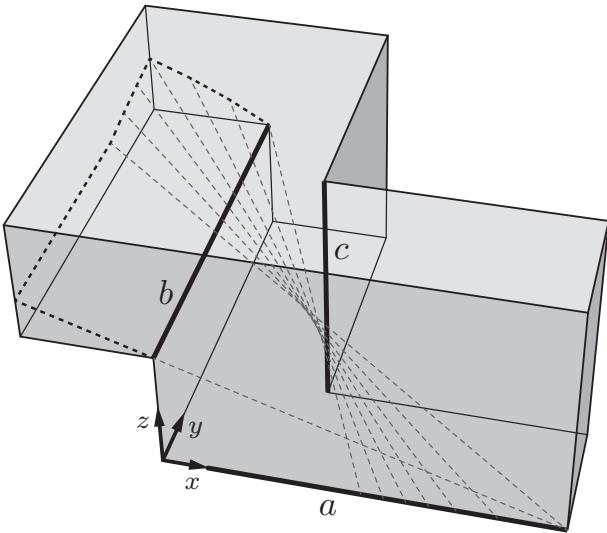


Figure 9: The visible region of the bottom face is bounded by a hyperboloid of one sheet.

The surface separating the visible and invisible regions consists of a right trapezoid plus a bundle of mutually skew segments whose extensions pass through the edges a , b , and c . These edges lie on three lines having equations

$$\begin{aligned} y^2 + z^2 &= 0, \\ x^2 + (z - 1)^2 &= 0, \\ (x - 1)^2 + (y - 1)^2 &= 0, \end{aligned}$$

respectively. A straightforward computation reveals that the bundle of lines passing through these three lines has equation

$$xy - xz + yz - y = 0,$$

which defines a hyperboloid of one sheet.

In general, the boundary of the visible area of a face (or an edge) is determined by lines passing through pairs or triplets of edges of the polyhedron. If three edges are all parallel to a common plane, the surface they determine is a hyperbolic paraboloid (degenerating into a plane if two of the edges are parallel to each other), otherwise they determine a hyperboloid of one sheet, as in the above example.

There exists an extensive literature of purely algebraic methods to compute intersections of quadric surfaces (see for instance [5]), but the parameterizations involved may yield coefficients containing radicals. At this stage in our understanding, it is not clear whether any of these methods can be effectively applied to reduce the minimization problem of face-guarding polyhedra (or even edge-guarding polyhedra) to SET COVER.

References

- [1] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k -restrictions. *ACM Transactions on Algorithms*, vol. 2, pp. 153–177, 2006.
- [2] N. Benbernou, E. D. Demaine, M. L. Demaine, A. Kurdia, J. O'Rourke, G. T. Toussaint, J. Urrutia, and G. Viglietta. Edge-guarding orthogonal polyhedra. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, pp. 461–466, 2011.
- [3] J. Cano, C. D. Tóth, and J. Urrutia. Edge guards for polyhedra in 3-space. In *Proceedings of the 24th Canadian Conference on Computational Geometry*, pp. 155–160, 2012.
- [4] T. Christ and M. Hoffmann. Wireless localization within orthogonal polyhedra. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, pp. 467–472, 2011.
- [5] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean. A new algorithm for the robust intersection of two general quadrics. In *Proceedings of the 19th Annual ACM Symposium on Computational Geometry*, pp. 246–255, 2003.
- [6] S. K. Ghosh. Approximation algorithms for art gallery problems. In *Proceedings of the Canadian Information Processing Society Congress*, pp. 429–434, 1987.
- [7] C. Iwamoto, Y. Kitagaki, and K. Morita. Finding the minimum number of face guards is NP-hard. *IEICE Transactions on Information and Systems*, vol. E95-D, pp. 2716–2719, 2012.
- [8] C. Iwamoto, J. Kishi, and K. Morita. Lower bound of face guards of polyhedral terrains. *Journal of Information Processing*, vol. 20, pp. 435–437, 2012.
- [9] J. O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press, New York, 1987.
- [10] T. Shermer. Recent results in art galleries. In *Proceedings of the IEEE*, vol. 80, pp. 1384–1399, 1992.
- [11] D. L. Souvaine, R. Veroy, and A. Winslow. Face guards for art galleries. In *Proceedings of the XIV Spanish Meeting on Computational Geometry*, pp. 39–42, 2011.
- [12] J. Urrutia. Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pp. 973–1027, North-Holland, 2000.
- [13] G. Viglietta. *Guarding and searching polyhedra*. Ph.D. Thesis, University of Pisa, 2012.
- [14] G. Viglietta. Searching polyhedra by rotating half-planes. *International Journal of Computational Geometry and Applications*, vol. 22, pp. 243–275, 2012.

On k -Guarding Polygons

Daniel Bustos*

William Evans*

David Kirkpatrick*

Abstract

We describe a polynomial time $O(k \log \log \text{OPT}_k(P))$ -approximation algorithm for the k -guarding problem of finding a minimum number, $\text{OPT}_k(P)$, of vertex guards of an n -vertex simple polygon P so that for every point $p \in P$, the number of guards that see p is at least the minimum of k and the number of vertices that see p . Our approach finds $O\left(\frac{k}{\varepsilon} \log \log \frac{1}{\varepsilon}\right)$ size (k, ε) -nets for instances of the k -hitting set problem arising from the k -guarding problem. These nets contain k distinct elements (or the entire set if it has fewer than k elements) from any set that has at least an ε fraction of the total weight of all elements. To find a nearly optimal k -guarding, we slightly modify the technique of Brönnimann and Goodrich [4] so that the weights of all elements remain small, which is necessary for our (k, ε) -net finder. Our approach, generalizes, simplifies, and corrects a subtle flaw in the technique introduced by King and Kirkpatrick [11] to find small ε -nets for set systems arising from 1-guarding instances.

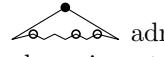
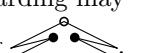
1 Introduction

In the classic art gallery problem one is given a simple polygon P in the plane and asked to find a smallest subset G of P , called guards, so that every point $p \in P$ is seen by at least one guard $g \in G$ (i.e., $\overline{gp} \subseteq P$). In this paper, we allow only *vertex guards*, meaning G is a subset of V , the vertices of polygon P . The original results on this problem addressed the extremal question of how many guards are needed to guard a simple polygon with n vertices. Chvátal [6], answering a question of Klee, showed that $\lfloor n/3 \rfloor$ guards are occasionally necessary and always sufficient to guard such polygons. O'Rourke's book [16] and Urrutia's chapter [18] describe the history and subsequent flurry of results in this area.

In some applications, we would like every point in P to be seen by more than one guard. For example, to use triangulation to locate an intruder, his position must be seen by at least two guards (whose locations are different). Surprisingly, the generalization of Klee's question to this form of 2-guarding[†] seems to have taken

over 30 years to appear, though, earlier, Belleville et al. [2] addressed a different form of k -guarding where each guard must be an interior point of a distinct edge of P . Salleh [17] showed that $\lfloor 2n/3 \rfloor$ guards are occasionally necessary and always sufficient to 2-guard a simple n -gon; and that for 3-guarding convexly quadrilateralizable n -gons, the bound is $\lfloor 3n/4 \rfloor$ guards. A simple proof [3, 15] follows from Fisk's triangulation colouring proof [8] of Chvátal's result. If we insist that guards must be at different vertices then there are simple polygons that cannot be k -guarded in this way for $k \geq 4$ because some points in P are seen by fewer than k vertices. While our original motivation concerned 2-guarding, which is always possible, our results apply to k -guarding in general, if we only require that the guarding do as well as it can for un- k -guardable points. Thus we say that a subset G of the vertices of P is a *k -guarding* of P if for every point $p \in P$ the number of guards that see p is at least the minimum of k and the number of vertices that see p .

Another option is to allow multiple guards at the same vertex. A multiset G of vertices of P is a *multi- k -guarding* of P if every point $p \in P$ is seen by at least k guards in G . Since k copies of any 1-guarding is a multi- k -guarding, $k\lfloor n/3 \rfloor$ guards are always sufficient to multi- k -guard any n -gon, and Chvátal's "comb"  shows they are occasionally necessary. While the smallest multi-2-guarding is at most twice the smallest 1-guarding, a smallest 2-guarding may be much larger.

An s -spiked fan  admits a 1-guarding (black dot) of size one and requires at least $\lceil s/2 \rceil + 1$ guards (all dots) to 2-guard. On the other hand, a 2-guarding may be smaller than twice the smallest 1-guarding . We address the problem of finding the smallest number of vertex guards, $\text{OPT}_k(P)$, needed to k -guard a given simple polygon P . In Section 3, we show that the problem is NP-hard for $k > 1$. Note, however, that for certain classes of polygons such as spiral polygons, a smallest 2-guarding can be found efficiently [3]. The hardness of the problem motivates consideration of approximation algorithms.

For multi- k -guarding, one option is to use k copies of a ρ -approximation of a smallest 1-guarding to produce a $k\rho$ -approximation of a smallest multi- k -guarding. For k -guarding, this is not an option since a k -guarding cannot place multiple guards at a single vertex. Even a k -step

*Department of Computer Science, University of British Columbia, {busto,will,kirk}@cs.ubc.ca

[†]Belleville [1] uses the term "two-guarding" to mean guarding the entire polygon using only two guards, which is different from our usage.

approach that approximates an optimal 1-guarding using only vertices that don't appear as guards in previous steps is difficult to make work; the relationship between the size of the resulting k -guarding and a smallest k -guarding is not clear.

Fusco and Gupta [9] introduced the notion of (k, ε) -nets to find k -covers of sets in the context of sensor networks. Their result implies an $O(\log \text{OPT}_k(P))$ -approximation algorithm for minimally k -guarding P . Chekuri, Clarkson, and Har-Peled [5] described a technique for set multicover with potentially different coverage requirements for each set that also implies an $O(\log \text{OPT}_k(P))$ -approximation algorithm. We describe a polynomial time $O(k \log \log \text{OPT}_k(P))$ -approximation algorithm that generalizes the technique King and Kirkpatrick [11] used to show an $O(\log \log \text{OPT}_1(P))$ -approximation algorithm for 1-guarding P .

2 k -guarding as k -hitting

A k -hitting set of a set \mathcal{R} of sets is a set H such that $|H \cap R| \geq \min\{k, |R|\}$ for all $R \in \mathcal{R}$. Let V_p be the set of potential guards that can see p . Let $\mathcal{R} = \{V_p \mid p \in P\}$. A k -hitting set of \mathcal{R} is a k -guarding of P .

Let V be the set of n potential guard locations. Let $w(v)$ be the weight of potential guard location v and $w(R) = \sum_{v \in R} w(v)$. Rather than k -hitting every set in \mathcal{R} , we will be happy with k -hitting the heavy sets, those with weight at least $\varepsilon w(V)$. Such a k -hitting set is called a (k, ε) -net for the weighted set system (V, \mathcal{R}, w) . The algorithm for finding a k -hitting set initially sets these weights to 1 and updates them so that a (k, ε) -net for the weighted set system will be a k -hitting set for \mathcal{R} .

Suppose we can find a (k, ε) -net for (V, \mathcal{R}, w) of size $s(1/\varepsilon)$ for some function s . We slightly modify the technique of Brönnimann and Goodrich [4] to obtain a k -hitting set of size $s(4c)$ where c is the smallest k -hitting set. Assume we know that c is the size of the smallest k -hitting set. We find a $(k, 1/2c)$ -net N of size $s(2c)$. If there is a set $R \in \mathcal{R}$ (called a *witness*) that is not k -hit, we double the weight of the guards in R that are not in the net N . (The italics indicate our modification.) Since $w(R) \leq w(V)/2c$, the weight of V is multiplied by at most a factor $1 + 1/2c$. If H is a k -hitting set of size c , $(H \cap R) \setminus N$ is not empty and the weight of some element in H is doubled. It follows from the proof of Lemma 3.4 [4] that:

Lemma 1 *If there is a k -hitting set of size c , the weight-doubling process iterates at most $4c \lg(n/c)$ times before finding a k -hitting set.*

The rest of their algorithm is a doubling search for the correct value of c . Since the search will succeed when

the guess is at most $2c$, the size of the k -hitting set is at most $s(4c)$. The running time is dominated by the time for the last weight-doubling process which involves $O(c \log(n/c))$ invocations of the (k, ε) -net finder (and witness finder).

King and Kirkpatrick [11] show how to find a $(1, \varepsilon)$ -net of size $O(\frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon})$ for the weighted set systems arising from 1-guarding. To approximately k -guard, we require small (k, ε) -nets for the weighted set systems arising from k -guarding. In Section 6, we show how to find such (k, ε) -nets of size $O(\frac{k}{\varepsilon} \log \log \frac{1}{\varepsilon})$. It will be important that the weight of each element remains small. Our modified weight-doubling process insures this. A possible alternative is to solve a linear program to obtain the weights before invoking an ε -net finder once [14, 7]. However, it seems difficult to modify the program to insure that the weights remain small enough that our (k, ε) -net finder will produce a near-optimal k -hitting set.

3 Hardness of k -guarding

Lee and Lin [13] show that minimally 1-guarding a simple polygon is NP-hard. We extend this result to k -guarding for $k > 1$.

Theorem 2 *Finding a minimum k -guarding of a simple polygon is NP-hard for all $k > 1$.*

Proof. We reduce from the problem of minimally 1-guarding a terrain (an x -monotone polygonal curve), which is known to be NP-hard [12]. Given a terrain T , place a concave, x -monotone path v_1, v_2, \dots, v_{k-1} sufficiently far above the terrain so that each v_i can see all of T . This can be done in polynomial time by intersecting the positive halfplanes coincident with edges of T . Create a polygon P by connecting v_1 to the leftmost and v_{k-1} to the rightmost vertex of T , as shown in Fig. 1. Now P contains a k -guarding of size h if and only if T contains a 1-guarding of size $h - k + 1$. Clearly, adding v_1, v_2, \dots, v_{k-1} to any 1-guarding of T creates a k -guarding of P . Let G be a k -guarding of P . Let ℓ be the number of v_i in G . Removing all v_i and any $k - 1 - \ell$ terrain vertices from G creates a 1-guarding of T . \square

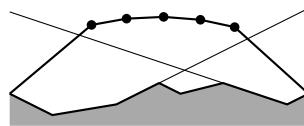


Figure 1: The construction used to show k -guarding is NP-hard (for $k = 6$).

A simple extension of Lin and Lee's original proof shows that minimally multi- k -guarding a simple polygon is NP-hard, but only for odd k . Their proof is based

on “critical locations” for guards; a guard at one location is equivalent to assigning a variable true and at the other to false. With odd k whichever location is given more guards can be interpreted as the choice for the corresponding variable. With even k an optimal guard set may have the same number of guards at each location.

4 Distinguished vertices

We describe in this section a method for choosing vertex guards in a vertex-weighted polygon P so that any vertex that sees a large fraction of the total weight will be seen by k guards. The basic idea is to partition the boundary of P into fragments of consecutive edges so that all fragments have approximately equal weight, and to place guards at extreme points of visibility between fragments. Lemma 3 implies that this is a good choice of guards. In the following section, we adopt a hierarchical fragmentation scheme to insure that the size of the guard set is small.

Let ∂P denote the boundary of polygon P . We refer to any sequence of consecutive edges on ∂P as a (*boundary*) *fragment* of P . Two fragments are *adjacent* if they share an endpoint. We say that a fragment f is *visible* from a point p of P if there is a point q on f such that the open line segment pq lies in the interior of P . The *extreme points of visibility* of fragment f from a set of points S are the first point in f visible from some point in S and the last point in f visible from some point in S . It follows from the simplicity of P that if f is visible from p then the points on f that are visible from p span a contiguous sector of angles centered at p determined by the extreme points of visibility of f from p . If the sectors of one or more visible fragments together have a span greater than π then the corresponding fragments are said to *surround* p ; clearly at most one fragment has this property in isolation. Two fragments a and b , visible from p , are *clockwise consecutive from p* if the clockwise extreme visibility angle of a coincides with the counterclockwise extreme visibility angle of b . Suppose that a and b are clockwise consecutive from p . Then, if the clockwise extreme visibility point of a is no further from p than the counterclockwise extreme visibility point of b , then a is said to support a right tangent from p ; otherwise b is said to support a left tangent from p , Fig. 2(a).

Following King and Kirkpatrick [11], we consider various partitions of ∂P into fragments, based in part on the weights associated with the vertices of P .² For any such fragmentation F of ∂P , we *distinguish* those vertices of P that coincide with extreme points of visibility between some pair of fragments in F , Fig. 2(b).

²As indicated earlier, there is a subtle, though significant, flaw in the construction described in [11]. This is discussed in more detail in Appendix A.

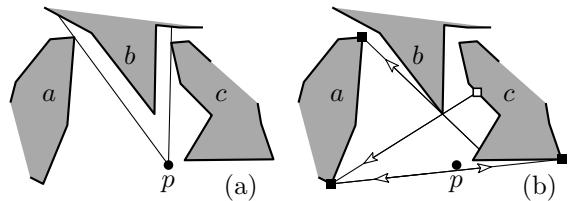


Figure 2: Fragments a , b , and c are clockwise consecutive from p . (a) Fragment a supports a right tangent and c supports a left tangent from p . (b) Squares are extreme points of visibility between fragments a and c . Filled squares are distinguished vertices.

The intuition that these distinguished vertices serve as a good choice for guard locations is based on the following geometric lemma:

Lemma 3 (Lemma 2 [11]) *Let a and b be clockwise consecutive fragments of a fragmentation F , visible from a point p , that do not surround p . If a supports a left tangent from p (resp. if b supports a right tangent from p) then p sees at least one of the distinguished vertices on a (resp. on b).*

The preceding lemma is enough to show that if p sees many visible fragments then it must see many different distinguished vertices, which is a generalization of King and Kirkpatrick’s Lemma 1 for $k \geq 1$:

Lemma 4 *Any point p of P that sees at least $2k + 3$ fragments of F in total must see a distinguished vertex on at least k fragments.*

Proof. If p sees at least $2k + 3$ fragments then it sees a set T of at least $2k$ consecutive fragments each of which cannot pair with a consecutive fragment to surround p , otherwise there would be two disjoint pairs that both span more than π . If a fragment in T doesn’t have a distinguished vertex then it has no tangent from p and its consecutive fragment(s) in T have tangents and thus distinguished vertices, by Lemma 3. Hence, at least $|T|/2 \geq k$ fragments have distinguished vertices. \square

Remark. If we divide ∂P into $\frac{2k+2}{\varepsilon}$ equal weight fragments where each vertex has weight 1 then it follows that every point p that sees more than a fraction ε of the total weight $w(V) = n$, sees more than $2k + 2$ fragments, and by Lemma 4, must see at least k among the set of $O(\frac{k^2}{\varepsilon^2})$ distinguished vertices associated with this fragmentation. Thus the distinguished vertices form a (k, ε) -net for the weighted set system $(V, \mathcal{R}, w(v) = 1)$. It remains to find other fragmentations that will work for more weight functions and whose associated distinguished vertices will provide a smaller net.

5 A net-finder based on hierarchical fragmentation

In the last section we showed that by placing guards at a set of $O\left(\frac{k^2}{\varepsilon^2}\right)$ distinguished vertices associated with a flat fragmentation of ∂P , we can ensure that any point that sees fewer than k of these guards sees less than an ε fraction of the total weight. In this section we discuss how hierarchical fragmentation can be used to reduce the number of guards required to $O\left(\frac{k}{\varepsilon} \log \log \frac{1}{\varepsilon}\right)$. It will suffice to prove the result for $k \leq \frac{\lg(1/\varepsilon)}{r \lg \lg(1/\varepsilon)}$, for a sufficiently large constant r , since otherwise the (k, ε) -net-finder of Fusco and Gupta [9] or Chekuri, Clarkson, and Har-Peled [5] can be used. It will be clear that the construction of our (k, ε) -net takes polynomial time. (In fact, with some care, it can be constructed in $O(n^3)$ time [10].)

We can represent the hierarchical fragmentation as a tree. At the root there is a single fragment representing the entire boundary ∂P . This root fragment is broken up into a certain number of child fragments. Fragmentation continues recursively until a specified depth t is reached. The integer t is chosen so that

$$(t-1) + 2^{t-1} < \lg \frac{1}{\varepsilon} \leq t + 2^t. \quad (1)$$

Note that $\lg \lg \frac{1}{\varepsilon} - 1 < t < \lg \lg \frac{1}{\varepsilon} + 1$, which implies $2^t/t > \frac{\lg(1/\varepsilon)}{2(\lg \lg(1/\varepsilon)+1)} > \frac{\lg(1/\varepsilon)}{3 \lg \lg(1/\varepsilon)}$, (provided $\varepsilon < 1/16$), which by our assumption is at least $rk/3$.

The number of children of a fragment f depends on both t and the level of f in the tree. Specifically, if $e = t + 2^t - \lceil \lg \frac{1}{\varepsilon} \rceil$ (note $0 \leq e \leq 2^{t-1}$), then b_i , the number of children of fragments at level $i-1$, is:

$$b_i = \begin{cases} \beta_k t \cdot 2^{2^{t-1}+1} \cdot 2^{1-e} & , \quad i = 1 \\ 2^{2^{t-i}+1} & , \quad 1 < i \leq t, \end{cases}$$

where β_k is a linear function of k that will be specified later. Let ϕ_i be the number of fragments at level i .

$$\phi_i = \begin{cases} 1 & , \quad i = 0 \\ \beta_k t \cdot 2^{2^t-2^{t-i}-e+i+1} & , \quad 0 < i \leq t \end{cases}$$

since $\phi_i = \prod_{j=1}^i b_j = \beta_k t \cdot 2^{1-e} \cdot \prod_{j=1}^i 2^{2^{t-j}+1} = \beta_k t \cdot 2^{1-e+i+\sum_{j=1}^i 2^{t-j}} = \beta_k t \cdot 2^{2^t-2^{t-i}-e+i+1}$. Note that

$$\phi_t = \beta_k t \cdot 2^{2^t+t-e} = \beta_k t 2^{\lceil \lg \frac{1}{\varepsilon} \rceil} \geq \beta_k t \frac{1}{\varepsilon}. \quad (2)$$

Each collection of child fragments with the same parent fragment f , together with the complement \bar{f} of f , defines a fragmentation F_f of ∂P . The guard set, D_{HF} , is the union, over all parent fragments f in the tree, of the set of vertices that are distinguished by fragmentation F_f . The total number of guards chosen is

$$|D_{HF}| \leq 4 \sum_{i=1}^t \binom{b_i + 1}{2} \phi_{i-1} \leq b_1^2 + 4 \sum_{i=2}^t b_i^2 \phi_{i-1}.$$

Since

$$\begin{aligned} b_1^2 &= (\beta_k t \cdot 2^{2^{t-1}+1} \cdot 2^{1-e})^2 = \beta_k t \cdot 2^{2^t+t-e} (\beta_k t \cdot 2^{4-t-e}) \\ &< \beta_k t \cdot 2^{\lceil \lg \frac{1}{\varepsilon} \rceil} \cdot (\beta_k t \cdot 2^{4-t-e}) \leq 2\beta_k t \cdot \frac{1}{\varepsilon}, \end{aligned}$$

for $2^t/t \geq 16\beta_k$ (which holds when $rk/3 \geq 16\beta_k$), and

$$\begin{aligned} \sum_{i=2}^t b_i^2 \phi_{i-1} &= \sum_{i=2}^t b_i \phi_i = \sum_{i=2}^t (2^{2^{t-i}+1}) (\beta_k t 2^{2^t-2^{t-i}-e+i+1}) \\ &= \beta_k t \cdot 2^{2^t-e+2} \sum_{i=2}^t 2^i < \beta_k t \cdot 2^{2^t+t-e+3} \\ &\leq 8\beta_k t \cdot 2^{\lceil \lg \frac{1}{\varepsilon} \rceil} \leq 16\beta_k t \cdot \frac{1}{\varepsilon}, \end{aligned}$$

we know,

$$|D_{HF}| = O\left(\beta_k \frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon}\right). \quad (3)$$

We must now provide a generalization of Lemma 4 that works with our hierarchical fragmentation.

Lemma 5 *Any point p in P that sees fewer than k vertices in D_{HF} sees no more than $(8k+4)i+1$ fragments at level i of the hierarchical fragmentation.*

The proof of Lemma 5 uses a potential function defined on fragments. A *narrow* fragment is a visible fragment with sector at most π . A *wide* fragment is a visible fragment that isn't narrow. The *potential* of a visible fragment f (wide or narrow) is given by $2g+1-t$, where g is the number of guards in D_{HF} from fragment f that see p , and t is the number of tangents from p to f . Note that g counts all the guards on f in D_{HF} , which includes distinguished vertices in fragmentations F_a for all descendants a of f in the tree. We call a visible fragment *potent* if it is wide or it is narrow and has positive potential; otherwise we call it *impotent*. In addition to potent and impotent fragments, both of which are visible to p , there are also non-visible fragments in the tree, which we regard as having potential zero.

Lemma 6 *The potential of f is at least the total potential of its children.*

Proof. Let f_1, f_2, \dots, f_c be the visible children of f . Let g_i be the number of guards in D_{HF} from f_i that see p , and t_i be the number of tangents from p to f_i . We want to show that $2g+1-t \geq \sum_{i=1}^c (2g_i+1-t_i)$. The number of tangents to \bar{f} is $2-t$ and the total number of tangents to visible fragments in F_f is $c+1$. Thus $2-t+\sum_{i=1}^c t_i = c+1$, which implies $\sum_{i=1}^c (1-t_i) = 1-t$. The lemma follows since $2g \geq \sum_{i=1}^c 2g_i$. \square

Lemma 7 *An impotent fragment f has at most one visible child and that child is impotent.*

Proof. Let f be an impotent fragment with potential $2g + 1 - t \leq 0$, which implies $g = 0$ and $t \geq 1$. If f has two visible children then the (at least one) child with f 's tangent contains a visible distinguished vertex by Lemma 3, which contradicts $g = 0$. Thus f has at most one visible child and, since it shares all tangents with f , its potential is at most $1 - t \leq 0$. \square

Lemma 8 *The number of potent fragments at any level is at most the total potential at that level plus two.*

Proof. Potent fragments have positive potential, except for the (at most one) wide fragment that has potential at least -1 . \square

Proof of Lemma 5. If p sees fewer than k vertices in D_{HF} then the potential of the root is at most $2(k-1)+1$. By Lemmas 6 and 8, the number of potent fragments at any level is at most $2(k-1)+3$. By Lemma 7, every impotent fragment has at most one visible child, which is impotent. Since an impotent child cannot contain a distinguished vertex, any fragment can have at most four impotent children (by Lemma 4 with $k = 1$). Thus every potent fragment has at most four impotent children and its other visible children all have positive potential. Since the total number of potent fragments remains at most $2(k-1)+3$, the number of visible fragments increases by at most $4 \times \#$ potent fragments $\leq 8k+4$ at each level. Thus the number of visible fragments at level i is no more than $(8k+4)i+1$. \square

6 Near optimal k -guarding

The discussion of the hierarchical fragmentation scheme in the previous section did not take into account the weight of the vertices V of P in the weighted set system (V, \mathcal{R}, w) . The weights play a role in the selection of the children of a parent fragment. We choose the children to have approximately equal weight and to contain an integral number of vertices. Both these requirements may be impossible to satisfy if the weights of vertices are very different. The potential problem is that a fragment at level i may have fewer than b_i vertices and thus cannot produce b_i child fragments in the hierarchy. To address this problem, we keep the weights of all vertices small and build the tree bottom up.

Let w_{\max} be the maximum weight of a vertex $v \in V$. Let $\gamma = w(V)/\phi_t$ be the target weight of a leaf, i.e. a fragment at level t in the tree. We create the tree from the bottom up by fragmenting the perimeter of the polygon P into ϕ_t leaf fragments where each leaf fragment f has weight $w(f)$ that satisfies $\gamma - w_{\max} \leq w(f) \leq \gamma + w_{\max}$. One way to do this is to imagine fragmenting the perimeter into ϕ_t equal weight pieces, which may split some vertices in two, and then putting any vertex that is split into the last (clockwise-most)

fragment in which it occurs. As long as $w_{\max} < \gamma$, the resulting fragmentation has at least one vertex in every fragment. We then combine each collection of b_i adjacent fragments ($i = t$) to form their parent fragment at level $i - 1$ and repeat for all i down to $i = 1$. This creates the tree from which we extract the set of guards D_{HF} . Applying Lemma 5 with $i = t$, choosing $\beta_k = 8k+5$, and using equations (2) and (3), we get

Lemma 9 *Given a weighted set system (V, \mathcal{R}, w) such that $w(v) < \varepsilon w(V)/(\beta_k t)$ for all $v \in V$ where t satisfies equation (1), the set D_{HF} is a (k, ε) -net for (V, \mathcal{R}, w) of size $O(\frac{k}{\varepsilon} \log \log \frac{1}{\varepsilon})$.*

To keep our slightly modified version of the technique of Brönnimann and Goodrich [4] from increasing the weight of any vertex to γ or more, we add all vertices v of weight $w(v) \geq \gamma/2$ to D_{HF} . Since no such vertex has its weight doubled, because of our modification, the maximum weight that any vertex can attain is less than γ . The number of such vertices is at most $2w(V)/\gamma \leq 2\beta_k t \frac{1}{\varepsilon}$, which increases the size of D_{HF} by only a (small) constant factor. Our main theorem follows:

Theorem 10 *For a simple polygon P with n vertices, we can find a k -guarding set of vertices for P of size $O(k \text{OPT}_k(P) \log \log \text{OPT}_k(P))$, where $\text{OPT}_k(P)$ is the size of the minimum k -guarding set of vertices for P , in time polynomial in n .*

7 Extensions and open questions

We have shown how to get an $O(k \log \log \text{OPT}_k(P))$ -approximation to the minimum k -guarding of a simple polygon P . This improves the $O(\log \text{OPT}_k(P))$ -approximation algorithms of Fusco and Gupta [9] and Chekuri, Clarkson, and Har-Peled [5], when $k = o\left(\frac{\log \text{OPT}_k(P)}{\log \log \text{OPT}_k(P)}\right)$. It would be interesting to know if the dependence on k in our algorithm can be eliminated or reduced.

Our version of k -guarding models situations in which the guarding requirement of every point of P is the same. We addressed the non-uniformity that arises simply because points of P may not see as many as k vertices of P , by reducing the guarding requirement of a point p to the minimum of k and the number of vertices that see p . In general, a uniform guarding requirement might be undesirable for other reasons as well; some (perhaps most) points may not need to be guarded at all, whereas others may need extraordinary attention. To capture this variety, we suppose that all points $p \in P$ have an associated non-negative guarding demand $d(p)$, and we seek a *demand-guarding* of P , a guard set G that satisfies the individual guarding demand of every point in P . The associated optimization problem takes as input a pair (P, d) and asks for a minimum size demand-guarding of P .

In fact, the $O(\log \text{OPT})$ -approximation algorithm of Chekuri, Clarkson, and Har-Peled [5] addresses this more general demand-guarding problem, where OPT denotes the size of the optimal demand-guarding set. Obviously, our algorithm could be used to provide an $O(d_{\max} \log \log \text{OPT}_{d_{\max}})$ -approximation (by increasing all demands to d_{\max}). However, since our algorithm never exploits the uniformity of guarding demands, it is straightforward to confirm that, essentially without modification, it provides an $O(d_{\max} \log \log \text{OPT})$ -approximation. Once again, it would be interesting to know if the dependence on d_{\max} can be eliminated or reduced.

Our k -guarding problem assumes that (i) all of the vertices of the polygon are potential guard locations, and (ii) the polygon has no holes. It would be interesting to determine if our techniques can be used to get good approximation algorithms when these constraints are relaxed.

Existing NP-hardness proofs for minimally 1-guarding polygons extend easily to show that minimally multi- k -guarding simple polygons is NP-hard when k is odd. It seems likely that minimally multi- k -guarding is NP-hard when k is even as well.

There is no known non-trivial lower bound on the ratio between the sizes of a minimum k -guarding and a minimum 1-guarding of a polygon. What is the precise relationship between these two values? Are there classes of polygons where the size of a minimum 2-guarding is not much larger than a minimum 1-guarding?

References

- [1] P. Belleville. Two-guarding simple polygons. In *Proc. 4th Canadian Conference on Computational Geometry*, pages 103–108, 1992.
- [2] P. Belleville, P. Bose, J. Czyzowicz, J. Urrutia, and J. Zaks. K-guarding polygons on the plane. In *Proc. 6th Canadian Conference on Computational Geometry*, pages 381–386, 1994.
- [3] S. Bereg. On k -vertex guarding simple polygons. In *Computational Geometry and Discrete Mathematics*, volume 1641 of *Kôkyûroku*, pages 106–113. Research Institute for Mathematical Sciences, Kyoto University, October 2008.
- [4] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
- [5] C. Chekuri, K. L. Clarkson, and S. Har-Peled. On the set multicover problem in geometric settings. *ACM Trans. Algorithms*, 9(1):9:1–9:17, Dec. 2012.
- [6] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.
- [7] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005.
- [8] S. Fisk. A short proof of Chvátal’s Watchman Theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.
- [9] G. Fusco and H. Gupta. ε -net approach to sensor k -coverage. In B. Liu, A. Bestavros, D.-Z. Du, and J. Wang, editors, *Wireless Algorithms, Systems, and Applications*, volume 5682 of *Lecture Notes in Computer Science*, pages 104–114. Springer Berlin Heidelberg, 2009.
- [10] J. King. Fast vertex guarding for polygons with and without holes. *Computational Geometry*, 46(3):219–231, 2013.
- [11] J. King and D. G. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete & Computational Geometry*, 46(2):252–269, 2011.
- [12] J. King and E. Krohn. Terrain guarding is NP-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.
- [13] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.
- [14] P. M. Long. Using the pseudo-dimension to analyze approximation algorithms for integer programming. In F. K. H. A. Dehne, J.-R. Sack, and R. Tamassia, editors, *WADS*, volume 2125 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2001.
- [15] K. Mehlhorn, J. Sack, and J. Zaks. Note on the paper “K-vertex guarding simple polygons” [Computational Geometry 42 (4) (May 2009) 352–361]. *Computational Geometry*, 42(6–7):722, 2009.
- [16] J. O’Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.
- [17] I. Salléh. K-vertex guarding simple polygons. *Computational Geometry*, 42(4):352–361, 2009.
- [18] J. Urrutia. Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of computational geometry*, pages 973–1027. North-Holland Publishing Co., 2000.

A Remark on the hierarchical fragmentation construction of King and Kirkpatrick [11]

A very similar hierarchical fragmentation (differing from ours only in the definition of t , and in the level-1 fragmentation factor b_1) was described by King and Kirkpatrick [11] in developing their approximation bound for optimal 1-guarding. Unfortunately, the choice of α (which, together with t determines b_1) given in their equation (3) does not always guarantee that their equation (1) holds. In particular, consider the case when $1/\varepsilon = 2^{2^{t-1}+1}$ (so $t = \lceil \log \log(1/\varepsilon) \rceil$, as specified). In this case, $\alpha = 1/(4t2^{t-1}+1-t)$ and so $t\alpha 2^t$ (the bound on $|S_{HF}|$, the size of their guard set) is essentially $2^t \cdot 1/\varepsilon$, which is $\Theta((1/\varepsilon) \log(1/\varepsilon))$, not $O((1/\varepsilon) \log \log(1/\varepsilon))$, as claimed in their equation (1).

Geometric Red-Blue Set Cover for Unit Squares and Related Problems

Timothy M. Chan*

Nan Hu*

Abstract

We study a geometric version of the RED-BLUE SET COVER problem originally proposed by Carr, Doddi, Konjevod, and Marathe (SODA 2000): given a red point set, a blue point set, and a set of objects, we want to use objects to cover all the blue points, while minimizing the number of red points covered. We prove that the problem is NP-hard even when the objects are unit squares in 2D, and we give the first PTAS for this case. The technique we use simplifies and unifies previous PTASes for the weighted geometric set cover problem and the unique maximum coverage problem for 2D unit squares.

1 Introduction

Given a red set R and a blue set B of total size m , and a family \mathcal{S} of n subsets of $R \cup B$, the RED-BLUE SET COVER problem is to find a subfamily of \mathcal{S} which covers all the elements in B , but covers the minimum number of elements in R .

The problem was first introduced by Carr, Doddi, Konjevod and Marathe [1], who proved that even in the restricted case where every set in \mathcal{S} contains only one blue and two red elements, the problem cannot be approximated to within $2^{\log^{1-\delta} n}$ factor for $\delta = 1/\log^c n$ and for any constant $c < 1/2$, unless P = NP. Carr et al. also gave a $2\sqrt{n}$ -approximation algorithm for the case where every set in \mathcal{S} contains only one blue element.

We study a geometric version of RED-BLUE SET COVER where the elements of R and B are points, and the sets of \mathcal{S} are geometric objects. Specifically, we focus on the case where the objects are unit squares¹ in 2D; we call the resulting problem RED-BLUE UNIT-SQUARE COVER. We prove that RED-BLUE UNIT-SQUARE COVER remains NP-hard, and we present a PTAS (i.e., a polynomial-time $(1 + \varepsilon)$ -approximation algorithm for any constant $\varepsilon > 0$) for this problem.

Previous work on PTASes. There have already been a number of PTASes for problems related to geometric set cover and hitting set in the literature. To put our new PTAS into context, we note that most of the known techniques can be classified into a few categories:

*Cheriton School of Computer Science, University of Waterloo,
 {tmchan,n3hu}@uwaterloo.ca

¹All squares in this paper are assumed to be axis-aligned.

1. Hochbaum and Maass' original *shifted grid* technique [8]. This is among the earliest PTAS techniques developed, and is usually applicable only to “continuous” versions of geometric set cover and hitting set problems. For example, in the continuous version of the standard (monochromatic) UNIT-SQUARE COVER problem, we want the smallest number of unit squares to cover a given point set, where the allowed unit squares can be located anywhere rather than from a given (“discrete”) set. When applicable, the technique is general enough to handle other types of similar-sized fat objects, such as unit disks in 2D, or unit balls in higher fixed dimensions. Extensions of the technique based on *shifted quadtrees* have also been explored for some related problems [2].
2. Mustafa and Ray's *local search* technique [10]. This yields the first PTAS for the general discrete version of the UNIT-SQUARE COVER and the analogous UNIT-DISK COVER problem in 2D. However, the technique inherently does not work for weighted problems. For example, in the WEIGHTED UNIT-SQUARE COVER problem, given a point set and a set of unit squares each with a positive weight, we want a subset of unit squares of the smallest total weight to cover the given point set.
3. Sophisticated *dynamic programming* combined with Hochbaum and Maass' shifting technique. Erlebach and van Leeuwen [5] used this approach to obtain the first PTAS for WEIGHTED UNIT-SQUARE COVER. Recently, Ito et al. [9] have also applied a similar approach to obtain a PTAS for the following variant of unit-square cover called UNIQUE UNIT-SQUARE COVERAGE: given a point set and a set of unit squares, we want a subset of unit squares to maximize the number of points that are covered exactly once. (Erlebach and van Leeuwen introduced the general unique coverage problem for sets in 2008 [4].) At the moment, these PTASes are limited to the special case of 2D unit squares and do not seem generalizable to unit disks or to higher dimensions.

Our PTAS belongs to the third category and is similar to Erlebach and van Leeuwen's and Ito et al.'s PTASes. For example, our approach can easily handle a weighted version of RED-BLUE UNIT-SQUARE COVER, where the

red points have weights and we want to minimize the total weight of the red points covered. However, our approach works only for unit squares and not for other types of objects.

Arguably the most interesting aspect of this paper lies not so much in the specific result about red-blue set cover, but in our technique, which we feel is conceptually simpler than Erlebach and van Leeuwen’s and Ito et al.’s PTASes [5, 9] for WEIGHTED UNIT-SQUARE COVER and UNIQUE UNIT-SQUARE COVERAGE. In fact, our technique leads to alternative PTASes for these two problems as well, and can potentially be more easily applied to other variants of set cover problems for unit squares.

The descriptions of the dynamic programming algorithm in both papers [5, 9] are lengthy. For example, the algorithm by Erlebach and van Leeuwen is obtained by simulating a plane sweep that involves multiple sweep lines moving at different speeds. We get around most of the complications by one very simple idea: a “mod-one trick”.

In section 2, we give the NP-hardness proof of RED-BLUE UNIT-SQUARE COVER. We introduce the mod-one trick and give a PTAS in section 3, followed by a brief discussion on how to apply our technique to other problems in section 4.

2 NP-Hardness

Theorem 1 RED-BLUE UNIT-SQUARE COVER is NP-hard.

Proof. We reduce from the vertex cover problem on degree-3 planar graphs, which is well known to be NP-hard [7].

Lemma 2 [3] Every planar graph $G = (V, E)$ of maximum degree at most 4 has an orthogonal planar drawing on an $O(|V|) \times O(|V|)$ grid (i.e., vertices are placed at grid points and edges are drawn as a rectilinear polygonal chain with corners at grid points, with no crossings).

Lemma 3 (Folklore) Given a graph G and an edge e in G , define a new graph G' obtained from G by subdividing e through the addition of two new “dummy” vertices. Then the size of a minimum vertex cover of G' is precisely the size of a minimum vertex cover of G plus 1.

Given a degree-3 planar graph G with n vertices, we create an orthogonal drawing by Lemma 2. We define a new graph G' by subdividing each edge e through the addition of new dummy vertices at each grid point along e . Each edge in G' is now a horizontal or vertical line segment of length 1 in the drawing. If e contains an odd number of dummy vertices, we insert an extra

new dummy vertex at the midpoint of a line segment. Then all edge lengths in G' are $1/2$ or 1 . By rescaling by a factor slightly less than 2, we can ensure that all edge lengths in G' are strictly between $2/3$ and 2 . Now, each edge in the original graph G has an even number of dummy vertices, and by repeated applications of Lemma 3, finding the size of the minimum vertex cover of G is equivalent to finding the size of the minimum vertex cover of G' .

To construct an instance of RED-BLUE UNIT-SQUARE COVER from G' , we replace each vertex in G' by a red point r_i . For each edge $r_i r_j$ in G' , we create a blue point b_{ij} in the middle of the edge and add a unit square containing precisely b_{ij} and r_i and a unit square containing precisely b_{ij} and r_j . See Figure 1. Such squares exist since the distance between two adjacent blue and red points is strictly between $1/3$ and 1 .

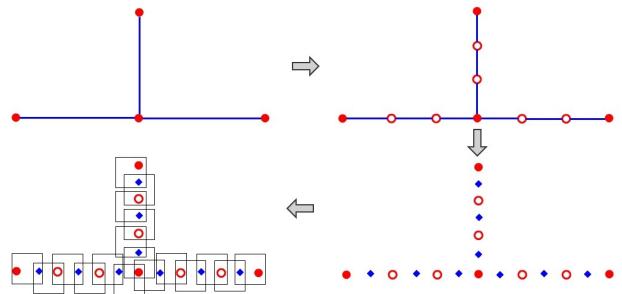


Figure 1: The reduction from vertex cover. (Red points are drawn as dots, and blue points are drawn as diamonds.)

Correctness of the reduction is easy to see: Given a vertex cover of G' of size k , we can select all the squares that cover the corresponding k red points; these squares would cover all blue points. Conversely, given a subset of squares covering all blue points, the red points covered by these squares form a vertex cover of G' . \square

3 PTAS

We now present a PTAS for RED-BLUE UNIT-SQUARE COVER. We begin with a definition:

Definition 4 Let $S = \{s_1, \dots, s_t\}$ be a set of unit squares, where s_1, \dots, s_t are arranged in increasing x -order of their centers. We say that S forms a monotone set, if the centers of s_1, \dots, s_t are in increasing or decreasing y -order.

Note that the boundary of the union of the squares in a monotone set S consists of two monotone chains (“staircases”), as shown in figure 2. We say that these two chains are *complementary*.

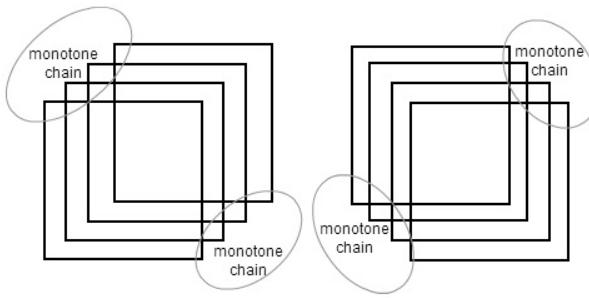


Figure 2: A monotone set may be “increasing” (left) or “decreasing” (right).

Lemma 5 *Let OPT be an optimal solution for an instance where the blue point set B is inside a $k \times k$ square. Then OPT can be decomposed into $O(k^2)$ monotone sets.*

Proof. We may assume that all squares in OPT intersect the $k \times k$ square. Draw a grid with unit side length over the $k \times k$ square. Consider a grid point p . Let $S(p)$ be the set of squares in OPT containing p ; every square in OPT belongs to one of the $S(p)$ ’s. Let $U(p)$ denote the boundary of the union of the squares of $S(p)$. We may assume that each square in $S(p)$ appears on $U(p)$, for otherwise we could remove the square from OPT and the resulting solution is no worse than OPT.

Divide the plane into 4 quadrants at p . For each $i \in \{1, 2, 3, 4\}$, let $S_i(p)$ be the subset of squares in OPT containing p that contributes to the portion of $U(p)$ inside the i -th quadrant. Then each $S_i(p)$ is a monotone set. Thus, we have decomposed OPT into $4(k+1)^2$ monotone sets. (These sets may not be disjoint, but can be made disjoint by deleting elements from sets, since a subset of a monotone set is still monotone.) \square

The heart of our PTAS is an exact dynamic programming solution for the special case of the problem where all points are inside a $k \times k$ grid for a constant k . The idea is to use a sweep-line algorithm to guess the $O(k^2)$ monotone sets at the same time. We can “remember” a constant number ($O(k^2)$) of intersections of the monotone chains with a vertical sweep line as we sweep from left to right. However, each monotone set defines two complementary monotone chains, and the guess of one chain should be consistent with the guess of its complementary chain; but by the time the sweep line gets to the second chain, we would have forgotten information about the first chain. This is why Erlebach and van Leeuwen [5] needed a more complicated approach involving multiple sweep lines moving at different speeds.

To avoid this difficulty, we overlay all the monotone sets into one grid cell by introducing a “mod-one” transformation:

Definition 6 *We define the mod-one mapping $(x, y) \mapsto (x \bmod 1, y \bmod 1)$, where $z \bmod 1$ denotes the fractional part of a real number z .*

With this transformation, a unit square is rearranged into four pieces covering the unit grid cell, as shown in figure 3.

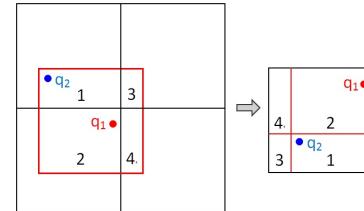


Figure 3: Applying the mod-one transformation to a unit square.

Furthermore, the union of the squares in a monotone set is rearranged as shown in figure 4. Notice that the two complementary monotone chains are mapped to two monotone chains that are connected at the corner points. This is the key property we need about the mod-one transformation. By redesigning the sweep-line algorithm to sweep over the unit grid cell in the transformed space, we can guess the two complementary monotone chains of each monotone set at the same time. The remaining pieces of the union consists of two rectangles defined by the start and end square of the monotone set; we can guess these two squares in advance.

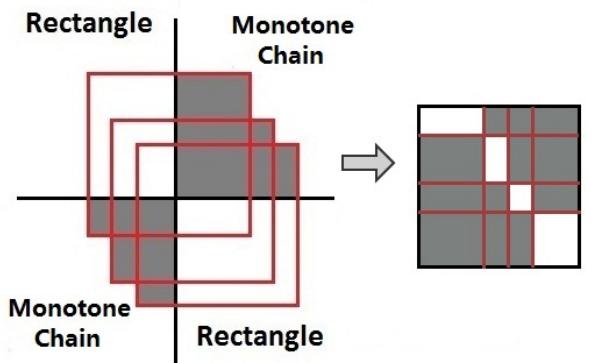


Figure 4: Applying the mod-one transformation to a monotone set.

Theorem 7 *For any instance of RED-BLUE UNIT-SQUARE COVER where B is inside a $k \times k$ square for a constant k , we can find the optimal solution in $O(mn^{O(k^2)})$ time.*

Proof. We find it best to describe our dynamic programming algorithm in terms of a state-transition diagram. We define a *state* to consist of

- a vertical sweep line ℓ that passes through a corner of an input square, after taking mod 1;
- $O(k^2)$ 4-tuples of the form $(s_{\text{start}}, s_{\text{prev}}, s_{\text{curr}}, s_{\text{end}})$, subject to the conditions that $s_{\text{start}}, s_{\text{prev}}, s_{\text{curr}}, s_{\text{end}}$ are in increasing x -order and form a monotone set, and that ℓ lies between the corners of s_{prev} and s_{curr} , mod 1.

Intuitively, a state represents current information about a decomposition of a solution into monotone sets at the sweep line (the monotone sets are not required to be disjoint). Specifically, each 4-tuple corresponds to a monotone set S ; s_{start} and s_{end} represent the start and end square of S ; and s_{prev} and s_{curr} represent the squares that define intersections of the sweep line with the two complementary monotone chains of S , after taking mod 1. These two squares s_{prev} and s_{curr} are adjacent in the monotone set S .

Given this state, we create a *transition* into a new state as follows: We pick the 4-tuple $(s_{\text{start}}, s_{\text{prev}}, s_{\text{curr}}, s_{\text{end}})$ such that the corner point of s_{curr} has the smallest x -coordinate, mod 1. The new sweep line ℓ' will be at the corner of s_{curr} . This 4-tuple is replaced by a new 4-tuple $(s_{\text{start}}, s_{\text{curr}}, s', s_{\text{end}})$ satisfying the stated conditions for some square s' . All other 4-tuples are unchanged. Let j_r (resp. j_b) be the number of red (resp. blue) points that lie between ℓ and ℓ' , after taking mod 1, and are covered (resp. not covered) by the squares in the $O(k^2)$ 4-tuples (before taking mod 1). If $j_b > 0$, we remove this transition. Otherwise, we set the cost of this transition to j_r .

The problem is thus reduced to finding the shortest path in this state-transition diagram (a directed acyclic graph), after adding suitable transitions involving start and end states. There are at most $O(mn^{O(k^2)})$ states, and each state has at most $O(n)$ outgoing transitions (since there are $O(n)$ choices for s'). Thus, we can construct the graph and find the shortest path by dynamic programming in $O(mn^{O(k^2)})$ time. \square

We can now apply Hochbaum and Maass' grid shifting technique [8] to obtain our final result:

Theorem 8 *There is a PTAS for RED-BLUE UNIT-SQUARE COVER.*

Proof. For each shift $a, b \in \{0, \dots, k-1\}$, let $S^{a,b}$ be the union of the solutions found by Theorem 7 for the blue points inside every $k \times k$ square $[ik+a, (i+1)k+a] \times [jk+b, (j+1)k+b]$, with $i, j \in \mathbb{Z}$. We return the $S^{(a,b)}$ with the smallest $c(S^{(a,b)})$, where $c(S)$ denotes the number of red points covered by S .

To analyze the approximation factor, let OPT be the optimal solution. Let $\text{OPT}^{a,*}$ (resp. $\text{OPT}^{*,b}$) be the subset of squares in OPT intersecting the lines $x = ik + a$ with $i \in \mathbb{Z}$ (resp. the lines $y = jk + b$ with $j \in \mathbb{Z}$). Since the algorithm in Theorem 2 covers the minimum number of red points for the subproblem for each $k \times k$ square, we have

$$c(S^{a,b}) \leq c(\text{OPT}) + 2c(\text{OPT}^{a,*}) + 2c(\text{OPT}^{*,b}).$$

Since $\sum_{0 \leq a < k} c(\text{OPT}^{a,*})$ and $\sum_{0 \leq b < k} c(\text{OPT}^{*,b})$ are both at most $2c(\text{OPT})$,

$$\sum_{0 \leq a, b < k} c(S^{a,b}) \leq (k^2 + 8k)c(\text{OPT}),$$

implying that

$$\min_{0 \leq a, b < k} c(S^{a,b}) \leq (1 + 8/k)c(\text{OPT}).$$

Setting $k = \lceil 8/\varepsilon \rceil$ gives a $(1 + \varepsilon)$ -approximation algorithm. \square

4 Related Problems

Weighted Unit-Square Cover. Erlebach and van Leeuwen [5] studied the following related problem: Given a set P of points and a set \mathcal{S} of unit squares in 2D where each square has a positive weight, we want to find a smallest-weight subset of \mathcal{S} to cover all the points in P .

Our algorithm can easily be modified to solve this problem. Specifically, in the proof of Theorem 7, if there are any points that lie between ℓ and ℓ' , after taking mod 1, and are not covered by any of the squares in the $O(k^2)$ 4-tuples, then we remove the transition. Otherwise, we set the cost of the transition to the weight of the square s' .

Budgeted Maximum Coverage for Unit Squares. Erlebach and van Leeuwen [5] also considered the following problem: Given a set P of points where each point has a positive profit value, and given a set \mathcal{S} of unit squares where each square has a positive cost, and given a budget B , we want to find a subset of \mathcal{S} with total cost at most B , maximizing the total profit of all points in P that are covered by the subset.

Erlebach and van Leeuwen [5] described how a modification of their dynamic programming algorithm combined with additional ideas can yield a PTAS for this problem. Our approach can be used to simplify the dynamic programming part of their PTAS.

Partial Unit-Square Cover. Gandhi et al. [6] studied the *partial set cover* problem. A geometric version can be stated as follows: Given a set P of points and a set \mathcal{S}

of unit squares in 2D, and given an integer K , we want to find a smallest subset of squares in \mathcal{S} to cover at least K points in P .

Gandhi et al. gave a PTAS for a continuous version of the problem based on Hochbaum and Maass' shifted grid technique [8]. For the discrete version, we can obtain a PTAS by using an appropriate modification of our dynamic programming algorithm, in conjunction with shifted grids as in Gandhi et al.'s paper.

Unique Unit-Square Coverage. Ito et al. [9] studied the following problem: Given a set P of points and a set \mathcal{S} of unit squares in 2D, find a subset of \mathcal{S} to maximize the number of points in P that are covered exactly once by the subset.

Again our algorithm can be modified to solve this problem. In the proof of Theorem 7, we use 6-tuples $(s_{\text{start}}, s_{\text{prev2}}, s_{\text{prev}}, s_{\text{curr}}, s_{\text{curr2}}, s_{\text{end}})$ instead of 4-tuples, where intuitively s_{prev2} represents the predecessor of s_{prev} and s_{curr2} represents the successor of s_{curr} in a monotone set. We set the cost of the transition to be the number of points that lie between ℓ and ℓ' , after taking mod 1, and are uniquely covered by the squares in the $O(k^2)$ 6-tuples. This works because squares that are not part of these 6-tuples are irrelevant as to whether a point is uniquely covered.

5 Conclusion

We have shown that RED-BLUE UNIT-SQUARE COVER is NP-hard, and have given a PTAS using a “mod-one” transformation. The main advantage of our PTAS is that it is simpler to describe than previous PTASes by Erlebach and van Leeuwen and Ito et al. for related problems [5, 9]. To be fair, we should mention that our $n^{O(1/\varepsilon^2)}$ running time is slower than the $n^{O(1/\varepsilon)}$ running time of the previous PTASes.

References

- [1] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proc. SODA*, pages 345–353, 2000.
- [2] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46:178–189, 2003.
- [3] B. N. Clark, C. J. Colbourn and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86:165–177, 1990.
- [4] T. Erlebach and E. J. van Leeuwen. Approximating geometric coverage problems. In *Proc. SODA*, pages 1267–1276, 2008.
- [5] T. Erlebach and E. J. van Leeuwen. PTAS for weighted set cover on unit squares. In *Proc. APPROX and RANDOM*, pages 166–177, 2010.
- [6] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53:55–84, 2004.
- [7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.
- [8] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985.
- [9] T. Ito, S.-I. Nakano, Y. Okamoto, Y. Otachi, R. Uehara, T. Uno, and Y. Uno. A polynomial-time approximation scheme for the geometric unique coverage problem on unit squares. In *Proc. SWAT*, pages 24–35, 2012.
- [10] N. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete Comput. Geom.*, 44:883–895, 2010.

Convex hull alignment through translation

Michael Hoffmann*

Vincent Kusters*

Günter Rote†

Maria Saumell‡

Rodrigo I. Silveira§

Abstract

Given k finite point sets A_1, \dots, A_k in \mathbb{R}^2 , we are interested in finding one translation for each point set such that the union of the translated point sets is in convex position. We show that if k is part of the input, then it is NP-hard to determine if such translations exist, even when each point set has at most three points.

The original motivation of this problem comes from the question of whether a given triangulation T of a point set is the *empty shape triangulation* with respect to some (strictly convex) shape S . In other words, we want to find a shape S such that the triangles of T are precisely those triangles about which we can circumscribe a homothetic copy of S that does not contain any other vertices of T . This is the Delaunay criterion with respect to S ; for the usual Delaunay triangulation, S is the circle.

1 Introduction

We study the following problem: given k finite point sets A_1, \dots, A_k in \mathbb{R}^2 , are there translations t_1, \dots, t_k such that the union of all $t_i(A_i)$ is in convex position? For $k = 1$ the problem is simply convexity testing. For $k = 2$, the problem can be solved using linear programming, under the additional assumption that the order of points along the convex hull is fixed. Even without this assumption, the case $k = 2$ is solvable in polynomial time: if there is a solution for a given instance, then

*Institute of Theoretical Computer Science, ETH Zürich, Switzerland. [hoffmann|vincent.kusters]@inf.ethz.ch. Partially supported by the ESF EUROCORES programme EuroGIGA, CRP GraDR and the Swiss National Science Foundation, SNF Project 20GG21-134306.

†Institut für Informatik, Freie Universität Berlin, Germany. rote@inf.fu-berlin.de. Supported by the ESF EUROCORES programme EuroGIGA-VORONOI, Deutsche Forschungsgemeinschaft (DFG): grant RO 2338/5-1.

‡Département d’Informatique, Université Libre de Bruxelles, Belgium. maria.saumell.m@gmail.com. Supported by ESF EuroGIGA project ComPoSe as F.R.S.-FNRS - EUROGIGA NR 13604 and ESF EuroGIGA project GraDR as GAČR GIG/11/E023.

§Dept. Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain. rodrigo.silveira@upc.edu. Partially supported by projects MINECO MTM2012-30951, Gen. Cat. DGR2009SGR1040, ESF EUROCORES programme EuroGIGA, CRP ComPoSe: MICINN Project EUI-EURC-2011-4306, and by FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235.

there is also a solution where a point p from the first set is collinear with two points q and r from the second set. For each such triple, it remains only to determine where p should be placed along the line qr , which can easily be done in polynomial time.

For general k (being part of the input), the straightforward formulation does not yield a linear program, because triples of points may come from different polygons and, thus, involve several translations, which makes the constraints quadratic. Similarly, the problem is easy if the size of the point sets is at most two: sort the line segments s_1, \dots, s_n by increasing slope and translate such that the right endpoint of each s_i is identified with the left endpoint of s_{i+1} . In contrast we prove that the general problem is NP-hard by reduction from 3-SAT:

Theorem 1 *Given k finite point sets A_1, \dots, A_k in \mathbb{R}^2 , it is NP-hard to decide if there are translations t_1, \dots, t_k such that the union of all $t_i(A_i)$ is in convex position.*

The reduction uses point sets of size three, as well as a regular polygon with size linear in the size of the 3-SAT formula. We also show that this regular polygon can be replaced by a set of triangles:

Theorem 2 *Given k finite point sets A_1, \dots, A_k in \mathbb{R}^2 , it is NP-hard to decide if there are translations t_1, \dots, t_k such that the union of all $t_i(A_i)$ is in convex position, even if each A_i has size at most three.*

Motivation. The original motivation of this problem comes from the question of whether a given triangulation T of a point set is the *empty shape triangulation* [10, 11] with respect to some (strictly convex) shape S (Problem A). In other words, we want to find a shape S such that the triangles of T are precisely those triangles about which we can circumscribe a homothetic copy of S that does not contain any other vertices of T . This is the Delaunay criterion with respect to S . For the usual Delaunay triangulation, S is the circle.

An abstraction of this question is the following more general problem (Problem B). We are given families $(P_i^+, P_i^=, P_i^-)_{i=1,2,\dots}$ of point sets. We look for a (strictly) convex shape S with the following property: for each i , we can scale and translate S so that the three sets $P_i^+, P_i^=, P_i^-$ lie inside S , on the boundary, and outside S , respectively.

In Problem A, each $P_i^=$ is a triplet corresponding to a triangle, and P_i^- is the complementary set of points. P_i^+

is always empty. Equivalently, we may form quadruples with three triangle points $P_i^=$ and each remaining point as a singleton set P_i^- .

We obtain a variation of this problem (Problem B') by allowing only translation of S , but no scaling. Let Problem C be the special case of Problem B where $P_i^+ = P_i^- = \emptyset$ and let Problem C' be the same special case of Problem B'. The problem that we consider in this paper is problem C'. Thus, our answer to problem C' does not imply an answer to our original question, since it is *more specialized* in one respect (allowing translations only) and *more general* in another respect (considering arbitrary point sets $P_i^=$).

Related work. We are not aware of previous work on the related problems mentioned above. Regarding our main motivation, Problem A, quite some work has been devoted to studying properties of Voronoi diagrams and their duals—often triangulations—based on a particular convex distance function (see e.g. [6, 12]), but not, to the best of our knowledge, to tackling the inverse question in which we are interested here.

The problem of finding a set of translations to place a set of points in convex position is related to certain matching and polygon placement problems. In some matching problems the goal is to find a rigid motion of one shape to make it as *similar* as possible to another shape. Here, similarity can be measured in a variety of ways, such as using Hausdorff distance [2], Fréchet distance [3] or maximizing their area of overlap [7], among others. Polygon placement problems are usually concerned with finding a transformation of a polygon to place it inside another polygon or to contain/avoid certain objects (such as points or other polygons). Multiple variants have been studied in the past, depending on the type of polygon (e.g. convex [14] or simple [4]), the type of transformation (e.g. translation versus translation and rotation [4]), and the final goal (e.g. to fit the largest possible copy of a polygon inside another one [1], or to cover as many points as possible [8]). We remark that, besides having clearly different goals, these problems are usually concerned with two single shapes or objects, whereas we are interested in translating $k > 2$ points sets altogether.

2 Reduction from 3-SAT

This section is devoted to the proof of Theorem 1. We prove the theorem by reduction from 3-SAT. Given a 3-SAT formula F with variables x_0, \dots, x_{n-1} and clauses C_0, \dots, C_{m-1} , construct point sets as follows.

Let $R = r_0, r'_0, \dots, r_{t-1}, r'_{t-1}$ be a regular¹ convex

¹A sufficiently fine rational approximation [5] of R suffices. The main property of R that is important for our reduction is that opposite sides are parallel.

polygon on $2t = 8n + 16m$ points centered at the origin. Given an edge $r_i r'_i$ of R , we define the *pocket* P_i of $r_i r'_i$ to be the compact set bounded by the line segment $r_i r'_i$ and the lines ℓ^- through $r'_{i-1} r_i$ and ℓ^+ through $r'_i r_{i+1}$. The intersection of ℓ^- and ℓ^+ is the *apex* of pocket P_i . Our reduction relies on the following fact: if a point p is placed on an open line segment $r_i r'_i$, then placing a point q in $P_i \setminus r_i r'_i$ destroys convexity of $R \cup \{p, q\}$. We say that p *blocks* the pocket P_i . Unblocked pockets are called *free*. Note that placing a point in P_i does not prevent us from placing a point in any other pocket. We say that two convex sets are *compatible* if their union is convex.

Variable gadget. In order to encode the variables of our formula, we will first define triangles $Q_i^d = (q_i, q_{i+d}, q_{i+t/2})$ for $0 \leq i < t$ and $0 \leq d < t$ as follows (note that in this section all indices are taken modulo t , and that t is even). Consider the diametrically opposed segments $r_i r'_i$ and $r_{i+t/2} r'_{i+t/2}$. Place $q_{i+t/2}$ on $r_{i+t/2}$ and place q_{i+d} on the apex of P_{i+d} (Figure 1). Translate $q_{i+t/2}$ and q_{i+d} rigidly, parallel along $r_{i+t/2} r'_{i+t/2}$ until q_{i+d} hits the line segment $r_{i+d} r'_{i+d}$. Now place q_i on r_i . Note that Q_i^d can slide freely along $r_i r'_i$ until:

- (1) q_i hits r_i and q_{i+d} hits the line segment $r_{i+d} r'_{i+d}$: moving further would move q_{i+d} inside the convex hull of R ; or
- (2) $q_{i+t/2}$ hits $r_{i+t/2}$ and q_{i+d} hits the apex of P_{i+d} : moving further would push q_{i+d} outside the pocket, removing r_{i+d} and r'_{i+d} from the joint convex hull.

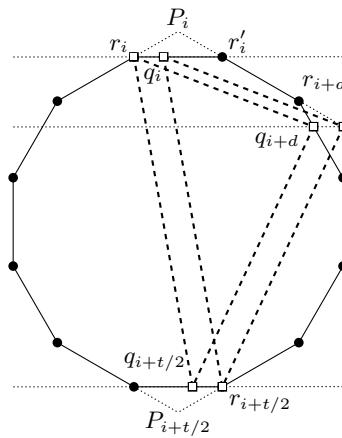


Figure 1: The construction of variable gadget Q_i^d . The figure shows the two extreme positions of the gadget.

Since $r_i r'_i$ and $r_{i+t/2} r'_{i+t/2}$ are diametrically opposed segments, moving Q_i^d in any other direction would place either q_i or $q_{i+t/2}$ inside the convex hull of R . For most of the compatible translations, the pockets P_i and

$P_{i+t/2}$ are both blocked. However, for the two extreme positions (shown in Figure 1), where q_{i+d} touches the apex or the line segment, exactly one of the pockets is free.

We say that the *state* of Q_i^d is *true* if Q_i^d does not block P_i , *false* if it does not block $P_{i+t/2}$ and *undefined* if P_i and $P_{i+t/2}$ are both blocked. After defining the other two gadgets, we will associate some Q_j^d with each variable x_i , with the interpretation that x_i is true if and only if the state of Q_j^d is true and \bar{x}_i is true if and only if the state of Q_j^d is false. If the state of Q_j^d is undefined, then both x_i and \bar{x}_i are false. Note that if R and a translation of Q_i^d are compatible, then x_i and \bar{x}_i are not both true.

Copy gadget. Given a variable gadget Q_i^d , we can copy the state of Q_i^d with the gadget Q_{i+k}^{d-k} , provided $k \notin \{0, t/2, d\}$. This can be seen as follows. Assume for the moment that Q_i^d is in one of its two extreme positions. The gadget Q_{i+k}^{d-k} shares the pocket P_{i+d} with Q_i^d (see Figure 2). If Q_i^d touches the apex of P_{i+d} (as in Figure 2), then so must Q_{i+k}^{d-k} in order to be compatible with R and the translation of Q_i^d . Similarly, if Q_i^d touches the open line segment $r_{i+d}r'_{i+d}$, then so must Q_{i+k}^{d-k} . Hence, the state of Q_{i+k}^{d-k} is completely determined by the state of Q_i^d . Specifically, the state of P_i is copied to P_{i+k} and the state of $P_{i+t/2}$ is copied to $P_{i+k+t/2}$. If Q_i^d is not in an extreme position, then neither is Q_{i+k}^{d-k} and hence both states are undefined.

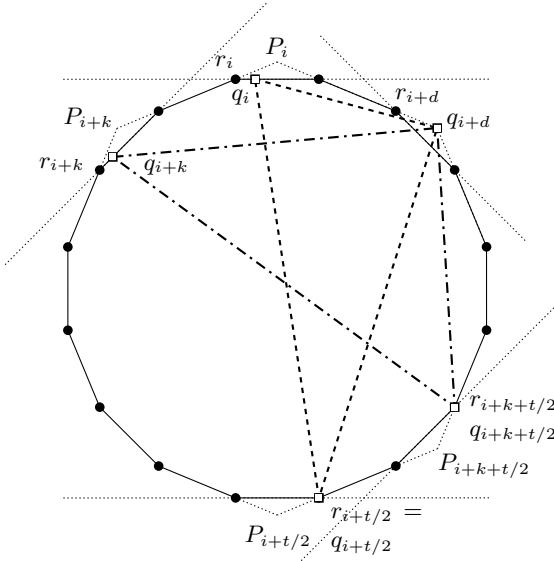


Figure 2: The construction of copy gadget Q_{i+k}^{d-k} with $k = -1$.

Clause gadget. Given i, j and k , let T_{ijk} be the triangle whose vertices t_i, t_j, t_k are the midpoints of the seg-

ments $r_ir'_i, r_jr'_j$ and $r_kr'_k$, respectively. For each clause C on variables x, y, z , we will use three copy gadgets to copy the states of the literals x (or \bar{x}), y (or \bar{y}) and z (or \bar{z}) of C to pockets P_i, P_j, P_k such that T_{ijk} contains the origin. Some care must be taken to ensure that this clause does not interfere with existing gadgets, but we will cover this issue in the subsection below. Figure 3 shows an example for a clause $C = x \vee \bar{y} \vee z$. Only the relevant corner of each copy gadget is shown.

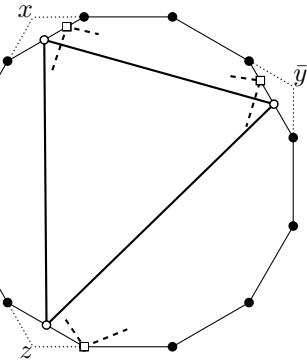


Figure 3: The construction of a clause gadget for a clause $C = x \vee \bar{y} \vee z$.

Note that x and \bar{y} are both in the false (or undefined) state, whereas z is in the true state. We now scale T_{ijk} by a factor of $1 + \varepsilon$ for ε sufficiently small. Since triangle T_{ijk} contains the origin, it no longer fits inside R . If it did not contain the origin, a small translation towards the origin would potentially bring it back in convex position with R . If one of the pockets P_i, P_j or P_k is free, e.g. P_i , then we can translate T_{ijk} such that t_j is again on $r_jr'_j$ and t_k is again on $r_kr'_k$ and t_i is inside P_i . This translation of T_{ijk} is compatible with R : hence, the clause C is satisfied. If all three pockets are blocked, then there is no translation for which T_{ijk} is compatible with R and hence C is not satisfied.

Selecting the gadgets. We are now ready to explicitly define the gadgets to be used for the given 3-SAT formula F with variables x_0, \dots, x_{n-1} and clauses C_0, \dots, C_{m-1} . We partition the polygon into paths $A, B'_1, B_2, Q, B'_3, X, A', B_1, B'_2, Q', B_3, X'$ as shown in Figure 4. The paths A, Q, A' and Q' all have length $2n$. The other paths have length $2m$. Hence R has size $2t = 8n + 16m$. We will not use the paths X, Q' and X' . Associate with each variable x_i the variable gadget Q_i^{2m+n} . This places the variable gadgets in paths A, Q and A' . For each clause $C_i = \{\ell_a, \ell_b, \ell_c\}$ add copy gadgets as follows. If $\ell_a = \bar{x}_a$ (i.e. the literal is negative), then add Q_{i+n}^{a+2m-i} to copy the value of x_a to the i th pocket in B'_1 . This copies the value of \bar{x}_a into the i th pocket in B_1 . Alternatively, if $\ell_a = x_a$ (i.e. the literal is positive), then add $Q_{i+3n+4m}^{a-2n-2m-i}$ to copy the value of x_a to the i th pocket in B_1 . Copy ℓ_b and ℓ_c analogously to

B_2 and B_3 and add the clause gadget corresponding to the selected pocket. Note that any triangle of vertices selected from B_1, B_2, B_3 contains the origin.

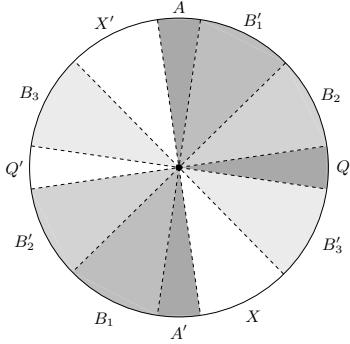


Figure 4: Placing the gadgets on the polygon.

Correctness. If our 3-SAT formula F is satisfiable, then set the state of the initial variable gadgets according to a satisfying assignment of F . The copy gadgets preserve the state of these initial variable gadgets, so each clause gadget has at least one free pocket. Hence, the union of R with all gadgets is in convex position. Conversely, if F is unsatisfiable, then suppose for the sake of obtaining a contradiction that the union of R with all gadgets is in convex position. Consider the assignment α of F corresponding to the state of the variable gadgets (this may set some x_i and \bar{x}_i both to false). Since F is unsatisfiable, there must be a clause C that is not satisfied by assignment α . Since the copy gadgets preserve the state of the original variable gadgets, all pockets of C are blocked. Hence, C is not compatible with the other gadgets and the union of R with all gadgets cannot be in convex position, which yields a contradiction. Theorem 1 follows.

3 Replacing a regular polygon by triangles

In this section we show that we can replace the regular polygon from our reduction above by a set of triangles. This will prove Theorem 2.

Proposition 3 *Let p_1, p_2, \dots, p_n be the vertices of a regular n -gon in counterclockwise order. Let us consider the family of triangles*

$$\begin{aligned} \mathcal{T} = & \{\triangle p_1 p_2 p_3, \triangle p_2 p_3 p_4, \dots, \triangle p_n p_1 p_2, \\ & \triangle p_1 p_2 p_{n-2}, \triangle p_2 p_3 p_{n-1}, \dots, \triangle p_n p_1 p_{n-3}\}. \end{aligned}$$

Let S be a point set obtained by translating each triangle in \mathcal{T} independently. If S is in convex position, then S consists of the vertices of a regular n -gon.

The rest of this section is devoted to the proof of Proposition 3. For a given set S of points in the plane, let $CH(S)$ denote the vertices of the convex hull of S .

Let $T \subseteq \mathcal{T}$. Suppose that every triangle $T_i \in T$ is translated according to some vector λ_i , and let S_{T_λ} be the resulting set of points. We call S_{T_λ} a *geometric placement* of T . A placement of T is called *combinatorial* if the order of the vertices of T around the convex hull is known, but not the exact position of the triangles. We say that S_{T_λ} satisfies the *convex condition* if all points of S_{T_λ} belong to $CH(S_{T_\lambda})$. In this case we might also say that the placement of the triangles is *valid*. We say that $p \in S_{T_\lambda}$ violates the convex condition if $p \notin CH(S_{T_\lambda})$.

Every vertex of the regular n -gon belongs to several triangles of \mathcal{T} . To distinguish the distinct copies of the vertex, we use superscripts, so for example p_1^i will denote vertex p_1 in some particular triangle of \mathcal{T} . We set $T_1 = \triangle p_1^1 p_2^1 p_3^1$, $T_2 = \triangle p_2^2 p_3^2 p_4^2, \dots$, and $T_n = \triangle p_n^n p_1^n p_n^n$. We also set $T_{n+1} = \triangle p_1^{n+1} p_2^{n+1} p_{n-2}^{n+1}$, $T_{n+2} = \triangle p_2^{n+2} p_3^{n+2} p_{n-1}^{n+2}, \dots$, and $T_{2n} = \triangle p_n^{2n} p_1^{2n} p_{n-3}^{2n}$. We assume that n is a multiple of 4, that the regular n -gon is oriented so that two of its sides are horizontal, and that $p_2 p_3$ is the bottom horizontal side.

Observation 1 *Let $p \in S \subseteq S'$. If $p \notin CH(S)$, then $p \notin CH(S')$.*

Lemma 4 *Let $T = \{T_1, T_2\}$. Suppose that S_{T_λ} satisfies the convex condition, and $p_2^1 p_3^1$ is not collinear with $p_2^2 p_3^2$. Then the counterclockwise order of S_{T_λ} in $CH(S_{T_\lambda})$ is either $p_1^1 p_2^1 p_2^2 p_3^2 p_4^1$ or $p_1^1 p_2^1 p_3^1 p_3^2 p_4^2 p_2^2$.*

Proof. Suppose that we fix the position of T_1 . Then if we extend the line segments of T_1 to lines, the plane is decomposed into four open regions F_1, \dots, F_4 and three closed regions R_1, \dots, R_3 , as shown in Figure 5. It is easy to verify that in order to satisfy the convex condition, no point of T_2 can lie in any of the regions F_1, F_2, F_3 , and no vertex of T_2 can lie in F_4 . In particular, p_2^2, p_3^2 , and p_4^2 can lie only in $R_1 \cup R_2 \cup R_3$. In principle, there are 27 cases to consider, based on all possible combinations. Fortunately, we only need to distinguish a few situations.

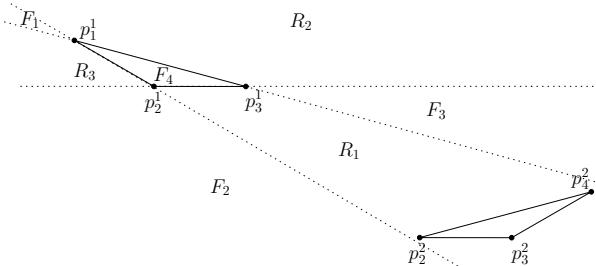


Figure 5: A valid combinatorial placement of T_1 and T_2 .

We first suppose that $p_2^2 \in R_1$. Notice that in this case it is not possible that $p_3^2 \in R_2$ or $p_3^2 \in R_3$. So we can assume that $p_3^2 \in R_1$. Now we can easily rule

out the case $p_4^2 \in R_3$. Point p_4^2 cannot lie in R_2 either, because in this case the side $p_3^2 p_4^2$ would intersect F_3 . Therefore p_4^2 can lie only in R_1 . This gives a valid combinatorial placement of T_1 and T_2 where the counterclockwise order of S_{T_λ} in $CH(S_{T_\lambda})$ is $p_1^1 p_2^1 p_2^2 p_3^2 p_4^2 p_1^1$ (see Figure 5).

Next suppose that $p_2^2 \in R_2$. Then, $p_3^2 \in R_2$ and $p_4^2 \in R_2$. This gives another valid combinatorial placement of T_1 and T_2 where the counterclockwise order of S_{T_λ} in $CH(S_{T_\lambda})$ is $p_1^1 p_2^1 p_3^1 p_3^2 p_4^2 p_2^2$ (see Figure 6). Note that there are no other possible combinatorial placements since we assume that $p_2^1 p_3^1$ and $p_2^2 p_3^2$ are not collinear.

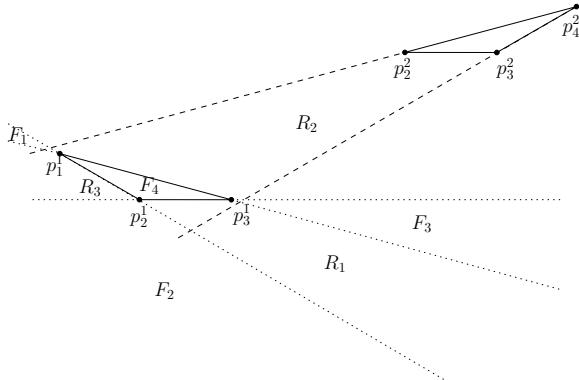


Figure 6: A valid combinatorial placement of T_1 and T_2 .

Finally suppose that $p_2^2 \in R_3$. Then, $p_3^2 \notin R_1$ and $p_4^2 \notin R_1$. Recall that p_2^2 is on the same horizontal line as p_3^2 and to its left, and p_4^2 is above and to the right of p_3^2 . If $p_3^2 \in R_2$ and $p_4^2 \in R_2$, then we have that p_3^1 is below and to the right of p_3^2 . Consequently, p_3^2 lies inside the triangle $\Delta p_2^2 p_4^2 p_3^1$ and violates the convex condition. The combination $p_3^2 \in R_2$ and $p_4^2 \in R_3$ is not possible. If $p_3^2 \in R_3$, then we have that p_2^1 is below and to the right of p_3^2 (since we are assuming that $p_2^1 p_3^1$ is not collinear with $p_2^2 p_3^2$). Consequently, p_3^2 lies inside the triangle $\Delta p_2^2 p_4^2 p_2^1$ and violates the convex condition. \square

Lemma 5 Let $T = \{T_1, T_2, T_{\frac{n}{2}+1}, T_{\frac{n}{2}+2}\}$. If S_{T_λ} satisfies the convex condition, then $p_2^1 p_3^1$ is collinear with $p_2^2 p_3^2$.

Proof. [Sketch] Due to space limitations, we only sketch the proof here. It suffices to prove that the combinatorial placements for $\Delta p_1^1 p_2^1 p_3^1$ and $\Delta p_2^2 p_3^2 p_4^2$ of Lemma 4 are no longer valid. To show that the combinatorial placement in which the counterclockwise order around the convex hull is $p_1^1 p_2^1 p_2^2 p_3^2 p_4^2 p_1^1$ is no longer valid, we try to add $T_{\frac{n}{2}+1}$ to the placement, and prove that this is not possible. The other combinatorial placement from Lemma 4 can be ruled out analogously. \square

By symmetry, we have the following corollary:

Corollary 6 If S_{T_λ} satisfies the convex condition, then $p_2^1 p_3^1$ is collinear with $p_2^2 p_3^2$, $p_3^2 p_4^2$ is collinear with $p_3^3 p_4^3, \dots$, and $p_1^n p_2^n$ is collinear with $p_1^1 p_2^1$.

Lemma 7 If S_{T_λ} satisfies the convex condition, then $p_2^1 p_3^1$ is collinear with $p_2^2 p_3^2$, and p_2^1 is not right of p_2^2 .

We omit the proof due to space considerations.

Suppose that, in some placement of T_{i-1} and T_i , $p_i^{i-1} p_{i+1}^{i-1}$ is collinear with $p_i^i p_{i+1}^i$. Suppose that we rotate the triangles so that T_{i-1} has the same orientation as T_1 and T_i has the same orientation as T_2 . We say that T_{i-1} and T_i cross if, after this rotation, p_i^{i-1} is to the right of p_i^i . By symmetry, we have the following corollary, which subsumes Corollary 6 and Lemma 7:

Corollary 8 Suppose that S_{T_λ} satisfies the convex condition. Then $p_2^1 p_3^1$ is collinear with $p_2^2 p_3^2$, $p_3^2 p_4^2$ is collinear with $p_3^3 p_4^3, \dots$, and $p_1^n p_2^n$ is collinear with $p_1^1 p_2^1$. Furthermore, none of the pairs $\{T_i, T_{i+1}\}$ cross.

Lemma 9 Suppose that S_{T_λ} satisfies the convex condition. Then $p_2^1 p_3^1$ is collinear with $p_2^2 p_3^2$, and p_2^1 is not to the left of p_2^2 .

Proof. We proceed by contradiction. Consider the regular n -gon having one side at $p_2^2 p_3^2$. Since none of the pairs $\{T_i, T_{i+1}\}$ cross, this polygon lies inside the polygon formed by the placement of T_1, \dots, T_n (see Figure 7 for an example). Since we are assuming that p_2^1 is to the left of p_2^2 , we also have that p_n of the polygon lies strictly inside (that is, not on the boundary) the polygon formed by the placement of T_1, \dots, T_n . We now try to place T_{n+3} . In order to maintain the convex condition, the vertices of T_{n+3} must be placed on the boundary of the polygon formed by the placement of T_1, \dots, T_n , or on the triangles bounded by an edge of the polygon and two dotted segments shown in Figure 7. Such a placement of T_{n+3} is not possible. \square

By symmetry, we obtain the following corollary, which subsumes Lemma 9 and Corollary 8:

Corollary 10 Suppose that S_{T_λ} satisfies the convex condition. Then p_2^1 is on the same position as p_2^2 , p_3^2 is on the same position as p_3^3, \dots , and p_1^n is on the same position as p_1^1 . Equivalently, the vertices of $\Delta p_1^1 p_2^1 p_3^1, \Delta p_2^2 p_3^2 p_4^2, \dots, \Delta p_n^n p_1^n p_2^n$ form a regular n -gon.

To complete the proof of Proposition 3, it only remains to see that the triangles $T_{n+1}, T_{n+2}, \dots, T_{2n}$ are also placed in the “natural” way. We prove it in the next lemma for T_{n+2} and, by symmetry, the result also holds for the other triangles.

Lemma 11 Suppose that S_{T_λ} satisfies the convex condition. Then, $\Delta p_2^{n+2} p_3^{n+2} p_{n-1}^{n+2}$ is placed so that p_2^{n+2} is on the same position as p_2^2 .

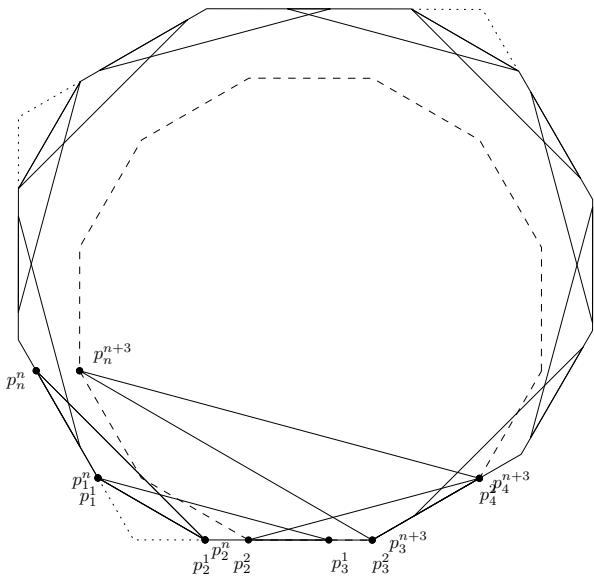


Figure 7: The vertex p_2^{n+3} cannot be placed in Lemma 9 if p_2^1 is to the left of p_2^2 (case 1).

Proof. We know that the vertices of T_1, T_2, \dots, T_n form a regular n -gon. In order to maintain the convex condition, the vertices of T_{n+2} must be placed on the boundary of the regular n -gon or on the triangles bounded by an edge of the polygon and two dotted segments shown in Figure 8. It is clear that the only way to do it consists in placing p_2^{n+2} on the same position as p_2^2 . \square

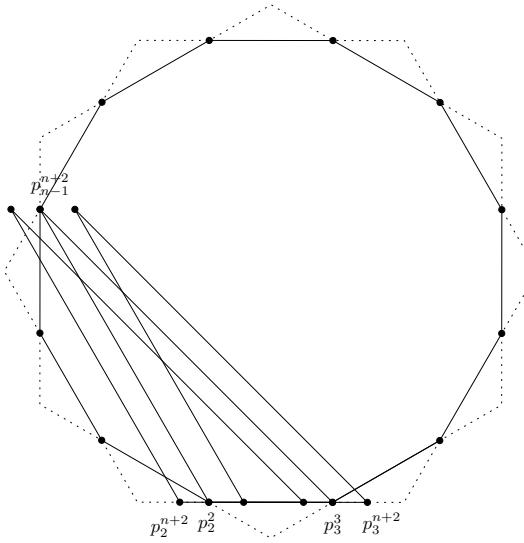


Figure 8: Points with the same superscript must again coincide after the translations.

4 Concluding remarks

If we allow scaling in addition to translation (Problem C from the introduction), then it is not known if the prob-

lem is still NP-hard. In addition, it is not clear if our problem is in NP. The obvious certificate would be the sequence of translations, but one must argue that the representation of these translations is not too large in terms of the input representation. In fact, our problem has some similarities to the order type realizability problem, which is known to be complete for the existential theory of the reals [13], and the coordinate representation of some order types requires exponential storage [9].

Acknowledgments. This research was initiated during a EuroGIGA workshop in Assisi (Italy) in March 2012. We thank Jan Kratochvíl, Tillmann Miltzow, Alexander Pilz, and Vera Sacristán for helpful discussions.

References

- [1] P. K. Agarwal, N. Amenta, and M. Sharir. Largest placement of one convex polygon inside another. *Discrete Comput. Geom.*, 19(1):95–104, 1998.
- [2] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell.*, 13(3-4):251–265, 1995.
- [3] H. Alt and M. Godau. Measuring the resemblance of polygonal curves. Proc. SCG '92, pp. 102–109, 1992.
- [4] F. Avnaim and J.-D. Boissonnat. Polygon placement under translation and rotation. *ITA*, 23(1):5–28, 1989.
- [5] J. Canny, B. R. Donald, and E. K. Ressler. A rational rotation method for robust geometric algorithms. Proc. SCG '92, pp. 251–260, 1992.
- [6] L. P. Chew and R. L. S. Drysdale, III. Voronoi diagrams based on convex distance functions. Proc. SCG '85, pp. 235–244, 1985.
- [7] M. de Berg, O. Cheong, O. Devillers, M. van Kreveld, and M. Teillaud. Computing the maximum overlap of two convex polygons under translations. *Theory Comput. Syst.*, 31(5):613–628, 1998.
- [8] M. Dickerson and D. Scharstein. Optimal placement of convex polygons to maximize point containment. *Comput. Geom.*, 11(1):1–16, 1998.
- [9] J. E. Goodman, R. Pollack, and B. Sturmfels. Coordinate representation of order types requires exponential storage. Proc. STOC '89, pp. 405–410, 1989.
- [10] T. Lambert. Systematic local flip rules are generalized delaunay rules. Proc. CCCG '93, pp. 352–357, 1993.
- [11] T. Lambert. *Empty-Shape Triangulation Algorithms*. PhD thesis, University of Manitoba, 1994.
- [12] L. Ma. Bisectors and Voronoi diagrams for convex distance functions. Master's thesis, FernUniversität Hagen, 2000.
- [13] M. Schaefer. Complexity of some geometric and topological problems. In *Graph Drawing*, volume 5849 of *LNCS*, pp. 334–344. Springer, 2010.
- [14] M. Sharir and S. Toledo. External polygon containment problems. *Comput. Geom.*, 4:99–118, 1994.

Faster approximation for Symmetric Min-Power Broadcast

G. Calinescu *

Abstract

Given a directed simple graph $G = (V, E)$ and a cost function $c : E \rightarrow R_+$, the *power* of a vertex u in a directed spanning subgraph H is given by $p_H(u) = \max_{uv \in E(H)} c(uv)$, and corresponds to the energy consumption required for the wireless node u to transmit to all nodes v with $uv \in E(H)$. The *power* of H is given by $p(H) = \sum_{u \in V} p_H(u)$.

Power Assignment seeks to minimize $p(H)$ while H satisfies some connectivity constraint. In this paper, we assume E is bidirected (for every directed edge $e \in E$, the opposite edge exists and has the same cost), a “source” $y \in V$ is also given as part of the input, and H is required to contain a directed path from y to every vertex of V . This is the NP-Hard Symmetric Min-Power Broadcast problem.

In terms of approximation, it is known that one cannot obtain a ratio better than $\ln |V|$, and at least five algorithms with approximation ratio $O(\ln |V|)$ have been published from 2002 to 2007. Here we take one of them, the $2(1 + \ln |V|)$ -approximation of Fredrick Mtenzi and Yingyu Wan, and improve its running time from $O(|V||E|)$ to $O(|E| \log^2 |V|)$, by careful bookkeeping and by using a previously-known geometry-based data structure.

1 Introduction

We study the problem of assigning transmission power to the nodes of ad hoc wireless networks to minimize power consumption while ensuring that the given source reaches all the nodes in the network (unidirectional links allowed for broadcast), in the symmetric cost model. This problem takes as input a directed simple graph $G = (V, E)$ and a cost function $c : E \rightarrow R_+$. The *power* of a vertex u in a directed spanning simple subgraph H of G is given by $p_H(u) = \max_{uv \in E(H)} c(uv)$, and corresponds to the energy consumption required for the wireless node u to transmit to all nodes v with $uv \in E(H)$. The *power* (or *total power*) of H is given by $p(H) = \sum_{u \in V} p_H(u)$. A “source” (called “root” sometimes in the literature) $y \in V$ is also given as part of the input, and H is required to contain a directed path

from y to every vertex of V ; we call the problem of minimizing the total power while ensuring this connectivity Symmetric Min-Power Broadcast. Among early work on this problem we mention [22, 24, 11, 23].

This problem is motivated by minimizing energy consumption in a static multi-hop wireless network, where $c(u, v)$ represents the transmission power wireless node u must spend to ensure a packet is received by node v . Our model is that wireless nodes have several levels of transmission power. A packet sent by u with power p is received by all nodes v with $c(u, v) \leq p$. This feature is useful for energy-efficient multicast and broadcast communications.

In some wireless settings, it is reasonable to assume that u and v are embedded in the two-dimensional Euclidean plane, and $c(u, v)$ is proportional to the distance from the position of u to the position of v , raised to a power κ , where κ is fixed constant between 2 and 5. This is the Euclidean input model.

We do not work in the Euclidean input model, but make a (less-restrictive) “symmetric” assumption that E is bidirected, (that is, $uv \in E$ if and only if $vu \in E$, and the two edges have the same cost).

A survey of Power Assignment problems is given by Santi [20]; like there we only consider centralized algorithms (there is a vast literature on distributed algorithms). Even in the Euclidean input model, Min-Power Broadcast was proven NP-Hard [11], and it was a folklore result in 2000 that Symmetric Min-Power Broadcast is as hard to approximate as Set Cover (this appears in several papers [11, 17, 23, 7, 2, 16]). Based on Feige’s hardness result for Set Cover [12], no approximation ratio better than $O(\ln n)$ is possible unless $P = NP$. Here $n = |V|$, and from now on $m = |E|$.

The first $O(\log n)$ approximation algorithm was given by Caragiannis et al. [7] (journal version: [8]). A similar algorithm was presented in [3], and the simplest and best variant (ratio of $2(1 + \ln n)$) of this algorithm was presented by [18] and achieves a $O(mn)$ running time (their analysis claims $O(mna(mn))$ running time, but one observation can get rid of the inverse Ackermann function α in the analysis). Later, [9] obtains another $O(\log n)$ approximation algorithm, with a complicated algorithms based on [14], that needs multiple calls to Minimum Weight Perfect Matching. Two other algorithms also achieve a $2(1 + \ln n)$ -approximation ratio: the Spider algorithm of [4] (which has the same ratio if the input graph is not bidirected) and the Relative

*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA. calinescu@iit.edu. Research supported in part by NSF grant NeTS-0916743.

Greedy algorithm of [6] (which also achieves the best known approximation ratio, of 4.2, in the Euclidean input model).

All these algorithms use greedy methods, mostly adopted from the Steiner Tree problems and its variants (precisely, [15, 26, 14]). Most of these papers do not explicitly analyze the running time of the presented algorithms (and none, as given, is faster than $O(mn)$). We set to achieve the same $2(1 + \ln n)$ -approximation ratio with an improved running time.

For this, we give a faster variant of the “Hypergraph-Greedy” algorithm of Mtenzi and Wan [18]. It turns out this algorithm is a special case of the greedy method for Polymatroid Cover of [25] (a simpler analysis in [13]), and we use this observation to give an alternative proof of its approximation ratio. For some readers, the direct proof of [18] may be more enlightening; we just point out in this paper that the proof is a special case of the [13] proof. We achieve a running time of $O(m \log^2 n)$ by careful book-keeping and by using a data structure of [5].

The next section presents the Hypergraph-Greedy algorithm of [18] with our notation, and gives an alternative, shorter proof of its approximation ratio, based on [25] and [7]. Then, in Section 3, we describe how to use a known data structure. Section 4 combines several data structures with careful book-keeping and analysis to obtain the improved running time.

2 Algorithm Description and Approximation Ratio

Given a directed edge uv , its *undirected version* is the undirected edge with endpoints u and v ; for a set of directed edges F , we denote by \widehat{F} the multiset of edges that are the undirected version of the edges of F .

For $u \in V$ and $r \in \{c(uv) \mid uv \in E\}$, let $S(u, r)$ be the *directed star* (or, simply, *star*) consisting of all the arcs uv with $c(uv) \leq r$. We call u the center of S and note that r is the power of $S(u, r)$. For a directed star S , let $p(S)$ denote its power, let $E(S)$ be its set of arcs, and define $V(S)$, its set of vertices, to be its center plus the heads of its arcs. See Figure 1 for an example. The algorithm treats $V(S)$ as a hyperedge in a hypergraph with vertex set V .

The algorithms described use all the possible stars, and there are $O(m)$ of them (for each vertex, the number of stars is its degree in the input graph). In the first phase, the algorithm keeps a set of stars (initially empty), giving a set of arcs H . It then selects the next star such that to maximize the decrease in the number of weakly connected components in (V, H) divided by the power of the star (see Figure 2). The first phase stops when (V, H) is weakly connected. At this moment, the second phase of the algorithm constructs a spanning tree in the undirected version of (V, H) (for example,

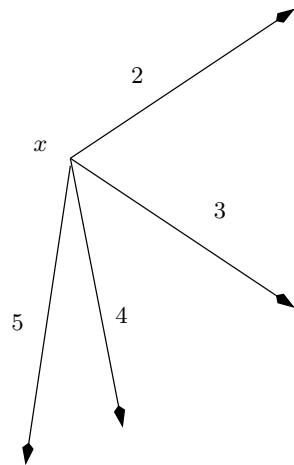


Figure 1: A star with center x and four arcs, of power $\max\{2, 3, 4, 5\} = 5$.

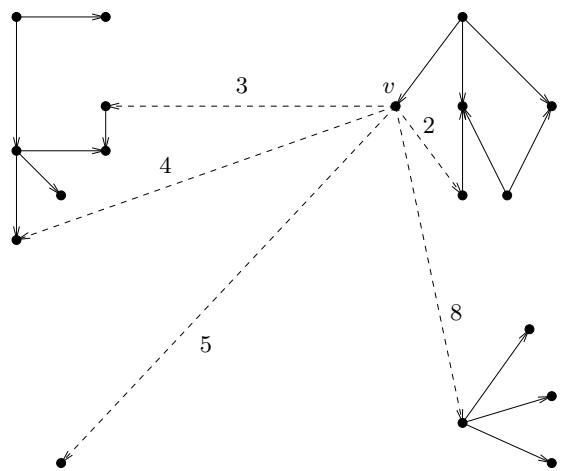


Figure 2: The current H is drawn as solid segments, with arrows indicating the direction of the edges. The directed edges with tail v are drawn as dashed segments. The star with center v and power 2 has one edge, and it does not decrease the number of weakly connected components of H . The star with center v and power 3, with two edges, decreases the number of weakly connected components of H by 1. The star $S(v, 4)$ has three edges and also achieves a reduction of 1. $S(v, 5)$ achieves a reduction of 2, and $S(v, 8)$ achieves a reduction of 3. Among the stars with center v , the algorithm would choose $S(v, 5)$ as the next star.

by breadth-first search or depth-first search), and it re-orientates if needed the edges of this tree to lead away from y (the vertex given in the input as the source), and thus obtains a feasible output. This re-orientation, first applied in [7], only works if the input graph is bidirected.

2.1 Approximation Ratio

We first cast the problem in a different setting. A polymatroid $f : 2^N \rightarrow \mathbb{Z}^+$ on a ground set N is a non-decreasing (monotone) integer-valued submodular function. A function f is monotone iff $f(A) \leq f(B)$ for all $A \subseteq B$. A function f is submodular iff $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq N$. Polymatroids generalize matroids which have the additional condition that $f(\{i\}) \leq 1$ for all $i \in N$. We call a subset $A \subseteq N$ spanning iff $f(A) = f(N)$.

Assume each element $j \in N$ has a weight w_j . Define $f_A(B) = f(A \cup B) - f(A)$ and $t = \max_{i \in N} f(\{i\})$. The greedy algorithm of Wolsey [25] find a H_t -approximation to the minimum weight polymatroid spanning set, where H_t is the t^{th} harmonic number, $\sum_{i=1}^t (1/i)$, which is known to be at most $1 + \ln t$. This algorithm, a generalization of Chvatal's algorithm [10] for Set Cover, starts with $B = \emptyset$, and as long as $f(B) < f(N)$, adds to B the element $j \in N$ that maximizes $f_B(\{j\})/w_j$.

In our setting, define N to be the set of all stars, and for a set B of stars, define $f(B)$ to be the size of a maximal forest in $\bigcup_{S \in B} \widehat{E(S)}$. We do have a polymatroid: as explained in [21], Example 44.1(a), the rank function of a matroid (in our case, the graphic matroid, where a set of edges of an undirected graph is independent iff it is a forest) produces a polymatroid. Note that $f(N) = n - 1$ and a spanning set in the polymatroid corresponds to a set of stars whose arcs form a spanning weakly connected subgraph of G . Note also that, if for a set of stars B , we let $co(B)$ be the number of connected components of $(V, \bigcup_{S \in B} \widehat{E(S)})$, we have $f(B) = |V| - co(B)$. Then $f_B(S) = f(\{S\} \cup B) - f(B) = |V| - co(B \cup \{S\}) - (|V| - co(B)) = co(B) - co(B \cup \{S\})$, which is the decrease in the number of weakly connected components in (V, H) when H , given by $\bigcup_{S' \in B} E(S')$, is replaced by $\bigcup_{S' \in B} E(S') \cup E(S)$. With the weight of a star defined to be its power, the algorithm of [18] is the greedy algorithm for polymatroids.

Let OPT be an optimum solution of the instance at hand. Without increasing total power or decreasing connectivity, add, if needed, to OPT every arc vu with $c(vu) \leq p_{OPT}(v)$. For each $v \in V$, call the star $S = S(v, p_{OPT}(v))$ a star of OPT . Since OPT contains a path from the source y to every other vertex of G , we have that the stars of OPT form a spanning set in the polymatroid above. Thus, using [25], the collection of stars \mathcal{A} selected by the first phase of the algorithm satisfies:

$$\begin{aligned} \sum_{S \in \mathcal{A}} p(S) &\leq (1 + \ln n) \sum_{\substack{S \text{ star of } OPT}} p(S) \\ &\leq (1 + \ln n) opt, \end{aligned} \quad (1)$$

where $opt = p(OPT)$.

Let H be obtained by keeping an arbitrary subtree of $\bigcup_{S' \in \mathcal{A}} \widehat{E(S')}$ and orienting the edges away from the source y . For vertex $u \in V$, we denote by u' its parent in this outgoing arborescence. Also, we denote by $\tilde{c}(S)$ the center of star S . Now, using the argument of [8], we have:

$$\begin{aligned} p(H) &= \sum_{u \in V} p_H(u) \\ &\leq \sum_{u \in V} \left(\sum_{S \in \mathcal{A} | u = \tilde{c}(S)} p(S) + \sum_{S \in \mathcal{A} | u = \tilde{c}(S)'} p(S) \right) \\ &\leq 2 \sum_{S \in \mathcal{A}} p(S), \end{aligned}$$

where we use that a star $S \in \mathcal{A}$ appears at most twice in the middle summation: once for the center of S , and once for the parent in H of the center of S . Combined with Inequality 1, we obtain the desired approximation ratio.

3 The data structure used

A *Rel-Max* data structure stores a list of items i , sorted by their *cost* c_i (non-decreasing). A query is finding the j maximizing j/c_j . The update consists of, given i , remove the i^{th} item from the list (this changes the position of the items k , for $k > i$). Calinescu and Qiao [5] present an implementation for a generalization of *Rel-Max* queries/updates. In their data structure, each item also has a “coverage” f_i , non-decreasing in i , and one must find the j maximizing f_j/c_j , while the update consist of re-setting, for given i and $\delta > 0$, for all $k \geq i$, $f_k = f_k - \delta$. Their approach is based on keeping upper convex hulls. See Figure 3 for some intuition.

It seems they re-invented some ideas from [19], also concerned with keeping convex hulls, under different update operations; [1] being a more recent work on this topic. [5] obtains an initialization/preprocessing time of $O(l \log^2 l)$, a query time of $O(\log l)$, and an update time of $O(\log^2 l)$, where l is the number of items in the initial list.

4 Book-keeping

One needs to find the next star at most n times, and the main challenge is to compute the star that maximizes the decrease in the number of weakly connected components in (V, H) divided by the power of the star. The method of [18] is to try all $O(m)$ stars, and with careful bookkeeping one gets a $O(mn)$ -time algorithm. Our goal is $O(m \log^2 n)$.

For every u , let $v_1^u, \dots, v_{d(u)}^u$ be the neighbors of u in G , sorted in non-decreasing order by $c(uv_i^u)$. Let $S_j(u)$ be the star with center u and power $c(uv_j^u)$.

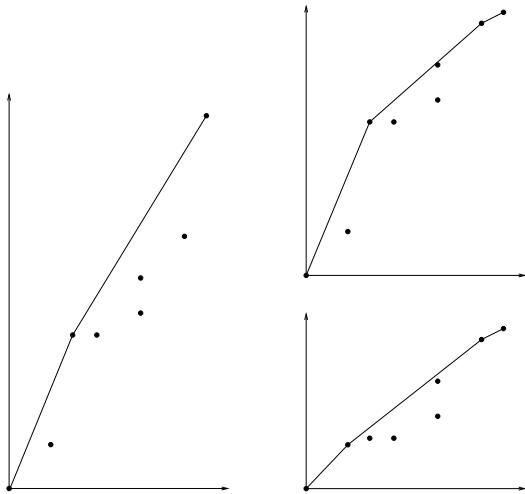


Figure 3: Points P_i have coordinates (c_i, f_i) . On the left, an example of points P_i , with the upper convex hull drawn. Add for convenience P_0 with coordinates $(0, 0)$. The answer to the query is the neighbor of P_0 on the upper hull. On the right, top, the points P_i after f_7 is updated (decreased), with the upper convex hull drawn. On the right, bottom, the points P_i after an update with $i = 2$, causing the second coordinate to drop for points $P_2 \dots P_7$. The upper convex hull is also drawn.

We keep the following three data structures. Let Q_1, \dots, Q_l be the vertex sets of the current weakly connected components of (V, H) . We keep the components by having an explicit representative vertex in each, that is, an array $\text{comp}[v]$ stores the representative of the component containing vertex v . We keep l binary search trees B_i (for $i \in \{1, 2, \dots, l\}$), one for each component. The tree B_i keeps, sorted by ID, the nodes of Q_i together with the nodes u such that an edge $uv \in E$ exists with $v \in Q_i$; in this case we also store the smallest j such that $v_j^u \in Q_i$. For each component with vertex set Q_i , we explicitly keep $|Q_i|$ and a linked list of its vertices.

For each u , we keep a list L_u of items j , each corresponding to the edge uv_j^u and of value $c_j = c(uv_j^u)$, sorted in non-decreasing order by c_j . We only keep the item j if there exists a Q_i with $u \notin Q_i$ and j is smallest among those k with $v_k^u \in Q_i$. As an example, in Figure 1, we only keep items 2, 4, and 5 corresponding to the edges of cost 3,, 5, and 8. Then it is easy to check that the star $S_j(u)$ has endpoints in exactly $l + 1$ sets Q_i , where j is the l^{th} item in L_u . Moreover, among the stars with center u and endpoints in exactly $l + 1$ sets Q_i , one with minimum power is $S_j(u)$, where j is the l^{th} item in L_u . Notice also that in this situation, l is the decrease in the number of weakly components if $E(S_j(u))$ is added to H . We also keep, for each u , the value $z_u = \min_{j \in L_u} l_j / c_j$, where l_j is the position of

item j in L_u , and the item j_u that achieves this minimum.

We also keep a binary max-heap with all $u \in V$ having as key the value z_u . With these data structures, we can find the star that maximizes the decrease in the number of weakly components of H , divided by the power of the star, if we pick an u with maximum z_u and then use $S_{j_u}(u)$. Finding $S_{j_u}(u)$ is then done in constant time.

Now we describe how the data structures are maintained when some $S_{j_u}(u)$ is added to the set of selected stars. Let l_u be such that j_u is the l_u^{th} item in L_u , and let j_1, \dots, j_{l_u} be such that item j_i is the i^{th} item in L_u . Let Q_{k_0} be the vertex set of the component of u , and Q_{k_i} be the vertex set of the component of v_{j_i} . The way we keep L_u implies that these components are distinct.

The effect of adding $S_j(u)$ to H is the merging into one of the components $Q_{k_0}, Q_{k_1}, \dots, Q_{k_l}$. The algorithm will make these merges one by one, first Q_{k_0} with Q_{k_1} , then the result with Q_{k_2} (if $l \geq 2$), and so on.

Consider such a merge between Q_r and Q_s , and assume by symmetry that $|Q_r| \leq |Q_s|$. We merge Q_r into Q_s ; that is Q_s will be the resulting component. First, for each vertex in Q_r , we add it to the list of vertices of Q_s and change its representative to the representative of Q_s . The running time is $O(n \log n)$ over all the merges, since if we spend time on vertex v , v will become part of a component that has at least twice as many vertices as the component of v before the merge.

Second, we traverse (inorder) the binary tree B_r , and for each v in the tree we proceed as described in the four cases below. In Case 1, $v \notin B_s$; then we insert v in B_s together with the j -index (if any) it has in B_r . In Case 2, $v \in B_s$ and $v \in Q_r$; then the v from B_s also has an index j such that $w_j^v \in Q_s$ and such that j is the only item in L_v among those k with $w_k^v \in Q_s$. We update B_s to mark that $v \in Q_s$. We also remove item j from L_v , updating if necessary, z_v , l_v , and the binary max-heap which keeps vertices u with keys z_u . Case 3 is when $v \in Q_s$ (and thus $v \in B_s$), and $v \notin Q_r$ (recall that $v \in B_r$); in this case, the v in B_r also has an index j such that $w_j^v \in Q_r$ and such that j is the only item in L_v among those k with $w_k^v \in Q_r$. We also remove item j from L_v , updating if necessary, z_v , l_v , and the binary max-heap which keeps vertices u with keys z_u . We update B_s to mark that $v \in Q_s$. Case 4 is when $v \in B_s$, but $v \notin (Q_r \cup Q_s)$; then we have two indices j (from the v in B_r) and j' (from the v in B_s), such that $w_j^v \in Q_s$, and $w_{j'}^v \in Q_r$, and such j is the only item in L_v among those k with $w_k^v \in Q_s$, and such that j' is the only item in L_v among those k with $w_k^v \in Q_r$. We will keep the smaller of j, j' for the v in B_s , and remove the larger of j, j' from L_v , updating if necessary, z_v , l_v , and the binary max-heap which keeps vertices u with keys z_u .

To analyze the overall time of this updates, consider

this: each directed edge uv_j^u appears in L_u initially, but will only be removed once, with a time of $O(\ln^2 n)$. This time is also enough for updating z_u after this removal, and updating the position of u in the max-heap after z_u changes.

When we merge B_r in B_s above, other than removals from L_v 's, we spend $O(\log n)$ per element of B_r , to find and if necessary insert it in B_s . Say we process a $v \in B_r$. If v appears in B_r without a j , or in other words, $v \in Q_r$, then we charge this $O(\log n)$ to v . Vertex v can be charged at most $\lg n$ times this way, as each time it belongs to a component with at least twice as many vertices. If v appears in B_r with a j , then we are in the following case: there a vertex $w_j^v \in Q_r$, the head of a directed edge vw_j^v . We charge the time spent to directed edge vw_j^v . Notice that w_j^v will belong to a component twice the size, and thus edge vw_j^v can be charged at most $\lg n$ times. Each charge is $O(\lg n)$, thus we spend $O(\lg^2 n)$ per vertex and per directed edge of v .

In conclusion, the running time of the Hypergraph-Greedy algorithm, implemented with these data structures is $O(m \log^2 n)$.

References

- [1] G.-S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. IEEE FOCS*, pages 617–626, 2002.
- [2] M. Cagalj, J.-P. Hubaux, and C. Enz. Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues. In *Proc. ACM Mobicom*, pages 172–182, 2002.
- [3] M. Cagalj, J.-P. Hubaux, and C. Enz. Energy-efficient broadcasting in all-wireless networks. *Wirel. Netw.*, 11(1-2):177–188, 2005.
- [4] G. Calinescu, S. Kapoor, A. Olshevsky, and A. Zelikovsky. Network lifetime and power assignment in ad-hoc wireless networks. In *Proc. ESA*, pages 114–126, 2003.
- [5] G. Calinescu and K. Qiao. Asymmetric topology control: Exact solutions and fast approximations. In *Proc. IEEE INFOCOM*, pages 783–791, 2012.
- [6] I. Caragiannis, M. Flammini, and L. Moscardelli. An Exponential Improvement on the MST Heuristic for Minimum Energy Broadcasting in Ad Hoc Wireless Networks. In *Proc. ICALP*, pages 447–458, 2007.
- [7] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. New results for energy-efficient broadcasting in wireless networks. In *Proc. ISAAC*, pages 332–343, 2002.
- [8] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. A logarithmic approximation algorithm for the minimum energy consumption broadcast subgraph problem. *Inf. Process. Lett.*, 86(3):149–154, 2003.
- [9] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Energy-efficient wireless network design. *Theor. Comp. Sys.*, 39(5):593–617, 2006.
- [10] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operation Research*, 4:233–235, 1979.
- [11] A. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca. On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In *Proc. STACS*, pages 121–131, 2001.
- [12] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:634–652, 1998.
- [13] G. Baudis and C. Gropl and S. Hougardy and T. Nierhoff and H. J. Prömel. Approximating minimum spanning sets in hypergraphs and polymatroids. In *Proc. ICALP*, 2000.
- [14] S. Guha and S. Khuller. Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets. *Information and Computation*, 150:57–74, 1999.
- [15] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19:104–115, 1995.
- [16] S. Krumke, R. Liu, E. Lloyd, M. Marathe, R. Ramanathan, and S.S. Ravi. Topology control problems under symmetric and asymmetric power thresholds. In *Proc. Ad-Hoc Now*, pages 187–198, 2003.
- [17] W. Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proc. ACM MOBIHOC*, pages 112–122. ACM Press, 2002.
- [18] F. Mtenzi and Y. Wan. The minimum-energy broadcast problem in symmetric wireless ad hoc networks. In *Proc. WSEAS ACOS*, pages 68–76, 2006.
- [19] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981.
- [20] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, 2005.
- [21] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

- [22] S. Singh, C. S. Raghavendra, and J. Stepanek. Power-aware broadcasting in mobile ad hoc networks. In *Proc. IEEE PIMRC*, 1999.
- [23] P.-J. Wan, G. Calinescu, X.-Y. Li, and O. Frieder. Minimum Energy Broadcast Routing in Static Ad Hoc Wireless Networks. *Wireless Networks*, 8(6):607–617, 2002.
- [24] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proc. IEEE INFOCOM*, pages 585–594, 2000.
- [25] L.A. Wolsey. Analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2:385–392, 1982.
- [26] A. Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. Technical Report CS-96-06, Department of Computer Science, University of Virginia, 1996.

Planar Convex Hull Range Query and Related Problems

Nadeem Moidu *

Jatin Agarwal †

Kishore Kothapalli ‡

Abstract

We consider the planar convex hull range query problem. Let P be a set of points in the plane. We preprocess these points into a data structure such that given an orthogonal range query, we can report the convex hull of the points in the range in $O(\log^2 n + h)$ time, where h is the size of the output. The data structure uses $O(n \log n)$ space. This improves the previous bound of $O(\log^5 n + h)$ time and $O(n \log^2 n)$ space. Given a range query, it also supports extreme points in a given direction, tangent queries through a given point, and line-hull intersection queries on the points in the range in time $O(\log^2 n)$ for each orthogonal query and $O(\log n)$ time for each additional query on that range. These problems have not been studied before.

1 Introduction

Planar convex hull is a well studied topic in computational geometry. Let P be a set of n points on the plane. Overmars and van Leeuwen gave a data structure which allows insertions and deletions in P in $O(\log^2 n)$ time and reporting of points on the hull in $O(\log n + h)$ time, where h is the number of points on the hull [7]. Instead of supporting reporting of the entire hull, recent works provide data structures to support common queries on the convex hull, $CH(P)$, without actually computing it. The following queries are typically studied:

1. *Extreme point query*: Find the most extreme vertex in $CH(P)$ along a query direction
2. *Tangent query*: Find the two vertices of $CH(P)$ that form tangents with a query point outside the hull
3. *Line stabbing query*: Find the intersection of $CH(P)$ with an arbitrary query line

These queries can be supported in $O(\log n)$ time by the structure of Overmars and van Leeuwen [7]. Brodal and Jacob gave a solution which supports insertions and deletions in $O(\log n)$ amortized time and the first two queries in $O(\log n)$ time without actually computing the hull [3]. Chan gave a data structure which supports the

third query in $O(\log^{3/2} n)$ time [4] and later improved it to $O(\log^{1+\epsilon} n)$ time [6].

In this paper, we study the orthogonal range query versions of the above problems. Given an orthogonal range query of the form $q = [x_{low}, x_{high}] \times [y_{low}, y_{high}]$, we support the above queries for the points in $P \cap q$. Brass et al. first gave a solution to report the convex hull of an orthogonal range in $O(\log^5 n + h)$ time in [2]. The other problems are being studied for the first time but the data structure in [2] can be enhanced to support these queries in $O(\log^5 n)$ time per orthogonal range query and an additional $O(\log^2 n)$ time for any of the above queries. Our data structure takes $O(\log^2 n)$ time to process one orthogonal range query. Once this is done, we can report the points on the hull of $P \cap q$ in $O(h)$ time and any of the above queries in $O(\log n)$ time.

2 Overview

In a standard two dimensional range tree, a query is divided vertically into $O(\log n)$ rectangular regions where each region corresponds to a canonical node in the primary tree. Each of these primary regions are further divided horizontally into $O(\log n)$ regions corresponding to canonical nodes in the secondary tree. However, these horizontally divided regions are independent of each other, i.e. they correspond to different intervals in different primary regions. In our data structure we modify the secondary trees such that, for a given query, the horizontal divisions are same across all canonical primary node regions. So an orthogonal query is divided into a grid of $O(\log n) \times O(\log n)$ regions which are perfectly aligned as shown in figure 1. By having the divisions aligned like this, we are able to discard a large number of regions which do not contribute to the final hull without processing them. This idea is similar to the method used by Abam et al. to enhance kinetic kd-trees in [1].

3 Data Structure

Let P be a static set of points on a plane. We construct a one dimensional range tree, T_y of all the points based on their y coordinates. We call this the *template* tree. Given a subset S of the point set P , the *contracted* tree of T_y with respect to S is defined as the tree obtained

*nadeem.moiduug08@students.iiit.ac.in

†jatin.agarwal@research.iiit.ac.in

‡kishore@iiit.ac.in

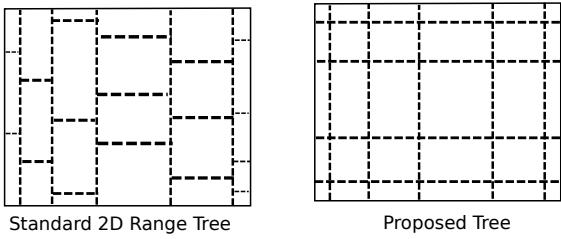


Figure 1: Example of how a query is split into canonical node regions for (a) normal 2D range tree and (b) our tree

by removing all subtrees which do not have a leaf in S and contracting all nodes which have only one child. See Figure 2. Since a contracted tree is a full binary tree (i.e. a tree in which each node has exactly either zero or two children), it takes $O(|S|)$ space.

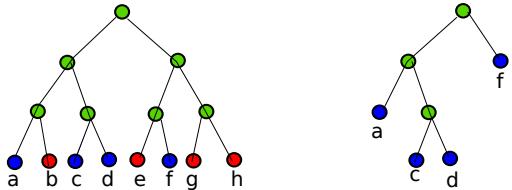


Figure 2: Original tree and the contracted tree for the set $\{a, c, d, f\}$

Next, we construct a primary tree, T , which is a one dimensional range tree based on the x coordinates. Each node in this primary tree contains a secondary tree which is a contracted version of T_y with respect to the points in the corresponding x range.

A convex hull can be divided into four parts based on the extreme points along each axis. The upper right part goes from the point with maximum y coordinate to the point with maximum x coordinate. The other parts are similar. Our data structure is designed to compute the upper right part of the convex hull. The other parts can be computed similarly and the four parts can be joined together to obtain the final convex hull. From here on, we will refer to the upper right convex hull as *urc-hull*.

We now describe the information stored at each secondary tree node. Each internal secondary tree node corresponds to a set of points which is a union of two disjoint sets of points, separated by a horizontal line. So the *urc-hull* of the points in a node will comprise a part of each of the child node *urc-hulls* and the outer common tangent (bridge) connecting them. We store the following variables in each internal node, u :

1. A boolean variable which indicates whether the left child (horizontally lower part) contributes any part to the *urc-hull*. If this variable is false, then the next three parameters are set to *NULL*.
2. The outer common tangent (bridge) connecting the *urc-hulls* of the children, represented by the points where it intersects the *urc-hulls*, $B_l(u)$ and $B_r(u)$.
3. Both neighbors of $B_l(u)$ and $B_r(u)$ in the *urc-hulls* of the respective child. (These parameters are required for computing the common tangent between two hulls).
4. Indices of $B_l(u)$ and $B_r(u)$ in the *urc-hulls* of the respective child, $Index_l(u)$ and $Index_r(u)$. We need these two parameters to know the portion of each child *urc-hull* that contributes to the *urc-hull* of the current node.
5. The y range spanned by the node.
6. Number of points in the *urc-hull*, $N(u)$.

For leaf nodes, we simply store the point. Since each secondary tree node stores a constant amount of information, the space taken by each primary tree node is proportional to the number of points in the interval. So the overall space taken is $O(n \log n)$.

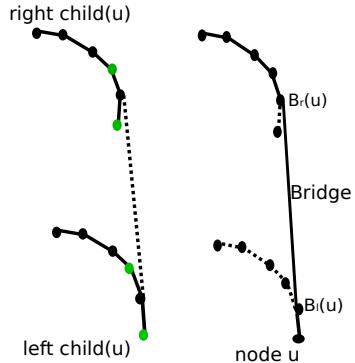


Figure 3: Different parameters stored at each node. The neighbors of $B_l(u)$ and $B_r(u)$ are marked green.

Note that we are not storing the *urc-hull*, as it is, in each node. However, given indices, i and j , we can report the points in the *urc-hull* from i to j in $O(\log n + j - i + 1)$ time as shown in section 5.

4 Preprocessing

The primary tree is constructed as a standard one dimensional range tree based on x coordinates. We pro-

cess the primary tree from top to bottom to obtain the secondary trees (contracted trees) at each primary tree node. The interval at the root node includes all the points in P , so the template tree, T_y , is stored as it is. Note that the interval corresponding to a non-root primary node is a subset of the interval corresponding to its parent. For each child, we replicate the tree present in its parent and then contract it with respect to the points in the interval of the node. A tree can be contracted with respect to a set of points by removing all leaves not present in the child and then updating the parents of the removed leaves as required. Each node in the tree is processed a constant number of times, so the time taken for this stage is $O(n \log n)$.

Once the tree structure is completed, we start storing the required information from bottom to top starting from the leaves. Except the common bridge, all the other parameters of the node can be easily found in constant time. The bridges can be computed as follows: At each node, discard the points of the child hulls which do not form part of the parent hull. Since each point is discarded atmost once, it takes amortized constant time per point. So the time taken for preprocessing is $O(n \log n)$ and the space usage is $O(n \log n)$.

5 Query Algorithm

Given an orthogonal query $[x_{low}, x_{high}] \times [y_{low}, y_{high}]$, we can identify $O(\log n)$ canonical nodes corresponding to the y range, $[y_{low}, y_{high}]$ in the template tree T_y . We will also identify the $O(\log n)$ canonical nodes corresponding to the x range in the primary tree. We then find out the nodes corresponding to the T_y nodes in each of the primary tree nodes. There are three cases here:

1. The node present in T_y exists in the contracted tree as it is. In this case, we simply use that node.
2. The node was removed because both its children were removed. This means that the node was empty, so this node does not contribute any point to the urc -hull.
3. The node was removed because it was the only child of its parent. In this case, we check the node present in its place to get the required information, if any.

So an orthogonal range query gets split into $O(\log n) \times O(\log n)$ secondary tree nodes, which are well aligned as shown in figure 1. These cases can be identified while doing a normal one dimensional range tree query on the secondary nodes.

Lemma 1 *The upper right convex hull of the orthogonal range can be computed in $O(\log^2 n)$ time.*

Proof. We define the region covered by each of the $O(\log n) \times O(\log n)$ secondary tree nodes as a *block*. Start by identifying the non empty *blocks*. If a *block* is non empty, then no *block* which is to its left and bottom can contribute points to the *urc-hull*. A *block* is called a *candidate block* if it is non-empty and all *blocks* to the right and top of its top right corner are empty. See figure 4. Based on this observation, the *candidate blocks* can be identified as follows: start from the bottommost non-empty *block* in the rightmost column. This is a *candidate block*. If there exists at least one non empty *block* above it in the same column, move to the next (non-empty) *block* in the upwards direction. Otherwise, move one *block* to the left. Continue this till we reach the top left *block*. Every *block* visited in this process is a *candidate block*. Since we are moving only up or left, we will move in the left direction at most $O(\log n)$ times and in the up direction at most $O(\log n)$ times. So the total number of *candidate blocks* is at most $O(\log n)$.

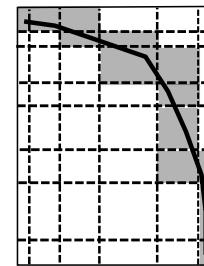


Figure 4: Example of a query. The candidate blocks have been darkened.

Now process the candidate blocks in the order they were visited. The individual *urc-hulls* of each node can be merged together to form the complete *urc-hull* in a manner similar to Graham's scan. This method has been used before, e.g. see [5]. Maintain the *urc-hull* up to the current block in a stack, H . Each block, contributes atmost one continuous range to the *urc-hull*. Each element of the stack contains a pointer to a candidate block, $H(u)$ and the indices of the start and end points of this range, $H_s(u)$ and $H_e(u)$. First, push the right bottom block. Process each subsequent block, v as follows: Compute the common tangent between the current block, v and the *urc-hull* on the top of the stack, $\text{top}(H)$. If this tangent does not intersect $\text{top}(H)$ between the range $\text{top}(H_s)$ and $\text{top}(H_e)$, then the current top does not contribute to the *urc-hull* anymore. So pop out top from the stack and compute the tangent with the new top of the stack. Continue this till the top of the stack is not popped out. Now push the current block to the top of the stack and update the ranges appropriately based on the tangent information.

The time taken is mostly for computing the tangents. This has to be done exactly once for each time a *urc-hull* is pushed or popped. Each tangent computation using the method of Overmars and van Leeuwen [7] takes $O(\log n)$ time. This method compares a point on each of the hulls and discards either the portion before it or the portion after it in the corresponding hull. This is where we use the neighbors of the bridges stored in each secondary tree node. This tangent computation is done $O(\log n)$ times. So the overall time complexity is $O(\log^2 n)$. \square

The above merging algorithm returns a stack of the secondary tree nodes and the indices of the range that each of these nodes contribute to the *urc-hull*. The line segment connecting the end points of two adjacent nodes in this stack gives the bridge connecting them. Using this structure, we can report all the points on the *urc-hull* in $O(\log n + h)$ time as shown in algorithm 1.

Input: Tree node u , Indices i and j
Output: Points on the *urc-hull* corresponding to u from indices i to j , inclusive

```

if  $u$  is a leaf node then
| Report the point if  $i = j = 1$ 
else
| if  $Index_l(u) > i$  then
| | Report points in the range of the left
| | subtree which forms part of the range  $[i, j]$ 
| | in the parent hull
| end
| if  $i \leq Index_l(u) \leq j$  then
| | Report  $B_l(u)$ 
| end
| if  $i \leq Index_l(u) + 1 \leq j$  then
| | Report  $B_r(u)$ 
| end
| if  $Index_l(u) + 1 < j$  then
| | Report points in the range of the right
| | subtree which forms part of the range  $[i, j]$ 
| | in the parent hull
| end
end

```

Algorithm 1: Algorithm to report the points on the *urc-hull* of a node from given indices i to j , inclusive

5.1 Other Problems

We now explain how to solve the extreme point query, line stabbing query and tangent query problems using the stack obtained above without constructing the entire convex hull. Recall that the convex hull was divided into four parts based on the extreme points. For the problems discussed in this section, the answer could be in any of these four parts. It is easy to identify the exact

part(s) by comparing the extreme points with the query parameter.

The basic idea is the following: By comparing an edge of the hull with a query parameter, we can discard, in constant time, either the part before the edge or the part after the edge. We proceed as follows: Compare each bridge connecting adjacent elements in the stack with the query parameter. If the required output lies on one of the bridges, then we report the answer and stop. Otherwise, this will help in identifying the exact node which contains the required output. This takes time proportional to the number of bridges, which is $O(\log n)$. Once the node is identified, by comparing the bridge at the root of the node with the query parameter, we can decide whether we should take the root, or go to the left or right subtree. This takes time proportional to the height of the tree which is also $O(\log n)$. So the overall time taken is $O(\log n)$.

For line stabbing query, there are at most two points to be reported. We have to query for each of them separately. Otherwise, it will not be possible to discard a half at each stage in the above algorithm.

6 Future Work

It might be possible to improve the bounds given in this paper. A more interesting problem is to make the set of points dynamic by allowing insertions and/or deletions. The problem is also open in higher dimensions.

The modified range tree approach can be used to improve various range query problems like reporting the smallest enclosing disk or the width of the points in a query rectangle.

References

- [1] M. A. Abam, M. de. Berg and B. Speckmann, Kinetic kd-Trees and Longest-Side kd-Trees. In SICOMP 39:1219-1232, 2009.
- [2] P. Brass, C. Knauer, C. S. Shin, M. Schmid and I. Vigan. Range-Aggregate Queries for Geometric Extent Problems. In Proc. 19th Computing: Australasian Theory Sympos. CATS 141:3-10, 2013.
- [3] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In Proc. 43rd IEEE Sympos. Found. Comput. Sci., pages 617-626, 2002.
- [4] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. In J. ACM, 48:1-12, 2001.
- [5] T. M. Chan. and E. Y. Chen, Multi-Pass Geometric Algorithms. In Discrete and Comput. Geom., 37(1): 79-102, 2007.
- [6] T. M. Chan. Three Problems about Dynamic Convex Hulls. In Proc. 27th ACM Sympos. Comput. Geom. 27-36, 2011.
- [7] M. H. Overmars and J. van Leeuwen. Maintenance of Configurations in the Plane. In J. Comput. Syst. Sci. 23:166-204, 1981.

What's in a gaze? How to reconstruct a simple polygon from local snapshots

Peter Widmayer

Department of Computer Science, ETH Zurich, Switzerland

Given a bunch of observations of geometric features of an unknown geometric object, we want to reconstruct the object that gave rise to our observations. Under which conditions is this possible, when is the solution unique, and how can we find it? Classical computational geometry questions like these have recently been studied with a twist coming from the current interest in mobile agents that explore unknown environments.

We discuss a particular family of problems of this flavour: For very simple mobile agents that can move only from vertex to vertex, what types of observations are needed to reconstruct a simple polygon? We allow the mobile agents to take snapshots of a variety of features at the vertices. Our interest is not in the efficiency of the reconstruction, but merely in the question of which local observations lead to a unique solution. Ideas from distributed computing appear to help to reconstruct the polygon topology, even in a situation in which the observations are essentially non-geometric.

The work discussed in this talk has been carried out with Davide Bilo, Jeremie Chalopin, Shantanu Das, Yann Disser, Beat Gfeller, Matus Mihalak, Subhash Suri, and Elias Vicari.

Index

- Abam, Mohammad Ali, 265
Abrahamsen, Mikkel, 157
Agarwal, Jatin, 307
Aichholzer, Oswin, 73, 169, 205
Allen, River, 241
Aloupis, Greg, 73
Alvarez, Victor, 85, 135
Angelini, Patrizio, 117
Aronov, Boris, 211, 259
Assadi, Sepehr, 235
- Bae, Sang Won, 205
Banik, Aritra, 37
Barba, Luis, 205, 217, 229
Barbay, Jérémie, 151
Beingessner, Alexis, 217
Bhattacharya, Bhaswar, 37
Biedl, Therese, 13
Biro, Michael, 129
Bose, Prosenjit, 175, 205, 217
Bringmann, Karl, 85
Busto, Daniel, 283
- Calinescu, Gruia, 301
Cardinal, Jean, 169
Chambers, Erin, 19, 135
Chan, Timothy M., 151, 289
Chaplick, Steven, 141
Chen, Ke, 271
Clement, Christopher, 199
Cohen, Elad, 141
Crepaldi, Bruno, 223
- Das, Sandip, 37
De Carufel, Jean-Lou, 175
Demaine, Erik D., 31, 43, 73
Demaine, Martin L., 43, 73
Dumitrescu, Adrian, 271
Durocher, Stephane, 229
- Emamjomeh-Zadeh, Ehsan, 235
Eppstein, David, 61, 117
Evans, William, 283
- Fekete, Sándor P., 73
Fraser, Robert, 229
Frati, Fabrizio, 117
- Gagie, Travis, 145
Gao, Jie, 129
Gawrychowski, Paweł, 145
Genov, Blagoy, 97
Gheibi, Amin, 123
Ghosh, Anirban, 271
Goodrich, Michael, 61, 181
Grimm, Carsten, 175
- Hackl, Thomas, 169
Held, Martin, 13
Heyer, Laurie, 241
Hoffmann, Michael, 73, 295
Hu, Nan, 289
Huber, Stefan, 13
Hurtado, Ferran, 169, 229
- Iacono, John, 211
Imai, Keiko, 187
Iwerks, Justin, 129
- Ju, Tao, 19
- Kaaser, Dominik, 13
Kaufmann, Michael, 117
Kawamura, Akitoshi, 25
Kazemi, Mohammad Reza, 265
Khanteimouri, Payam, 265
Kirkpatrick, David, 283
Korman, Matias, 169, 205
Kostitsyna, Irina, 129
Kothapalli, Kishore, 307
Kozma, László, 135
Kumar, Nirman, 103
Kusters, Vincent, 295
- Löffler, Maarten, 163
Lazard, Sylvain, 117
Lee-St.John, Audrey, 199
Letscher, David, 19
Lubiw, Anna, 73
- Maheshwari, Anil, 123, 175
Mchedlidze, Tamara, 117
Mehrabi, Saeed, 229
Mitchell, Joseph S. B., 109, 129
Mohades, Ali, 265
Moidu, Nadeem, 307

- Mondal, Debajyoti, 229
 Morgenstern, Gila, 141
 Morrison, Jason, 229
 Mukherjee, Satyaki, 37
 Navarro, Gonzalo, 151
 Nekrich, Yakov, 145
 Nishat, Rahnuma Islam, 49, 241
 O'Rourke, Joseph, 79
 Okamoto, Takuma, 25
 Okamoto, Yoshio, 31
 Özkan, Özgür, 211
 Pérez-Lantero, Pablo, 151
 Palfrader, Peter, 13
 Panahi, Fatemeh, 247
 Pilz, Alexander, 169
 Pszona, Paweł, 181
 Raichel, Benjamin, 103
 Ray, Saurabh, 85
 Renssen, André van, 205
 Rezende, Pedro de, 223
 Rote, Günter, 295
 Sack, Jörg-Rüdiger, 123
 Saumell, Maria, 295
 Schulz, André, 163
 Seidel, Raimund, 85
 Shaffer, Alla, 115
 Sheehy, Donald R., 253
 Sidman, Jessica, 199
 Silveira, Rodrigo, 169, 295
 Simons, Joseph A., 61
 Skala, Matthew, 229
 Smid, Michiel, 175, 217
 Snoeyink, Jack, 73
 Souza, Cid de, 223
 Stappen, A. Frank van der, 247
 Sun, Timothy, 193
 Taslakian, Perouz, 205
 Tatsu, Yuichi, 25
 Teillaud, Monique, 117
 Tóth, Csaba, 163
 Uehara, Ryuhei, 31, 43, 169
 Uno, Yushi, 25, 31
 Verdonschot, Sander, 205
 Viglietta, Giovanni, 55, 277
 Vogtenhuber, Birgit, 169
 Wagner, David P., 67
 Wahid, Mohammad Abdul, 229
 Wang, Haitao, 91
 Welzl, Emo, 169
 Whitesides, Sue, 11, 49, 241
 Widmeyer, Peter, 311
 Winslow, Andrew, 73
 Wolff, Alexander, 117
 Yagnatinsky, Mark, 211, 259
 Yamato, Masahide, 25
 Yazdanbod, Sadra, 235
 Yokosuka, Yusuke, 187
 Zarrabi-Zadeh, Hamid, 235