Technische Universität Braunschweig

The 29th European Workshop on Computational Geometry

# EuroCG 2013

March 17-20, 2013, Braunschweig, Germany

# Preface

Computer Science is arguably *the* universal scientific field of our time. There is no discipline that does not make use of tools from computer science; what is more, the fundamental insights and perspectives of computation provide a powerful outlook on our world, so that the mere *observation* of phenomena in nature gives way to active *construction* of complex systems.
Computational Geometry reflects this broad range of topics and objectives of Computer Science. It reaches all the way from pure theory to practical application, some of which involve diverse fields such as visualization, robotics, computer graphics, and geographic information systems.
The community of researchers working in Computational Geometry spans the world. Consisting of hundreds of professional researchers, young and old, its work is conducted in a spirit of friendly and cooperative collaboration, and fostered by a wide range of meetings, workshops and conferences.
Among these, the European Workshop on Computational Geometry (EuroCG) plays an important and prominent role as an annual, informal workshop whose goal is to provide a forum for scientists to meet, present their work, interact, and establish collaborations, in order to promote research in the field of Computational Geometry, within Europe and beyond. The workshop aims at providing an informal atmosphere through which both established and young researchers will have a productive exchange of ideas and collaboration.
EuroCG does not have formally reviewed proceedings; a book of abstracts is distributed electronically and does not have an ISBN. Therefore, results presented at EuroCG are often also submitted to peer-reviewed conferences and/or journals. This volume presents the abstracts of the 29th EuroCG, which takes place from March 17–20, 2013 in Braunschweig.
We are happy to welcome you to this city of science, and hope that you will find many opportunities to discover new ideas, meet other scientists, extend existing collaborations, start new ones—and enjoy our wonderful environment!

March 2013 Sándor P. Fekete

# Organization

## Local Arrangements

Sándor P. Fekete, TU Braunschweig
Christiane Schmidt, TU Braunschweig

## Program Commitee

Dominique Attali, CNRS, France
Erin Chambers, Saint Louis U., USA
Tamal Dey, Ohio State U., USA
Sándor Fekete, TU Braunschweig (chair), Germany
Jie Gao, Stony Brook U., USA
Joachim Giesen, Jena U., Germany
Sariel Har-Peled, U. Illinois at Urbana-Champaign, USA
Michael Hemmer, Tel-Aviv U., Israel
Ferran Hurtado, U. Politècnica de Catalunya, Spain
Michael Kerber, Stanford U., USA
David Kirkpatrick, U. of British Columbia, Canada
Christian Knauer, U. Bayreuth, Germany
Alexander Kröller, TU Braunschweig, Germany
Marc van Kreveld, U. Utrecht, The Netherlands
Sylvain Lazard, INRIA Nancy, France
Maarten Löffler, U. Utrecht, The Netherlands
Alejandro López-Ortiz, U. of Waterloo, Canada
Henk Meijer, Roosevelt Academy, The Netherlands
Joe Mitchell, Stony Brook University, USA
Christiane Schmidt, TU Braunschweig, Germany
Bettina Speckmann, TU Eindhoven, The Netherlands
Monique Teillaud, INRIA Sophia Antipolis, France
Jan Vahrenhold, U. Münster, Germany
Carola Wenk, Tulane U., USA

## Additional Referees

| | |
|---|---|
| Mahmuda Ahmed | Christian Scheffer |
| Olivier Devillers | Jessica Sherette |
| Robert Fraser | Isabelle Sivignon |
| Shahin Kamali | Frank Staals |
| André Lieutier | Fabian Stehn |
| Quentin Merigot | Remy Thomasse |
| Jazmin Romero | Sonia Waharte |

Support

PLANTAGEN Rausch SCHOKOLADE

Rausch Schokolade

HEIMBS
MANUFAKTUR SEIT 1880

Heimbs Kaffee

# Table of Content

# Schedule

## MONDAY, MARCH 18, 2013

**09:00-09:15:** Opening

**09:15-09:30:** Fast Forward: 1A, 1B, 2A, 2B

**09:40-11:10:** Session 1A—Visibility and Guarding

**1A1** Facets for Art Gallery Problems. *Sándor P. Fekete, Stephan Friedrichs, Alexander Kröller and Christiane Schmidt*

**1A2** The Minimum Guarding Tree Problem. *Adrian Dumitrescu, Joseph Mitchell and Paweł Żyliński*

**1A3** Covering Class-3 Orthogonal Polygons with the Minimum Number of r-Stars. *Leonidas Palios and Petros Tzimas*

**1A4** Extending Visibility Polygons by Mirrors to Cover Specific Targets. *Arash Vaezi and Mohammad Ghodsi*

**09:40-11:10:** Session 1B—Topology

**1B1** Rotation Minimizing Frames on Monotone-helical Quintics: Approximation and Applications to Modeling Problems. *Fatma Şengüler-Çiftçi and Gert Vegter*

**1B2** Layered Reeb Graphs of a Spatial Domain. *Birgit Strodthoff and Bert Jüttler*

**1B3** Collapsing Rips Complexes. *David Salinas, Dominique Attali and André Lieutier*

**1B4** Constructing Complicated Spheres. *Mimi Tsuruga and Frank H. Lutz*

**11:00-11:30:** Coffee Break

**11:30-12:30:** Session 2A—Packing and Covering

**2A1** Selecting a Small Covering from a Double Covering. *Peter Brass*

**2A2** Smart-grid Electricity Allocation via Strip Packing with Slicing. *Soroush Alamdari, Therese Biedl, Timothy M. Chan, Elyot Grant, Krishnam Raju Jampani, S. Keshav, Anna Lubiw and Vinayak Pathak*

**2A3** Packing Identical Simple Polygons of Constant Complexity is NP-hard. *Ning Xu*

**11:30-12:30:** Session 2B—Fréchet and Hausdorff

**2B1** Computing the Fréchet Distance with Shortcuts is NP-Hard. *Maike Buchin, Anne Driemel and Bettina Speckmann*

**2B2** Parallel Computation of the Hausdorff Sistance between Shapes. *Helmut Alt and Ludmila Scharf*

**2B3** Hardness Results on Curve/Point Set Matching with Fréchet Distance. *Paul Accisano and Alper Üngör*

**12:30-14:00:** Lunch Break

**14:00-15:00:** Session 3—Invited Talk: *Konrad Polthier* Differential-Based Geometry.

**15:00-15:10:** Fast Forward: 4A, 4B

**15:10-15:40:** Coffee Break

**15:40-17:00:** Session 4A—Convexity

**4A1** New Results on Convex Stabbers. *Lena Schlipf*

**4A2** Unions of Onions. *Maarten Löffler and Wolfgang Mulzer*

**4A3** Finding a Largest Empty Convex Subset in Space Is W[1]-Hard. *Panos Giannopoulos and Christian Knauer*

**4A4** The Degree of Convexity. *Günter Rote*

**15:40-16:40:** Session 4B—Distance Problems

**4B1** Algorithms for Distance Problems in Planar Complexes of Global Nonpositive Curvature. *Daniela Maftuleac*

**4B2** On the Complexity of Finding Spanner Paths. *Mikael Nilsson*

**4B3** Polylogarithmic Approximation for Generalized Minimum Manhattan Networks. *Aparna Das, Krzysztof Fleszar, Stephen Kobourov, Joachim Spoerhase, Sankar Veeramoni and Alexander Wolff*

**17:15-18:15:** Business Meeting

## TUESDAY, MARCH 19, 2013

**09:00-10:00:** Session 5—Invited Talk: *James McLurkin.*
Distributed Computational Geometry and Multi-Robot Systems: Twins Separated at Birth?

**10:00-10:20:** Fast Forward: 6A, 6B, 7A, 7B, 8A, 8B

**10:20-10:50:** Coffee Break

**11:10-12:10:** Session 6A—Representation and Reconstruction

**6A2** Topologically Safe Curved Schematization. *Arthur van Goethem, Herman Haverkort, Wouter Meulemans, Andreas Reimer and Bettina Speckmann*

**6A3** Straight-Line Triangle Representations. *Nieke Aerts and Stefan Felsner*

**6A4** Reconstructing Polygons from Embedded Straight Skeletons. *Therese Biedl, Martin Held and Stefan Huber*

**10:50-12:10:** Session 6B—Higher-Dimensional Problems

**6B1** A Conceptual Take on Invariants of Low-Dimensional Manifolds Found by Integration. *Mathijs Wintraecken and Gert Vegter*

**6B2** Cut Equivalence of d-Dimensional Guillotine Partitions. *Andrei Asinowski, Gill Barequet, Toufik Mansour and Ron Pinter*

**6B3** Approximating Weighted Geodesic Distances on 2-Manifolds in $R^3$. *Christian Scheffer and Jan Vahrenhold*

**6B4** Efficient Volume and Edge-Skeleton Computation for Polytopes Given by Oracles. *Ioannis Emiris, Vissarion Fisikopoulos and Bernd Gaertner*

**12:10-13:30:** Lunch Break

**13:30-14:30:** Session 7A—Triangulations

**7A1** Flip Distance Between Triangulations of a Simple Polygon is NP-Complete. *Oswin Aichholzer, Wolfgang Mulzer and Alexander Pilz*

**7A2** Selecting the Aspect Ratio of a Scatter Plot Based on Its Delaunay Triangulation. *Martin Fink, Jan-Henrik Haunert, Joachim Spoerhase and Alexander Wolff*

**7A3** Computational Aspects of Triangulations with Bounded Dilation. *Wolfgang Mulzer and Paul Seiferth*

**13:30-14:30:** Session 7B—Drawing and Embedding

**7B1** Exploiting Air Pressure to Map Floorplans on Point Sets. *Stefan Felsner*

**7B2** Book Embeddings of Iterated Subdivided-Line Graphs. *Toru Hasunuma*

**7B3** Area Requirement of Graph Drawings with Few Crossings per Edge. *Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta and Fabrizio Montecchiani*

**14:30-15:00:** Coffee Break

**15:00-16:00:** Session 8A—Reconfiguration

**8A1** Distributed Universal Reconguration of 2D Lattice-Based Modular Robots. *Ferran Hurtado, Enrique Molina, Suneeta Ramaswami and Vera Sacristan*

**8A2** The Number of Different Unfoldings of Polyhedra. *Takashi Horiyama and Wataru Shoji*

**8A3** Computational Complexity of Piano-Hinged Dissection. *Zachary Abel, Erik D. Demaine, Martin L. Demaine, Takashi Horiyama and Ryuhei Uehara*

**15:00-16:00:** Session 8B—Combinatorics

**8B1** Coding Ladder Lotteries. *Tomoki Aiuchi, Katsuhisa Yamanaka, Takashi Hirayama and Yasuaki Nishitani*

**8B2** On Counting Triangles, Quadrilaterals and Pentagons in a Point Set. *Sergey Bereg and Kira Vyatkina*

**8B3** Randomized Incremental Construction of the Hausdorff Voronoi Diagram of Non-Crossing Clusters. *Panagiotis Cheilaris, Elena Khramtcova and Evanthia Papadopoulou*

**16:00-19:00:** Social Event

**19:30-** Conference Banquet

## WEDNESDAY, MARCH 20, 2013

**09:00-10:00:** Session 9—Invited Talk: *Marcus Magnor.* Geometry and Images

**10:00-10:20:** Fast Forward: 10A, 10B, 11A, 11B, 12A, 12B

**10:20-10:50:** Coffee Break

**10:50-12:10:** Session 10A—Clustering

**10A1** Balanced Partitions of 3-colored Geometric Sets in the Plane. *Sergey Bereg, Ferran Hurtado, Mikio Kano, Matias Korman, Dolores Lara, Carlos Seara, Rodrigo Silveira, Jorge Urrutia and Kevin Verbeek*

**10A2** The Complexity of Separating Points in the Plane. *Sergio Cabello and Panos Giannopoulos*

**10A3** Kinetic Euclidean 2-centers in the Black-Box Model. *Mark de Berg, Marcel Roeloffzen and Bettina Speckmann*

**10A4** New Representation Results for Planar Graphs. *Farhad Shahrokhi*

**10:50-12:10:** Session 10B—Voronoi and Relatives

**10B1** New Sequential and Parallel Algorithms for Computing Beta-spectrum. *Gabriela Majewska and Mirosław Kowaluk*

**10B2** Voronoi Diagrams from Distance Graphs. *Mario Kapl, Franz Aurenhammer and Bert Jüttler*

**10B3** A Sweepline Algorithm for Higher Order Voronoi Diagrams. *Evanthia Papadopoulou and Maksym Zavershynskyi*

**10B4** On the Complexity of the Partial Least-Squares Matching Voronoi Diagram. *Matthias Henze, Rafel Jaume and Balazs Keszegh*

**12:10-13:30:** Lunch Break

**13:30-14:30:** Session 11A—Optimal Motion

**11A1** A Truly Local Strategy for Ant Robots Cleaning Expanding Domains. *Rolf Klein, David Kriesel and Elmar Langetepe*

**11A2** 2-Modem Pursuit-Evasion Problem. *Yeganeh Bahoo Torudi, Ali Mohades, Marzieh Eskandari and Mahsa Sorouri*

**11A3** Euclidean Traveling Salesman Tours through Stochastic Neighborhoods. *Pegah Kamousi and Subhash Suri*

**13:30-14:30:** Session 11B—Integer Distance

**11B1** Large-Volume Open Sets in Normed Spaces without Integral Distances. *Sascha Kurz and Valery Mishkin*

**11B2** Clear Unit-Distance Graphs. *Marc van Kreveld, Maarten Löffler and Frank Staals*

**11B3** Extending Partial Representations of Proper and Unit Interval Graphs. *Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell and Tomas Vyskocil*

**14:30-15:00:** Coffee Break

**15:00-16:00:** Session 12A—Matching

**12A1** Augmentability of Geometric Matching and Needlework. *Tillmann Miltzow*

**12A2** Quasi-Parallel Segments and Characterization of Unique Bichromatic Matchings. *Andrei Asinowski, Tillmann Miltzow and Günter Rote*

**12A3** Hierarchical Flows with an Application to Image Matching. *Stefan Funke and Sabine Storandt*

**15:00-16:00:** Session 12B—Labeling

**12B1** Two-Sided Boundary Labeling with Adjacent Sides. *Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz and Alexander Wolff*

**12B2** Trajectory-Based Dynamic Map Labeling. *Andreas Gemsa, Benjamin Niedermann and Martin Nöllenburg*

**12B3** Dynamic Point Labeling Is Strongly PSPACE-hard. *Kevin Buchin and Dirk H.P. Gerrits*

**16:00-16:10:** Closing Remarks

# Facets for Art Gallery Problems

Sándor P. Fekete* Stephan Friedrichs* Alexander Kröller* Christiane Schmidt*

**Abstract** We discuss polyhedral methods for computing optimal solutions for large instances of the Art Gallery Problem (AGP). We extend our previous work [7], which uses a primal-dual linear programming approach to solve the fractional AGP to optimality, using cutting planes that eliminate fractional solutions.

We identify two classes of facets of the associated polytopes, based on Edge Cover (EC) and Set Cover (SC) inequalities. Solving the separation problem for these facets is NP-complete, but exploiting the underlying geometric structure of the AGP, we show that large subclasses of fractional SC solutions cannot occur for the AGP. This allows us to separate the relevant subset of facets in polynomial time. Finally, we characterize all facets for finite AGP relaxations with coefficients in $\{0, 1, 2\}$.

## 1 Introduction

The Art Gallery Problem (AGP) asks for the minimum number of points that can guard a given polygonal region $P$ with $n$ vertices. Chvátal [3] (and Fisk [6]) showed that $\lfloor \frac{n}{3} \rfloor$ guards are sometimes necessary and always sufficient for a simple polygon $P$. See O'Rourke [9] for a classical survey.

Algorithmically, the AGP is closely related to the Set Cover (SC) problem; it is NP-hard, even for simple polygons [8]. However, there are two differences to a discrete SC problem. On the one hand, geometric variants of problems may be easier to solve or approximate than their discrete, graph-theoretic counterparts; on the other hand, the AGP is far from being discrete: both the set that is to be covered (all points in $P$) and the covering family (all visibility polygons within $P$) usually are uncountably infinite.

Amit, Mitchell and Packer [1] have considered purely combinatorial primal and dual heuristics for general AGP instances. Only very recently have researchers begun to combine methods from integer linear programming with non-discrete geometry in order to obtain optimal solutions. As we showed in [7], it is possible to combine an iterative primal-dual relaxation approach with structures from computational geometry in order to solve AGP instances with unrestricted guard positions. Couto et al. [5] used a similar approach for the AGP with vertex guards. Closely related to this paper, Balas and Ng [2] describe all facets with coefficients in $\{0, 1, 2\}$ of the discrete SC polytope.

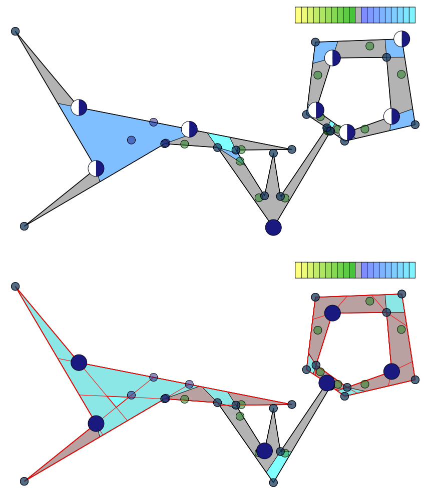


Figure 1: (Top) An optimal fractional solution of an AGP instance. The half-filled circles indicate $\frac{1}{2}$-guards. (Bottom) Cutting planes from Sections 3 and 4 force 2 guards in the left and 5 in the right part of the polygon respectively, thus yielding an integer optimum.

**Formal Description** We consider a polygonal region $P$, possibly with holes, with $n$ vertices. For a point $p \in P$, we denote by $\mathcal{V}(p)$ the *visibility polygon* of $p$ in $P$. $P$ is *star-shaped* if $P = \mathcal{V}(p)$ for some $p \in P$. The set of all such points is the *kernel* of $P$. For a set $S \subseteq P$, $\mathcal{V}(S) := \cup_{p \in S} \mathcal{V}(p)$. A set $C \subseteq P$ is a *guard cover*, if $\mathcal{V}(C) = P$. The AGP asks for a guard cover of minimum cardinality.

**Our Results** In this paper, we extend and deepen our previous work on iterative primal-dual relaxations, by proving a number of polyhedral properties of the resulting AGP polytopes.

- We show how to employ cutting planes for an iterative primal-dual framework for solving the AGP. This is interesting in itself, as it provides an approach to tackling optimization problems with infinitely many constraints and variables. The particular challenge is to identify constraints that

*Department of Computer Science, TU Braunschweig, Germany. `{s.fekete, stephan.friedrichs, a.kroeller, c.schmidt}@tu-bs.de`

remain valid for *any* choice of infinitely many possible primal and dual variables, as we are solving an iteratively refined sequence of LPs.

- Based on a geometric study of the involved SC constraints, we characterize all facets of involved AGP polytopes that have coefficients in $\{0, 1, 2\}$. We also provide an additional class based on Edge Cover (EC) constraints.
- One class of discussed facets originates from the SC polytope. In that setting, the separation problem is NP-complete. We exploit the geometry to prove that the majority of these facets cannot cut off fractional solutions in an AGP setting (under reasonable assumptions), allowing us to avoid the NP-complete separation problem.
- We demonstrate the practical usefulness of our results with experiments.

## 2 Mathematical Programming Formulation and LP-Based Solution Procedure

Let $G \subseteq P$ be a set of possible guard locations, and $W \subseteq P$ a set of *witnesses*, i. e., points to be guarded. We assume $W \subseteq \mathcal{V}(G)$. In previous work [7], we have presented an LP-based procedure for the original AGP. It can be formulated as an integer linear program $\mathrm{AGP}(G, W)$:

$$\min \sum_{g \in G} x_g \tag{1}$$

$$\text{s.t.} \sum_{g \in G \cap \mathcal{V}(w)} x_g \geq 1 \quad \forall w \in W \tag{2}$$

$$x_g \in \{0, 1\} \qquad \forall g \in G \tag{3}$$

The original problem, $\mathrm{AGP}(P, P)$, has uncountably many variables and constraints and thus cannot be solved directly, especially, because a finite witness set generally cannot ensure coverage of $P$ [4]. So we consider finite $G, W \subset P$. For dual separation and to generate lower bounds, we require the LP relaxation $\mathrm{AGR}(G, W)$ obtained by relaxing the integrality constraint (3):

$$0 \leq x_g \leq 1 \quad \forall g \in G\ . \tag{4}$$

We have shown that $\mathrm{AGR}(P, P)$ can be solved optimally for many problem instances by using finite $G$ and $W$ and solving the primal and dual separation problems, see [7]:

1. Given a solution $(x_g)_{g \in P}$, decide if it is feasible for $\mathrm{AGR}(G, P)$, i. e., completely covers the polygon, or prove infeasibility by presenting an insufficiently covered point $w$. In the latter case a new witness $w$ is added to $W$, and the LP is re-solved. Otherwise, $(x_g)_{g \in P}$ is optimal for $\mathrm{AGR}(G, P)$, and its objective value is an upper bound for $\mathrm{AGR}(P, P)$.
2. Given a solution $(y_w)_{w \in P}$ for the dual LP of $\mathrm{AGR}(G, W)$, decide whether it is feasible for the dual of $\mathrm{AGR}(P, W)$, or prove infeasibility by presenting a violated dual constraint. This coincides with presenting an additional guard point $g$ that will improve the solution. If such a $g$ does not exist, $(y_w)_{w \in P}$ is an optimal dual solution for $\mathrm{AGR}(P, W)$ and the objective value is a lower bound for $\mathrm{AGR}(P, P)$.

If the upper and the lower bound meet, we have an optimal solution of the fractional AGP, $\mathrm{AGR}(P, P)$.

Both separation problems can be solved efficiently using the overlay of the visibility polygons of all points $g \in G$ with $x_g > 0$ (for the primal case) and all $w \in W$ with $y_w > 0$ (for the dual case), which decomposes $P$ into a planar arrangement of bounded complexity.

Our approach may produce fractional solutions as in Fig. 1. In this paper, we use cutting planes to eliminate such fractional solutions. The cuts must remain feasible in all iterations of our algorithm, so feasibility for $\mathrm{AGP}(G, W)$ is insufficient; we require them not to cut off integer solutions of $\mathrm{AGP}(G', P)$ for any $G' \supset G$.

## 3 Set Cover Facets

We transfer known facets [2] of the SC polytope to the AGP polytope, and show that the underlying geometry greatly reduces their impact on the involved AGP polytopes.

### 3.1 A Family of Facets

Consider a finite non-empty subset $\emptyset \subset S \subseteq W$ of witness positions. Every feasible cover of $P$ is a cover of $S$. Analogous to what Balas and Ng [2] did for the SC polytope, we partition $P = J_0 \dot\cup J_1 \dot\cup J_2$, as follows, see Fig. 2: $J_2 := \{g \in P \mid S \subseteq \mathcal{V}(g)\}$, the set of positions that cover all of $S$; $J_0 := \{g \in P \mid \mathcal{V}(g) \cap S = \emptyset\}$, the set of positions that see none of $S$; $J_1 := P \setminus (J_2 \cup J_0)$ the set of positions that cover a non-trivial subset of $S$. Thus, it takes one guard in $J_2$, or at least two guards in $J_1$ to cover $S$. This can be captured in the following inequality:

$$\sum_{g \in J_2 \cap G} 2x_g + \sum_{g \in J_1 \cap G} x_g \geq 2\ . \tag{5}$$

Sufficient coverage of $S$ is necessary for sufficient coverage of $P$, so (5) is valid for any feasible solution of $\mathrm{AGP}(G, P)$. However, covering $S$ may require more than two guards in $J_1$, so (5) does not always provide a supporting hyperplane of $\mathrm{conv}(\mathrm{AGP}(G, W))$. For $|S| \leq 2$, the inequality never cuts off any point of $\mathrm{AGR}(G, W)$; hence, we only consider the case $|S| \geq 3$.

In order to show when Inequality (5) defines a facet of $\mathrm{conv}(\mathrm{AGP}(G, W))$, we apply a result of [2] to the

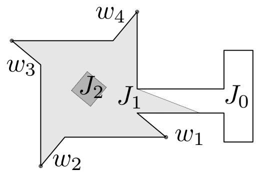


Figure 2: Witness selection $S = \{w_1, w_2, w_3, w_4\}$ and resulting partition $P = J_0 \dot\cup J_1 \dot\cup J_2$.

AGP setting. All proofs in this paper are omitted due to space limitations.

**Lemma 1** *Let $P$ be a polygon and $G, W \subset P$ finite sets of guard and witness positions. Then* $\operatorname{conv}(\mathrm{AGP}(G, W))$ *is full-dimensional, if and only if* $|\mathcal{V}(w) \cap G| \geq 2\ \forall w \in W$.

We require some terminology adapted from [2]. Two guards $g_1, g_2 \in J_1$ form a *2-cover* of $S$, if $S \subseteq \mathcal{V}(\{g_1, g_2\})$. The *2-cover graph* is the graph with nodes in $J_1 \cap G$ and an edge between $g_1$ and $g_2$ iff $g_1, g_2$ are a 2-cover. Finally, we define $T(g) := \{w \in \mathcal{V}(g) \cap W \mid \mathcal{V}(w) \cap G \cap (J_0 \setminus \{g\}) = \emptyset\}$.

**Theorem 2** *Given a polygon $P$ and finite $G, W \subset P$, let* $\operatorname{conv}(\mathrm{AGP}(G, W))$ *be full-dimensional, and let $S$ be maximal, i. e., there is no $w \in W \setminus S$ with $\mathcal{V}(w) \subseteq \mathcal{V}(S)$. Then Inequality* (5) *is facet-defining for* $\operatorname{conv}(\mathrm{AGP}(G, W))$, *if and only if:*

1. *Every 2-cover graph component has an odd cycle.*
2. *$\forall g \in J_0 \cap G$ with $T(g) \neq \emptyset$ there exists either*
   (a) *some $g' \in J_2 \cap G$ such that $T(g) \subseteq \mathcal{V}(g')$, or*
   (b) *$g', g'' \in J_1 \cap G$ with $T(g) \cup S \subseteq \mathcal{V}(g') \cup \mathcal{V}(g'')$.*

## 3.2 Geometric Properties.

For any size $|S|$, there are SC instances where the general, abstract variant of Inequality (5) actually cuts off fractional solutions [2]. In this section, we show that for the AGP, only a very reduced number of these actually occur.

**Lemma 3** *Let $P$ be a polygon and $G, W \subset P$ be finite. Assume $\emptyset \subset S \subseteq W$ is minimal for $G$, i. e., there is no proper subset $R \subsetneq S$ inducing the same Inequality* (5) *as $S$. Then the LP coefficient matrix of* $\mathrm{AGP}(G, S)$ *contains a permutation of the full circulant of order $k = |S|$, which is defined as*

$$C_k^{k-1} := \begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & 0 \end{pmatrix} \in \{0,1\}^{k \times k} . \qquad (6)$$

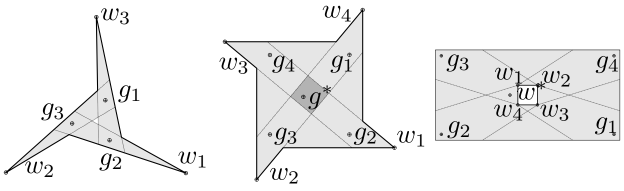


Figure 3: $P_3^2$ (left) and two attempts for $P_4^3$ (middle and right). In the left case, Ineq. (5) enforces using two guards instead of three $\frac{1}{2}$-guards. The attempts for $P_4^3$ are star-shaped (middle) or invalid, as $x_{g_1} = \ldots = x_{g_4} = \frac{1}{3}$ is infeasible (right, at $w^*$).

This motivates a formal definition of polygons that correspond to $C_k^{k-1}$.

**Definition 1 (Full Circulant Polygon)** *A polygon $P = P_k^{k-1}$ along with $G = \{g_1, \ldots, g_k\} \subset P$ and $W = \{w_1, \ldots, w_k\} \subset P$ for $k \geq 3$ is called* Full Circulant Polygon, *if*

$$\mathcal{V}(g_i) \cap W = W \setminus \{w_i\} \quad \forall 1 \leq i \leq k, \text{ and} \qquad (7)$$
$$|\mathcal{V}(w) \cap G| \geq k - 1 \quad \forall w \in P . \qquad (8)$$

Note that $P_k^{k-1}$ is defined such that $C_k^{k-1}$ completely describes the visibility relations between $G$ and $W$. This implies that the optimal solution of $\mathrm{AGR}(G, W)$ is $\frac{1}{k-1}\mathbb{1}$ (i.e., assigns a value of $\frac{1}{k-1}$ to each $g \in G$), with total cost $\frac{k}{k-1}$. It is feasible for $\mathrm{AGR}(G, P_k^{k-1})$ by Property (8).

Figure 3 captures construction attempts for models of $C_k^{k-1}$. $P_3^2$ exists, but for $k \geq 4$, the polygons are either star-shaped or not full circulant. If they are star-shaped, the optimal solution is to place one guard within the kernel. If they are not full circulant polygons, the optimal solution of $\mathrm{AGR}(G, W)$ is infeasible for $\mathrm{AGR}(G, P_k^{k-1})$. In this case, the current fractional solution is intermittent, i.e., it will be cut off by the algorithm by introducing new witnesses. Both cases eliminate the need for a cutting plane. In the following we argue that $P_k^{k-1}$ is indeed star-shaped for $k \geq 4$, allowing us to avoid the NP-complete separation problem of finding all permutations of all full circulants in the matrix of $\mathrm{AGP}(G, W)$ by reducing our search to $k = 3$.

The first step is Lemma 4, which restricts the possible structure of $P_k^{k-1}$. It provides the basis for our main theorem.

**Lemma 4** *For $k \geq 4$, every full circulant polygon is simple, i.e., it has no holes. This is not true for $k < 4$.*

**Theorem 5** *For $k \geq 4$, every full circulant polygon is star-shaped.*

By Theorem 5, no cuts of type (5) are necessary to cut off fractional solutions for a full circulant polygon $P_k^{k-1}$ with $k \geq 4$. It is still possible to embed

$P_k^{k-1}$ in larger polygons, where these cuts play a role. However, our experiments, see Section 5, suggest that these cases rarely occur in practice.

### 3.3 All Facets with Coefficients in $\{0, 1, 2\}$

Balas and Ng [2] identified all SC facets with coefficients in $\{0, 1, 2\}$. The only nontrivial facet class corresponds to Ineq. (5). As for finite $G, W \subset P$, $\mathrm{AGP}(G, W)$ is an SC instance, we have identified all nontrivial AGP facets with coefficients in $\{0, 1, 2\}$.

## 4 Edge Cover Facets

Solving $\mathrm{AGR}(G, W)$ for finite $G, W \subset P$ in which no guard sees more than two witnesses is equivalent to solving a fractional edge cover instance on the following graph: The nodes correspond to the witnesses, each guard is represented by an edge or a loop. This is the case in the 5-gonal corridor in the right part of Fig. 1. As outlined in [7], the inequality

$$\sum_{g \in \mathcal{V}(W) \cap G} x_g \geq \left\lceil \frac{k}{2} \right\rceil \tag{9}$$

can cut off such fractional solutions, provided $|W|$ is odd. It is a valid constraint if no guard exists that sees more than two witnesses in $W$. Inequality (9) is facet-defining for $\mathrm{conv}(\mathrm{AGP}(G, W))$ under some conditions that we leave out due to space restrictions.

## 5 Computational Experience

A variety of experiments on benchmark polygons show the usefulness of our cutting planes. We employed the same four classes of test polygons as in [7] with approximately 60, 200, 500 and 1000 vertices.

The experiments keep track of the gap between the smallest (integer) upper bound and largest (fractional) lower bound. They were conducted on 3.0 GHz Intel dual core PCs with 2 GB of memory. Our algorithms used CGAL 4.0 and CPLEX 12.1.

Due to space restrictions, we only present the relative gap over time for so-called *von Koch* polygons with 1000 vertices. Fig. 4 shows the distribution of relative gaps over time for different combinations of cutting planes in the third quartile.

Clearly, using no separation yields the largest gaps. The EC cuts are successful in reducing the gap and the SC cuts for $k = 3$ close the gaps faster than all other cut separators. The SC cuts for $k \in \{3, 4\}$ are weaker than those limited to $k = 3$, supporting the practical relevance of our arguments from Section 3: The separation for $k = 4$ takes time, but has no benefits for the gap. Combining SC cuts for $k = 3$ and EC cuts is slightly slower than SC cuts for $k = 3$ only due to an overlap of the facet classes.

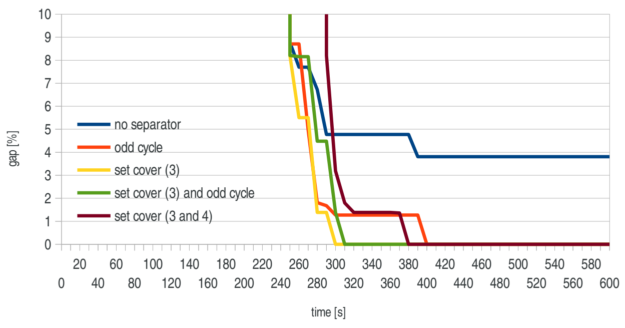


Figure 4: Separator performance.

## 6 Conclusion

In this paper, we have shown how we can exploit both geometric properties and polyhedral methods of mathematical programming to solve a classical and natural, but highly challenging problem from computational geometry. We have shown that an NP-complete separation problem for the SC case can mostly be avoided in the AGP scenario by considering its underlying geometric structure.

## Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG), contract KR 3133/1-1 (Kunst!).

## References

[1] Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. *Int. J. Comput. Geometry Appl.*, 20(5):601–630, 2010.

[2] E. Balas and M. Ng. On the set covering polytope: I. all the facets with coefficients in $\{0, 1, 2\}$. *Math. Program.*, 43(1):57–69, January 1989.

[3] V. Chvátal. A combinatorial theorem in plane geometry. *J. Comb. Theo., Series B*, 18(1):39 – 41, 1975.

[4] K.-Y. Chwa, B.-C. Jo, C. Knauer, E. Moet, R. van Oostrum, and C.-S. Shin. Guarding art galleries by guarding witnesses. *Int. J. Comput. Geometry Appl.*, 16:205–226, 2006.

[5] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *Int. Trans. Op. Res.*, 18(4):425–448, 2011.

[6] S. Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Theo., Series B*, 24(3):374, 1978.

[7] A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt. Exact solutions and bounds for general art gallery problems. *J. Exp. Alg.*, 17, 2012.

[8] D.-T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inf. Theor.*, 32(2):276–282, 1986.

[9] J. O'Rourke. *Art Gallery Theorems and Algorithms.* International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

# The Minimum Guarding Tree Problem

Adrian Dumitrescu[*] Joseph S. B. Mitchell[†] Paweł Żyliński[‡]

**Abstract**

We provide a simple alternative NP-hardness proof of the problem of finding a guarding tree of minimum length for a set of orthogonal line segments in the plane (N. Xu, Complexity of minimum corridor guarding problems, *Inform. Process. Lett.* 112(17-18), 691-696). Then, we present two approximation algorithms of factors 2 and 3.98, respectively, for computing a minimum guarding tree for a subset of a set of $n$ arbitrary non-parallel lines in the plane; their running times are $O(n^8)$ and $O(n^6 \log n)$, respectively. Finally, we show that this problem is NP-hard for lines in 3-space.

## 1 Introduction

A connected set of lines or line segments can model of the corridors in a building. Consider a mobile guard that patrols the corridors; the guard has unlimited visibility, so a line/segment can be "seen" in both directions from any point on it. The problem is to find a shortest guarding "network" for the guard, e.g., a path, a tree, or a closed route, such that when traversing all the edges of the network (moving only within the network), each line/segment is visited at least once.

In this paper we study the variant of the problem in which the guarding network is restricted to form a tree; obviously, a minimum length guarding network is always a tree. Formally, the problem is defined as follows. Let $\mathcal{X}$ be a connected set of lines or line segments in the plane, i.e., there exists a path $\xi \subset \cup_{l \in \mathcal{X}} l$ from any point $p \in l$ to any other point $p' \in l'$, for any $l, l' \in \mathcal{X}$. (Observe that a set of lines $\mathcal{X}$ is connected if and only if not all lines are parallel, i.e., there exist two non-parallel lines in $\mathcal{X}$). Given a subset $\bar{\mathcal{X}} \subseteq \mathcal{X}$, a *guarding tree* $T = (V(T), E(T))$ for $\bar{\mathcal{X}}$ is a tree, with vertices (points) and edges (segments) contained in the union of the elements in $\mathcal{X}$, such that if the mobile guard runs on the edges of $T$, all elements in $\bar{\mathcal{X}}$ are visited by the guard. The *weight* $|T|$ of a guarding tree $T$ is the sum of its weights of edges, where the weight of an edge is its Euclidean length.

**The minimum guarding tree problem for lines (MGTL)**

> Given a subset $\bar{\mathcal{L}}$ of a set $\mathcal{L}$ of non-parallel lines in the plane, find a minimum-weight guarding tree for $\bar{\mathcal{L}}$.

**The minimum guarding tree problem for segments (MGTS)**

> Given a subset $\bar{\mathcal{S}}$ of a connected set $\mathcal{S}$ of line segments in the plane, find a minimum-weight guarding tree for $\bar{\mathcal{S}}$.

Previously, MGTS appears to have been only considered for arrangements of axis-aligned segments, so-called *grids*: Xu and Brass [15, 16] prove the NP-hardness of MGTS by a reduction from the connected vertex cover problem in planar graphs with maximum degree four.

**Our results.** We first show that MGTS is NP-hard, and in view of [15, 16], we reprove this result in a simpler way. Next, we propose two approximation algorithms for MGTL. Their approximation ratios are 2 and 3.98, respectively, and the running times are $O(n^8)$ and $O(n^6 \log n)$, respectively. Finally, we show that the problem of computing a minimum-weight guarding tree for a set of lines in 3-space is NP-hard even for orthogonal lines.

**Related work.** The MGT problem for lines or segments in the plane is a variant of minimum corridor connection problem [2, 4, 8]. A wider perspective locates the problem as a variant of the art gallery problem for segments; see [5, 7, 10, 12, 13, 14], to mention just a few. Finally, the MGT problem for lines/segments is closely related to the group Steiner tree problem [11].

**Notation.** Let $V(\mathcal{A}(\mathcal{L}))$ denote the set of vertices of the arrangement $\mathcal{A}(\mathcal{L})$ formed by lines in a given set $\mathcal{L}$. Let $G(\mathcal{L})$ be the weighted planar graph with vertex set $V(\mathcal{A}(\mathcal{L}))$ whose edges connect successive vertices on the elements in $\mathcal{L}$; the weight of an edge is the Euclidean distance between the corresponding vertices along the connecting line. For $s, t \in V(\mathcal{A}(\mathcal{L}))$,

[*]Department of Computer Science, University of Wisconsin–Milwaukee, USA, `dumitres@uwm.edu`. Partially supported by NSF grant DMS-1001667.

[†]Department of Applied Mathematics and Statistics, State University of NY at Stony Brook, USA, `jsbm@ams.sunysb.edu`. Partially supported by NSF (CCF-1018388) and by Metron Aviation (subcontract from NASA Ames

[‡]Institute of Informatics, University of Gdańsk, Poland, `zylinski@ug.edu.pl`.

let $\pi(s,t)$ denote a shortest path connecting $s$ and $t$ in $G(\mathcal{L})$, and let $|\pi(s,t)|$ denote its length. Finally, for a subset $\bar{\mathcal{L}} \subseteq \mathcal{L}$, let $T_{\text{opt}}(\bar{\mathcal{L}})$ denote an optimal guarding tree for $\bar{\mathcal{L}}$, and for two points $p$ and $q$ in the plane, let $|pq|$ denote the Euclidean length of the segment $pq$.

## 2 NP-hardness of MGTS

In this section we present a simple alternative NP-hardness proof of MGTS for orthogonal line segments. We relate MGTS to the rectilinear Steiner tree problem [6], which is known to be NP-hard; its decision version is the following.

**The rectilinear Steiner tree problem**

Given a set $S$ of $n$ lattice points in the plane, and a positive integer $m$, does there exists a rectilinear Steiner tree for $S$ of total weight at most $m$?

**Theorem 1** [15, 16] *MGTS is NP-hard. The problem remains so even for orthogonal segments.*

Xu and Brass proved the above result by a reduction from the connected vertex cover problem in planar graphs with maximum degree four [15, 16]. Our simpler alternative proof is as follows.

**Proof.** Let $P$ be a set of $n$ lattice points, and let $R$ be the minimal axis-parallel rectangle containing $P$. Consider the arrangement $H(P)$ of lines induced by $P$, the so called Hanan grid [9], and for each point in $p \in P$, add to a set $\mathcal{S}$, initially empty, the maximal (w.r.t. $R$) horizontal and vertical segments incident to $p$; see Fig. 1 for an illustration. Next, for each point in $p \in P$, add to $\mathcal{S}$ a short horizontal segment $s(p)$ of length $1/(80n)$ at distance $1/(80n)$ from $p$ in $R$. Observe that the segment $s(p)$ can be only visited from the grid segment incident to $p$, moreover, from the vicinity of $p$. Since the Hanan grid $H(P)$ is

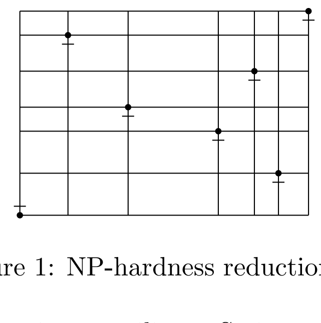


Figure 1: NP-hardness reduction [5].

known to contain a rectilinear Steiner tree for $P$ [9], the reduction, and thus the NP-hardness, follows by the following claim, which is easy to verify.

**Claim.** *For a positive integer $m$, there exists a rectilinear Steiner tree for $P$ of length at most $m$ if and only if there exists a guarding tree for $\mathcal{S}$ of weight at most $m + 0.1$.* □

## 3 MGTL: A slower 2-approximation

The computational complexity of MGTL remains unsettled. In this section we obtain the following result.

**Theorem 2** *There exists a ratio 2-approximation algorithm for MGTL, running in $O(n^8)$ time.*

**Proof.** In [5], the authors provide an algorithm that computes the shortest guarding route (i.e., closed tour) $\mathcal{R}_{\text{opt}}$ for a given subset $\bar{\mathcal{L}}$ of a set $\mathcal{L}$ of $n$ non-parallel lines in the plane in $O(n^8)$ time. Consider a vertex $x_1 \in V(\mathcal{A}(\mathcal{L}))$ that lies on $\mathcal{R}_{\text{opt}}$, and let $x_1, x_2, \ldots, x_k \in V(\mathcal{A}(\mathcal{L}))$ be the consecutive vertices along $\mathcal{R}_{\text{opt}}$ such that at each vertex $x_i$, $i = 1, \ldots, k$, at least one new line in $\bar{\mathcal{L}}$ becomes visited; vertices $x_1, \ldots, x_k$ can be determined in $O(n^3)$ time (notice $k < n$). Let $T$ be any Steiner tree for $\{x_1, x_2, \ldots, x_k\}$ in the graph $\Pi \subseteq G(\mathcal{L})$ that results from the union of all the edges of the shortest paths $\pi(x_i, x_{i+1})$, $i = 1, \ldots, k-1$; clearly, $T$ can be computed in total $O(n^5)$ time [3], where $n = |\mathcal{L}|$.

Now, by the construction, $T$ is a guarding tree for $\bar{\mathcal{L}}$, and $|T| \le |\mathcal{R}_{\text{opt}}|$. Since doubling the edges of an optimal guarding tree $T_{\text{opt}}$ for $\bar{\mathcal{L}}$ results in a guarding route for $\bar{\mathcal{L}}$, we obtain $|\mathcal{R}_{\text{opt}}| \le 2 \cdot |T_{\text{opt}}|$, and thus $|T| \le 2 \cdot |T_{\text{opt}}|$, as required. □

## 4 MGTL: A slightly faster 3.98-approximation

The running time of the algorithm from Section 3 is quite high. In this section, we present a slightly faster algorithm, with running time $O(n^6 \log n)$, and approximation ratio 3.98.

**Guessing key elements of an optimal guarding tree.** Assume w.l.o.g. that no line in the input set $\mathcal{L}$ is horizontal or vertical, or is of the positive/negative degree 60 slope, and there are no two vertices in $V(\mathcal{A}(\mathcal{L}))$ that lie on a horizontal line or a line with the positive/negative degree 60 slope.

Consider now a minimum guarding tree $T_{\text{opt}} = T_{\text{opt}}(\bar{\mathcal{L}})$ for $\bar{\mathcal{L}}$. Let $E$ be a minimal equilateral triangle containing $T_{\text{opt}}$, with a horizontal lower side. Since $T_{\text{opt}}$ visits all lines in $\bar{\mathcal{L}}$, $E$ intersects all lines in $\bar{\mathcal{L}}$. Let $A = \{a_1, a_2, a_3\}$ be the set of (at most three) vertices of $\mathcal{A}(\mathcal{L})$ that determine $E$, labeled counterclockwise starting with the lower side of $E$; we have $2 \le |A| \le 3$. In particular, $T_{\text{opt}}$ visits $A$. Let $E = r_1r_2r_3$ be labeled counterclockwise starting with its lower right corner (see Fig. 2). Suppose we *guess* $A$, hence $E$ is determined, and then we *mark* all lines in $\bar{\mathcal{L}}$ that intersect the triangle $a_1a_2a_3$. Observe that any tree or route (not necessarily contained in $E$) visiting the points in $A$ visits the marked lines as well.

Our algorithm will generate a tree $T$ (not necessarily contained in $E$) that visits $A$, thus all marked

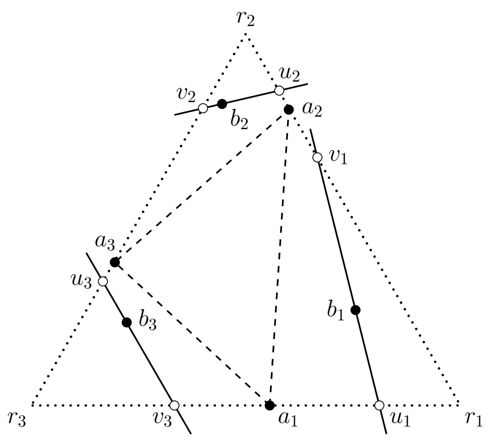


Figure 2: Guessing key elements of an optimal tour. Vertices of the arrangement $\mathcal{A}(\mathcal{L})$ are drawn as filled circles; other points are drawn as empty circles. Lines in $\mathcal{L}$ are drawn solid.

lines as well as all the unmarked lines. We will ensure that $|T| \leq \frac{5+4\sqrt{3}}{3} \cdot |T_{\rm opt}|$. Consequently, $T$ will be a valid guarding tree for $\bar{\mathcal{L}}$ that gives a ratio 3.98 approximation of the optimal solution.

Consider first the lower right *corner* (vertex) $r_1$ of $E$; the other corners of $E$ are handled in a similar way. Consider the set $\bar{\mathcal{L}}_1 \subset \bar{\mathcal{L}}$ of unmarked lines that intersect the *corner triangle* $\Delta a_1 r_1 a_2$. If $\bar{\mathcal{L}}_1$ is empty continue with the next corner triangle of $E$; so assume that $\bar{\mathcal{L}}_1$ is not empty. Identify an unmarked line $\ell_1 \in \bar{\mathcal{L}}_1$ such that the triangle $\Delta u_1 a_1 v_1$ is minimal with respect to inclusion (i.e., no other such triangle is contained in it), where $u_1$ and $v_1$ are the intersection points of $\ell_1$ with the boundary of $E$ (see again Fig. 2.) Both the line $\ell_1$ and the segment $u_1 v_1 = \ell_1 \cap E$ are called *extremal*.

Obviously, each line in $\bar{\mathcal{L}}_1$ is visited by $T_{\rm opt}$ at some vertex on that line contained in $\Delta a_1 r_1 a_2$. Assume that the extremal line $\ell_1 \in \bar{\mathcal{L}}_1$ is visited by $T_{\rm opt}$ at vertex $b_1 \in \ell_1 \cap E$. Let $B = \{b_1, b_2, b_3\}$ be the set of (at most three) vertices, where $T_{\rm opt}$ visits the extremal lines; we have $0 \leq |B| \leq 3$.

**Lemma 3** *The following inequality holds:*

$$|\pi(a_1,b_1)| + |\pi(b_1,a_2)| + |\pi(a_2,b_2)| + |\pi(b_2,a_3)| + |\pi(a_3,b_3)| + |\pi(b_3,a_1)| \leq 2 \cdot |T_{\rm opt}|.$$

**Lemma 4** *The following inequality holds:*

$$|u_1v_1| + |u_2v_2| + |u_3v_3| \leq \frac{2\sqrt{3}}{3}\big(|a_1a_2| + |a_2a_3| + |a_3a_1|\big).$$

Recall, $u_i$ and $v_i$ are the intersection points of extremal line $\ell_i$ with the boundary of $E$, $i = 1, 2, 3$; due to space limits, the proofs of the above lemmas are omitted.

**Approximation algorithm.** For each equilateral triangle $E'$ determined by (at most) 3 vertices $A' \subset V(\mathcal{A}(\mathcal{L}))$, with the horizontal lower side, do: Check whether $E'$ intersects all lines in $\bar{\mathcal{L}}$. If it does not, move to the next triangle ($E'$ cannot be the minimal equilateral triangle $E$, with the horizontal lower side, containing a minimum guarding tree for $\bar{\mathcal{L}}$). Otherwise, compute the tree $T = T(E')$ (described below), which intersects all lines. Output the shortest tree among these.

Determine the (at most 3) extremal lines $\ell_1, \ell_2, \ell_3$ corresponding to the three corner triangles of $E'$, as computed in the preprocessing step (details omitted). Consider the line $\ell_1$; the same computation is done for the other two lines. Retrieve (again from the preprocessing) the vertex $w_1 \in \ell_1 \cap E'$ that minimizes the sum $|\pi(a_1, w_1)| + |\pi(w_1, a_2)|$. Similarly, retrieve $w_i \in \ell_i \cap E'$, for $i = 2, 3$. If for some $i \in \{1, 2, 3\}$, $\ell_i$ exists but $w_i$ does not exist, abandon this triangle $E'$, and skip to the next one. (Notice that if $E'$ corresponds to the equilateral triangle $E$ containing a minimum guarding tree for $\bar{\mathcal{L}}$ and $\ell_i$ exists, then $b_i \in \ell_i \cap E'$ exists, hence also $w_i \in \ell_i \cap E'$ exists.)

Consider (at most) six paths $\pi_1 = \pi(a_1, w_1)$, $\pi_2 = \pi(w_1, a_2)$, $\pi_3 = \pi(a_2, w_2)$, $\pi_4 = \pi(w_2, a_3)$, $\pi_5 = \pi(a_3, w_3)$, $\pi_6 = \pi(w_3, a_1)$, and w.l.o.g. assume that $|\pi_6| = \max_{i \in \{1,\ldots,6\}} |\pi_i|$. Now, to the initially empty tree $T'$, add all the edges of (at most 5) paths $\pi_1, \ldots, \pi_5$; but do not add multiple edges/segments. (Notice that the temporary tree $T'$ is a Steiner tree for $A \cup \{w_1, w_2, w_3\}$, and we do not require that the computed guarding tree $T$ lies inside $E$.) Next, to each vertex $w_i$, append (at most six in total) edges $u_i w_i$ and $w_i v_i$, $i = 1, \ldots, 3$, if not added yet, thus obtaining the final tree $T$.

**Correctness and approximation ratio.** First, observe that $T$ is connected, and since $T$ is a Steiner tree for $A'$, it intersects all marked lines. Next, each unmarked line intersects a corner triangle $\Delta a_i r_i a_{i+1}$, for some $i = 1, 2, 3$, with $a_4 = a_1$. Consider an unmarked line $\ell$ intersecting the triangle $\Delta a_1 r_1 a_2$. Since $\ell_1$ is extremal, $\ell$ either (i) intersects $u_1 v_1$ or (ii) intersects the segment $a_1 u_1$ on the lower side of $E'$ and the segment $v_1 a_2$ on the right side of $E'$. In case (i), $\ell$ intersects one of the edges $v_1 w_1$ and $w_1 u_1$. In case (ii), $\ell$ separates $w_1$ from $a_1$ and from $a_2$, and thus the path $\pi_T(a_1, w_1)$ (and $\pi_T(a_2, w_1)$) in $T$, connecting vertices $a_1$ and $w_1$ (resp. $a_2$ and $w_1$) in $T$, intersect $\ell$. Hence in either of the two cases $\ell$ intersects $T$. Consequently, $T$ intersects all unmarked lines as well, and thus $T$ is a valid guarding tree for $\bar{\mathcal{L}}$.

It remains to show that for the shortest tree among those determined for any choice of $E'$, its weight is at most $3.98 \cdot |T_{\rm opt}|$. Consider the guarding tree $T$ computed while handling $E' = E$. Since $w_1 \in \ell_1$ minimizes the sum $|\pi(a_1, w_1)| + |\pi(w_1, a_2)|$, we have

$$|\pi(a_1, w_1)| + |\pi(w_1, a_2)| \leq |\pi(a_1, b_1)| + |\pi(b_1, a_2)|.$$

By adding the analogous inequalities for all three corners of $E' = E$ and using Lemma 3, we obtain

$$\textstyle\sum_{i=1}^{6} |\pi_i| \leq 2 \cdot |T_{\text{opt}}|.$$

Next, since $|\pi_6| = \max_{i\in\{1,\dots,6\}} |\pi_i|$, we obtain

$$\textstyle\sum_{i=1}^{5} |\pi_i| \leq \frac{5}{3} \cdot |T_{\text{opt}}|.$$

Consequently, by combining the triangle inequality, Lemma 3 and 4, we obtain

$$\begin{aligned} |T| &\leq \tfrac{5}{3} \cdot |T_{\text{opt}}| + |u_1v_1| + |u_2v_2| + |u_3v_3| \\ &\leq \tfrac{5+4\sqrt{3}}{3} \cdot |T_{\text{opt}}| = 3.97606.... \cdot |T_{\text{opt}}|. \end{aligned}$$

**Running time.** The running time of the algorithm is determined by the number of triples of vertices in $V(\mathcal{A}(\mathcal{L}))$ that constitute the set $A'$; there are at most $\binom{n}{2}^3 = O(n^6)$ such triples. Each such triple is handled in $O(\log n)$ time using the information gathered during preprocessing (details omitted). Therefore, the total running time is $O(n^6 \log n)$, and we have the following result.

**Theorem 5** *There exists a ratio* 3.98*-approximation algorithm for MGTL, running in* $O(n^6 \log n)$ *time.*

## 5 Concluding remarks

We conclude with a few open problems.

(i) What is the complexity of MGTL?

(ii) Can the running time of our algorithms for MGTL be substantially improved?

(iii) What is the complexity of the MGT problem for an arrangement of planes in 3-space?

(iv) What is the complexity of the minimum guarding path problem for an arrangement of lines in the plane? (Here we are interested in finding the shortest path that visits all lines.)

Similarly, as in the case of line segments in the plane, we observe that our (simple) NP-hardness proof of the problem of computing a shortest guarding route for a set of lines in 3-space [5] carries over, without any modification, to the problem of computing a minimum-weight guarding tree for a set of lines in 3-space, thus resulting in the following corollary.

**Corollary 6** *The MGT problem for lines in* 3*-space is NP-hard.*

## Acknowledgment

The authors thank Marc Benkert, Étienne Schramm, and Alexander Wolff for valuable remarks and interesting conversations on the topic.

## References

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry*, Springer Verlag, 3rd edition, 2010.

[2] H. L. Bodlaender, C. Feremans, A. Grigoriev, E. Penninkx, R. Sitters, T. Wolle, On the minimum corridor connection problem and other generalized geometric problems, *Comput. Geom.*, 42(9), 939–951 (2009).

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, third edition, MIT Press, Cambridge, 2009.

[4] E. D. Demaine, J. O'Rourke, Open problems from CCCG 2000, *Proc. eedings of 13th CCCG*, 185–187 (2001).

[5] A. Dumitrescu, J. S. B. Mitchell, and P. Żyliński, Watchman routes for lines and segments, *Proc of 13th SWAT*, *Lect. Notes Comp. Sci.* 7357, 36–47 (2012).

[6] M. R. Garey and D. S. Johnson, The rectilinear Steiner tree problem in NP complete, *SIAM J. Appl. Math.* 32, 826–834 (1977).

[7] L. P. Gewali and S. Ntafos, Covering grids and orthogonal polygons with periscope guards, *Computat. Geom.* 2(6), 309–334 (1993).

[8] A. Gonzalez-Gutierrez, T. F. Gonzalez, Approximation algorithms for the minimum-length corridor and related problems, *Proc. the 19th CCCG*, 253–256 (2007).

[9] M. Hanan, On Steiner's problem with rectilinear distance, *SIAM J. Appl. Math.* 14, 255–265 (1966).

[10] A. Kosowski, M. Małafiejski, and P. Żyliński, Cooperative mobile guards in grids, *Computat. Geom.* 37(2), 59–71 (2007).

[11] J. S. B. Mitchell, Geometric shortest paths and network optimization, in *Handbook of Computational Geometry* (J.-R. Sack, J. Urrutia, eds.), Elsevier, 633–701 (2000).

[12] S. Ntafos, On gallery watchmen in grids, *Inform. Process. Lett.* 23(2), 99–102 (1986).

[13] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.

[14] C. D. Tóth, Illuminating disjoint line segments in the plane, *Discrete Comp. Geom.* 30(3), 489–505 (2003).

[15] N. Xu, Complexity of minimum corridor guarding problems, *Inform. Process. Lett.* 112(17-18), 691–696 (2012).

[16] N. Xu and P. Brass, On the complexity of guarding problems on orthogonal arrangements, *Abstracts of the 20th FWCG*, #33 (2010).

# Covering Class-3 Orthogonal Polygons with the Minimum Number of $r$-Stars

Leonidas Palios[*][†] Petros Tzimas[*]

## Abstract

We consider the problem of covering simple orthogonal polygons with the minimum number of $r$-stars. The problem has been considered by Worman and Keil [10] who described an $O(n^{17}\text{poly-log}n)$-time algorithm where $n$ is the size of the given polygon.

We consider the above problem on simple class-3 orthogonal polygons; a class-3 orthogonal polygon is defined to have dents along at most 3 different orientations. By taking advantage of geometric properties of these polygons, we are able to provide an $O(n^2)$-time algorithm; this is the first purely geometric algorithm for this problem and it paves the way for obtaining algorithms for the problem on general simple orthogonal polygons that are faster than Worman and Keil's.

## 1 Introduction

The field of Art Gallery problems is a vibrant and large research area in combinatorial and computational geometry [7, 9]. The multitude of variants is in part due to the fact that getting the minimum number of guards to watch a given polygon is NP-complete [1], which stimulated research in restricted types of polygons or with guards possessing different visibility or mobility characteristics.

Guarding problems have been considered on *orthogonal* polygons, i.e., polygons whose edges are either horizontal or vertical; the edges can then be characterized as N-edges, S-edges, E-edges, and W-edges (see Figure 1). Of particular importance are edges whose both endpoints are reflex vertices of the polygon; such edges are called *dents* and as above they are characterized as N-dents, S-dents, E-dents, and W-dents (see Figure 1). Orthogonal polygons can be classified in terms of the types of dents that they contain [2]: a *class-k* orthogonal polygon ($0 \leq k \leq 4$) is defined to have dents along at most $k$ different orientations. Class-2 polygons can be further classified into class-2a where the 2 dent orientations are parallel

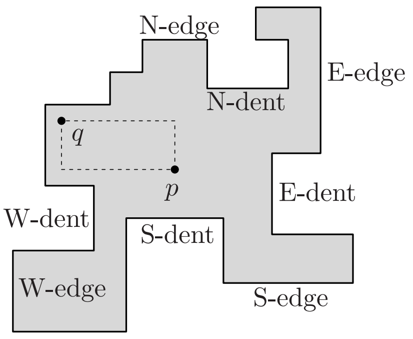


Figure 1: Illustration of the main definitions.

(i.e., N and S, or E and W), and class-2b where the 2 dent orientations are perpendicular to each other.

Essential to a guarding problem is the notion of visibility of the guards used. We consider $r$-visibility: in an orthogonal polygon $P$, two points $p, q$ of $P$ are *$r$-visible* from one another if and only if the axis-parallel rectangle with $p, q$ at opposite corners lies within $P$ (see Figure 1). Then, a polygon $P$ is an *$r$-star* if there exists a point $p$ of $P$ such that every point $q \in P$ is $r$-visible from $p$; an $r$-star is a convex orthogonal polygon such that there exists a perpendicular line intersecting both its topmost and bottommost edge and a horizontal line intersecting both its leftmost and rightmost edge. (We note that in orthogonal polygons we may also have *$s$-visibility* and *$s$-stars*.)

Clearly, the problem of determining a minimum set of $r$-visibility guards to watch a simple polygon is equivalent to determining a minimum covering of the polygon by $r$-stars. A *covering* of a polygon $P$ into a set $S$ of pieces (or subpolygons or components) requires that the union of the pieces in $S$ is equal to $P$. If additionally the pieces are required to be mutually disjoint (except along boundaries), then we have a *partition*. Obviously, a partition of a polygon also forms a covering of the polygon; thus, coverings are better than partitions in terms of the number of pieces. On the other hand, covering problems prove to be harder than their corresponding partition problems and there are cases where the former are NP-hard whereas the latter admit polynomial solutions.

Covering by $r$-stars has been investigated early enough. Keil [4] described an $O(n^2)$-time algorithm to cover a class-2a orthogonal polygon by $r$-stars. Cul-

[*]Computer Science Department, University of Ioannina, GR-45110 Ioannina, Greece, {palios, ptzimas}@cs.uoi.gr

[†]This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the national Strategic Reference Framework (NSRF) - Research Funding Program: THALIS UOA (MIS 375891).

berson and Reckhow [2] showed that Keil's algorithm is worst-case optimal if the $r$-stars need to be explicitly reported and presented an $O(n)$-time algorithm to count the number of $r$-stars needed. The time complexity of covering a class-2a orthogonal polygon with $r$-stars was improved by Gewali, Keil, and Ntafos [3] who gave an $O(n)$-time algorithm to report the locations of a minimum-cardinality set of guards. An improvement of this algorithm is presented in [5]. The problem of covering general orthogonal polygons with $r$-stars was addressed by Worman and Keil who took advantage of the graph-theoretic approach used in [6] (for $s$-star coverings) to describe an $O(n^{17}\text{poly-log}n)$-time algorithm [10].

In this paper, we study the $r$-star covering problem for class-3 orthogonal polygons. We take advantage of geometric properties of these polygons and we describe an $O(n^2)$-time plane-sweep algorithm to report the locations of a minimum-cardinality set of $r$-visibility guards to watch the entire polygon. This is the first purely geometric algorithm for this problem and it paves the way for obtaining algorithms for the problem on general simple orthogonal polygons that are faster than Worman and Keil's.

## 2 Theoretical Framework

We consider simple orthogonal polygons; so, in the following, we will omit the adjective simple.

Consider an orthogonal polygon $P$ that does not have N-dents. For any horizontal line $L$ intersecting $P$, the portion of $P$ on and below $L$ consists of a number of *disjoint* parts of $P$ each intersecting $L$ in a single line segment (due to the lack of N-dents); for convenience, we call each such part of $P$ a *trouser*. Next, we give extensions of the notions of "grid segment" and "level" used in [3]: a *grid segment* of $P$ or a trouser $T$ is a maximal (closed) horizontal line segment in $P$ or $T$; the *level* of a point or a horizontal line segment (which may be a grid segment or a horizontal edge) is its $y$-coordinate. We also use the notion of orthogonal projection in an orthogonal polygon $P$ given in [5]: the *orthogonal projection* $o(s)$ of a horizontal line segment $s$ at level $\ell$ in $P$ onto the grid segment $s'$ at level $\ell' \geq \ell$ is the maximal subsegment of $s'$ such that for each point $a$ of $o(s)$ there exists a vertical line segment in $P$ that goes through $a$ and intersects $s$. Finally, for a horizontal line segment $s$ (edge or grid segment) we define its *$x$-range* to be the set of $x$-coordinates of the points of $s$. We note that although a polygon is considered a closed set, we consider edges to be open sets (i.e., they do not include their endpoints) and thus their $x$-ranges are open sets as well.

The following lemma provides three very important properties of class-3 orthogonal polygons.

**Lemma 2.1** *Let $P$ be a class-3 orthogonal polygon and assume that $P$ has no N-dents. Then:*

*(i) The polygon $P$ has a single topmost edge.*

*(ii) Consider sweeping $P$ from bottom to top. Each edge encountered other than the bottommost edge of each trouser is incident on the boundary of the swept polygon.*

*(iii) Let $T$ be a trouser when $P$ is intersected by a horizontal line at level $\ell$, and let $s_1$ and $s_2$ be grid segments of $T$ at levels $\ell_1$ and $\ell_2$, respectively, such that $\ell_1 < \ell_2 \leq \ell$ and there exists a vertical line segment in $T$ intersecting both $s_1$ and $s_2$. Then, the orthogonal projection of $s_1$ onto $\ell$ is a subset of the orthogonal projection of $s_2$ onto $\ell$.*

## 3 The Algorithm

Our algorithm applies plane-sweeping from bottom to top (as do the algorithms in [3, 5]) assuming that the given class-3 polygon has no N-dents (thus we can take advantage of Lemma 2.1); the sweep-line stops at each S-edge placing a guard-request (Section 3.2), and at each N-edge where it may place a guard or a guard request, and it clips information on guards and guard-requests (Sections 3.1-3.3). The algorithm reports the locations of a minimum-cardinality set of $r$-visibility guards to watch the entire input polygon.

### 3.1 Maintaining and Processing Guards

We follow the convention in the algorithms in [3] and [5], according to which the guards are placed at the leftmost possible point of the highest possible level; thus each guard is located at the level of a N-edge. In order to find the appropriate locations of the guards, with each guard we maintain:

- a *location-range*, or *loc-range* for short, which is the range of $x$-coordinates of the points at which the guard can currently be placed so that it watches a S-edge (if assigned; see Section 3.3) and as much of the unseen polygon as possible;
- a *visibility-range*, or *vis-range* for short, which is the range of $x$-coordinates of the points *above* the current position of the sweep-line that are visible to the guard.

Since there are no N-dents, each of these ranges is a *single interval* of $x$-coordinates, and because we place guards so that they can see as much of the polygon above them as possible, it always holds that the loc-range of a guard is a subset of its vis-range.

For a guard $g$ to be placed at a grid segment $s_\ell$ at a level $\ell$, initially its loc-range and its vis-range coincide with the range of $x$-coordinates of $s_\ell$. As the sweep-line moves upward, both ranges get clipped by N-edges whose $x$-ranges intersect them. Finally, when

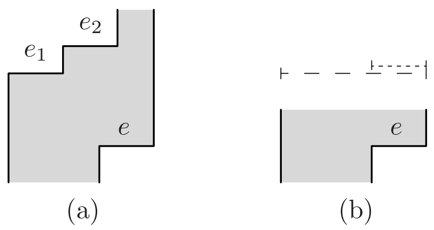


Figure 2: (a) A guard needs to be placed no higher than the N-edge $e_2$ to watch the S-edge $e$; (b) the f-range (shown dotted) and the p-range (shown dashed) of the S-edge $e$.

a N-edge $e$ is encountered such that the (possibly clipped) loc-range of $g$ is a subset of the closure of the $x$-range of $e$, then $g$ is placed at the point $(x_l, \ell)$ where $x_l$ is the left bound value of $g$'s loc-range right before the N-edge $e$ is encountered (in accordance with the convention followed by [3, 5]).

## 3.2 Determining Where to Place a Guard

Consider any S-edge $e$ of the given polygon; see Figure 2(a). As long as the $x$-range of no encountered N-edge intersects the $x$-range of $e$, then a guard at a level higher than the level of the N-edge can see the entire $e$; this is the case with the N-edge $e_1$ in Figure 2(a). However, if the $x$-range of a N-edge $d$ intersects $e$'s $x$-range, then a guard must be placed at a level *between (and including) the levels of* $e$ *and* $d$ since no guard at a level higher than the level of $d$ can see the entire $e$; see edges $e$ and $e_2$ in Figure 2(a). Additionally, if $\ell$ is the level to place such a guard, the requirement that the guard sees the entire edge $e$ implies that the guard needs to be placed at *the orthogonal projection of the grid segment containing* $e$ *onto level* $\ell$.

Therefore, in order to enforce the above observations, each S-edge $e$, when processed, submits a *type-1 guard-request* with which we maintain:

- a *forcing-range*, or *f-range* for short, which is the $x$-range of the edge $e$ (because a guard is needed to watch $e$ if the $x$-range of a N-edge above $e$ intersects $e$'s f-range);
- a *placement-range*, or *p-range* for short, which is the $x$-range of the grid segment containing $e$ (this is the initial range of $x$-coordinates of the guard's location).

Each of these ranges is a *single interval* of $x$-coordinates (the f-range is open, the p-range is closed), and it always holds that the f-range of a S-edge is a subset of its p-range (Figure 2(b)).

In fact, there is one more case in which we need a guard-request. See Figure 3 (left). While processing the N-edge $e$, a guard $g$ gets positioned as shown to watch the lowermost S-edge. The same guard watches

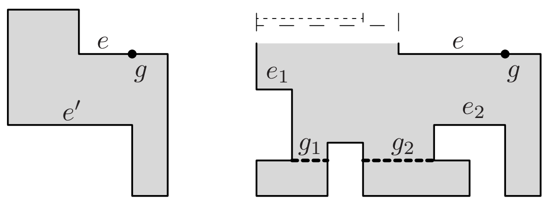


Figure 3: Type-2 guard-requests (f-range shown dotted, p-range shown dashed).

the S-edge $e'$ which justifies the removal of the guard-request produced due to $e'$; however, if we do not do anything else, no need will be recorded for a guard to watch the orthogonal projection of $e'$ onto a level slightly above the level of $e$. This clearly leads to an error in the case of Figure 3 (left).

Therefore, at each N-edge $e$ (of a trouser $T$), we investigate the need to place a *type-2 guard-request*. Let $I$ be the grid-segment of $T$ at a level slightly above $e$'s level, and let $U$ be the set of points of $I$ that are not watched by any of the currently placed guards. If $U = \emptyset$, no guard-request is needed. Otherwise, a guard-request $r$ is submitted with p-range equal to $I$ and f-range equal to $(x_l, x_r)$ where $x_l$ ($x_r$, resp.) is the $x$-coordinate of the leftmost (rightmost, resp.) point in $U$ (see Figure 3 (right)).

Here is how the f- and p-range of a guard-request $r$ submitted by an edge $e$ are used: During the sweeping, as long as we encounter N-edges whose $x$-ranges do not intersect either range, no change occurs. If we encounter a N-edge whose $x$-range intersects the p-range of $r$, then the p-range simply gets clipped. However, if we encounter a N-edge $d$ whose $x$-range intersects the f-range of $r$, then a guard is needed immediately; any guard located at a level between (and including) the levels of $e$ and $d$, which can be positioned at a point with $x$-coordinate in the p-range of $r$ will do. Then, the loc-range of the guard chosen to take care of the guard-request $r$ is clipped about the p-range of $r$ (in this way, the guard will be able to meet the need recorded by $r$ and to watch as much of the *unseen* polygon as possible), and the guard-request $r$ is discarded.

## 3.3 Selecting a Guard to Watch a S-Edge

Many guards at different levels in the polygon may be able to take care of the need recorded by a guard-request $r$ when the f-range of $r$ is intersected by the $x$-range of a N-edge. In order to make a good choice among them, we apply the policy suggested in the following lemma (the proof takes advantage of Lemma 2.1(iii)).

**Lemma 3.1** *Whenever a guard-request needs to be fulfilled, among all guards that can fulfill it, it suffices to choose the lowermost one.*

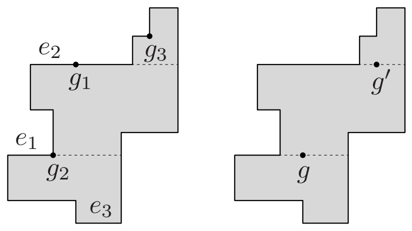


Figure 4: Not selecting the lowermost candidate guard may lead to a non-minimum number of guards.

In fact, there are cases where by choosing a guard other than the lowermost available we get an incorrect result; see Figure 4. When encountering the N-edges $e_1$ and $e_2$, we realize that guards are needed at these levels. If when assigning a guard to watch the S-edge $e_3$, we select a guard at the level of $e_2$ (see guard $g_1$ in the polygon at left), then a third guard $g_3$ will also be needed; yet, two guards suffice to watch the entire polygon as shown at right.

**3.4 Outline of the Algorithm**

As mentioned, we sweep the given class-3 orthogonal polygon from bottom to top maintaining information on the current trousers. The trousers are stored in a balanced binary search tree in order from left to right so that we can quickly insert trousers, delete trousers, and search for the trouser incident on an edge (see Lemma 2.1(ii)). Along with each trouser $T$, we store lists for the guards' loc-ranges and vis-ranges, a list for the guard-requests' p-ranges, and two doubly-linked lists for the guard-requests' f-ranges, one ordered by increasing left endpoint of the f-ranges the other ordered by decreasing right endpoint, with each pair of nodes in these two lists corresponding to the same request linked to each other. We also maintain sets $Positioned(T)$ and $Available(T)$, storing the guards in $T$ that can watch points in $P$ above the current position of the sweep-line or not, respectively.

During the sweeping, we stop at each horizontal edge $e$ and process it. If $e$ is a S-edge, we update the trouser information and set up and insert a corresponding type-1 guard-request. If $e$ is a N-edge, we process the guard-requests whose f-ranges are intersected by $e$'s $x$-range, we position the guards whose loc-ranges are subsets of the closure of $e$'s $x$-range, we clip the p-ranges of all the guard requests and the loc- and vis-ranges of the guards, and we conditionally set up and insert a type-2 guard-request.

After all the edges have been processed, the resulting guard set *Positioned* gives us the locations of a minimum-cardinality set of $r$-visibility guards.

Since at any given time the number of trousers, the number of guard-requests, and the number of guards is linear in the size of the given polygon, we can show the following theorem (details can be found in [8]):

**Theorem 3.1** *Let $P$ be a simple class-3 orthogonal polygon with $n$ vertices. Then, a minimum-cardinality set of $r$-visibility guards can be computed in $O(n^2)$ time using $O(n)$ space.*

## 4 Open Problems

An immediate open question is to investigate how ideas from this work can be generalized to yield algorithms for the problem of $r$-star covering a general simple orthogonal polygon.

Another interesting open question is to try to obtain faster algorithms for the $s$-star covering problem on general simple orthogonal polygons; the current fastest algorithm requires $O(n^8)$ time [6] and is based on the graph-theoretic approach.

Finally, it would also be interesting to try to improve the time complexity of our algorithm.

## References

[1] A. Aggarwal, *The Art Gallery Theorem: its Variations, Applications, and Algorithmic Aspects*, PhD Thesis, Department of Electrical Engineering and Computer Science, John Hopkins University, 1984

[2] J. Culberson and R.A. Reckhow, Orthogonally convex coverings of orthogonal polygons without holes, *J. Comput. Systems Science* 39(2), 166-204, 1989

[3] L. Gewali, M. Keil, and S.C. Ntafos, On covering orthogonal polygons with star-shaped polygons, *Information Sciences* 65, 45-63, 1992

[4] J.M. Keil, Minimally covering a horizontally convex orthogonal polygon, *Proc. 2nd Annual ACM Symp. Computational Geometry*, 43-51, 1986

[5] A. Lingas, A. Wasylewicz, and P. Żyliński, Note on covering orthogonal polygons with star-shaped polygons, *Information Processing Letters* 104(6), 220-227, 2007

[6] R. Motwani, A. Raghunathan, and H. Saran, Covering orthogonal polygons with star polygons: the perfect graph approach, *J. Comput. Systems Science* 40, 19-48, 1990

[7] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987

[8] L. Palios and P. Tzimas, Covering class-3 orthogonal polygons with the minimum number of $r$-stars, Technical Report, Department of Computer Science, University of Ioannina, 2012

[9] J. Urrutia, Art gallery and illumination problems, *Handbook of Computational Geometry*, Elsevier Science, Amsterdam, 973-1027, 2000

[10] C. Worman and J.M. Keil, Polygon decomposition and the orthogonal art gallery problem, *International Journal of Computational Geometry & Applications* 17(2), 105-138, 2007

# Extending Visibility Polygons by Mirrors to Cover Specific Targets

Arash Vaezi* Mohammad Ghodsi†

## Abstract

The visibility polygon *VP* of a point $q$ ( $VP(q)$) inside a simple polygon $P$ with $n$ vertices, can be computed in linear time. We propose a linear time algorithm to extend *VP*, by converting some edges of $P$ to mirrors, so that a given segment $d$ can also be seen from the viewer. In linear time our algorithm finds every edge such that, when converted to a mirror, makes $d$ visible to our viewer.

## 1 Introduction

Many variations of the visibility polygon have been studied so far. In general, we have a simple polygon $P$ with $n$ vertices, and a viewer point $q$ inside $P$. The goal of the visibility problem is to find the maximal sub-polygon of $P$ ( $VP(q)$) visible to the viewer. There are linear time algorithms to compute $VP(q)$ ([6], [3]).

It was shown in 2010 that *VP* of a given point or segment can be computed in presence of a mirror in $O(n)$ [5]. Also, it was shown in the same paper that the union of two visibility polygons can be computed in $O(n)$.

We consider the problem of finding all edges any of which when converted to a mirror (and thus called *mirror-edge*) can make a specific segment visible (also called *mirror-visible*) to a given point. We propose a linear time algorithm for this problem.

This paper is organized as follows: In Section 2, notations are described. Next in Section 3, we present a linear time algorithm to solve the above problem. Section 4 contains some discussions and future works.

## 2 Notations

Suppose $P$ is a simple polygon and $int(P)$ denote its interior. Two points $x$ and $y$ are visible to each other, if and only if the open line segment $\overline{xy}$ lies completely in $int(P)$. The visibility polygon of a point $q$ in $P$, denoted as $VP(q)$, consists of all points of $P$ visible to $q$. Edges of $VP(q)$ that are not edges of $P$ are called *windows*. Weak visibility polygon of a segment $d$, denoted as $WVP(d)$, is the maximal sub-polygon of $P$ visible to at least one point (not the endpoints) of $d$. The visibility of an edge $v_iv_{i+1}$ of a simple polygon $P$ can be viewed in different ways [1]: $P$ is said to be *completely visible* from $v_iv_{i+1}$ if every point $z \in P$ and for any point $w \in v_iv_{i+1}$, $w$ and $z$ are visible. All these different visibilities can be computed in linear time (see [4] for the weakly visibility polygon and [1] for the strongly).

Suppose an edge $e$ of $P$ is a mirror. Two points $x$ and $y$ are $e$-mirror-visible, if and only if they are directly visible with one specular reflection. $VP(q)$ with a mirror-edge $e$, is the maximal sub-polygon of $P$ visible to $q$ either directly or via $e$.

Two points or segments are *mirror-visible* if and only if they cannot see each other directly, but can with an edge middling as a mirror. We consider the whole edge as a mirror, thus two points can be mirror-visible by just a part of an edge. Also, if a point can see a part of a segment through a mirror, we call them mirror-visible.

## 3 Expanding a point visibility polygon

### 3.1 Recognizing all mirror-edges

We intend to find all edges of $P$, any of which when converted to a mirror causes a given point $q$ see a segment $d$.

**Theorem 1** *Suppose $P$ is a simple polygon with the complexity of $n$, $q$ is a given point inside $P$, and $d$ is a given segment which is not directly visible by $q$. All edges any of which can makes $d$ mirror-visible to $q$ can be found in $O(n)$ time.*

*Remark.* We will prove this theorem using a given diagonal of $P$, instead of the given segment. We will use the two endpoints of the diagonal. Since the assertion that the segment is actually a diagonal in not used in the proof, the stated proof holds for any segment inside $P$. Instead of the endpoints of the diagonal, we can use one endpoint of the closet edge of $P$ to the given segment. Let at least one endpoint of this edge be upon the given segment inside the polygon.

**Proof.** First we prove that with an $O(n)$ time of preprocessing, we can answer any query of whether a particular edge of $P$ can make $d$ mirror-visible to $q$ in $O(1)$ time. The processing time is for computing

*Department of Computer Engineering, Sharif University , Technology, avaezi@ce.sharif.ir

†Department of Computer Engineering, Sharif University of Technology, and Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. ghodsi@sharif.edu. This author's research was partially supported by the IPM under grant No: CS1389-2-01

$VP(q)$ and $WVP(d)$, and finding some reflex vertices which may block the mirror-visibility area.

Obviously, any mirror-edge that makes $d$ visible to $q$ should lie in the intersection of $VP(q)$ and $WVP(d)$ which can be computed in linear time. If goal is the visibility of the whole segment, we should compute the complete visibility polygon of the given segment instead of the weakly visibility polygon of which.

Suppose that $e$ is intersected by both visibility polygon from $v_1(e)$ to $v_2(e)$ in the order of $P$'s vertices. We assume that this part of $e$ is mirror. We will find out whether any part of $d$ is $e$-mirror-visible. Let $L_1$ and $L_2$ be two half-lines from the ray-reflection of q at $v_1(e)$ and $v_2(e)$ respectively (see Figure 1).

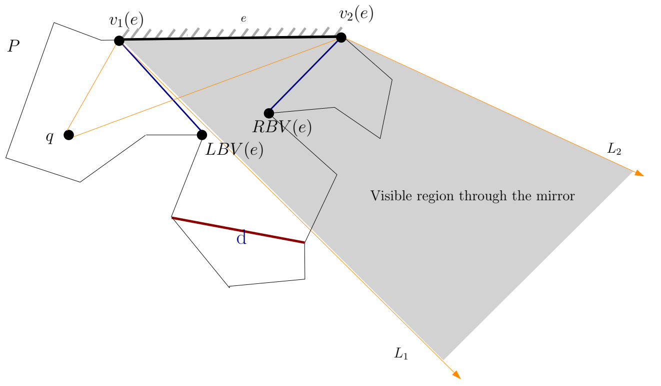


Figure 1: The region between $L_1$ and $L_2$ is the visible area by $q$ through the mirror $e$.

If $d$ is in the region between $L_1$ and $L_2$ and no part of $P$ obstructs $d$, then $d$ is $e$-mirror-visible (see Figure 1). Since $P$ is simple, any obstruction has to contain a reflex vertex. Considering that $P$'s vertices are ordered in clockwise direction, we define $LBV(e)$ (for Left Blocking Vertex of $e$) to be the reflex vertex before $v_1(e)$ that can obstruct most the $e$-mirror-visibility of $d$, and similarly $RBV(e)$ (for Right-Blocking Vertex of $e$) to be the reflex vertex after $v_2(e)$. Different mirror-edges may have the same $LBV$s or $RBV$s, but $LBV$ and $RBV$ for any edge $e$ are unique and may be $v_1(e)$ or $v_2(e)$. It is sufficient to check only the corresponding $LBV$ and $RBV$ vertices not to block the mirror visibility area. We will show that we can find all $LBV$s and $RBV$s for all mirror-edges in linear time.

Following cases should be considered:

1. If $L_1$ and $L_2$ both lie on one side of $d$, $d$ is not in the mirror-visible area. $q$ cannot see $d$ through this mirror-edge.
2. If $L_1$ and $L_2$ do not lie on one side of $d$ and if $d$ is in the middle of the mirror-visible area, $q$ can see $d$ through the mirror-edge. Because $e$ is visible to $d$, and the visibility area from $L_1$ to $L_2$ is a continuous region.
3. Otherwise $L_1$ or $L_2$ crosses $d$, and we should check whether any part of $P$, obstructs *the whole* visible area through the mirror-edge (In the case of the completely visibility polygon of $d$, it is sufficient to check $L_1$ and $L_2$ not to cross $d$, except in its endpoints.

   Now, we should check the polygon not to block the rays from the right or form the left side of the mirror-edge. Consider the two segments $s_1 = \overline{LBVv_1(e)}$ also $s_2 = \overline{RBVv_2(e)}$. If $L_2$ crosses $s_1$, or if $L_1$ crosses $s_2$, consequently $q$ and $d$ are not $e$-mirror-visible.

Obviously, collision checking of a constant number of points is done in $O(1)$ for any candidate edge in the intersection polygon of $VP(q)$ and $WVP(d)$, which in addition to the processing time leads to an $O(n)$ algorithm to find all feasible mirror-edges. □

### 3.1.1 Computing $LBV$ and $RBV$ vertices

First we consider the computation of the $LBV$ vertices. Starting from $d_1$ (the left endpoint of the given diameter) and similar to Graham's algorithm [2] in finding the convex hull of points, we trace $WVP(d)$ using the reflex vertices of $P$ in the $P$'s order of vertices. We act as the following:

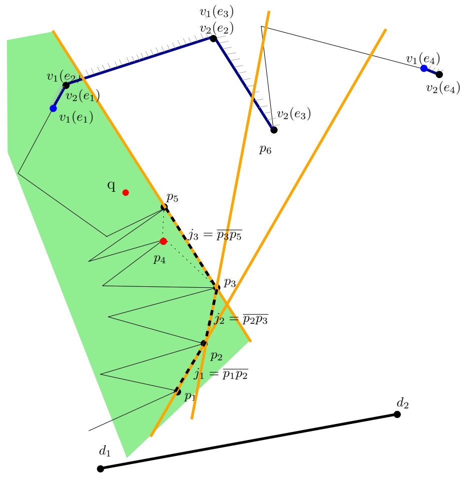


Figure 2: Constructing the convex hull for distinguishing $LBV$ vertices for all the mirror-edges. $p_1$, $p_2$, $p_3$ and $p_5$ are the vertices of the convex hull. Four mirrors are shown, for example $p_5$ is $LBV(e_1)$ .

Suppose the reflex vertices from $d_1$ to $d_2$, are $p_1, p_2, ..., p_k$, where $k \in O(n)$. We start making the convex hull of the reflex vertices, whose concave region is directed to the outside of the polygon (see Figure 2). When we reach a new mirror-edge, we use the lines containing the edges of the updated convex shape till that moment.

Assume $v_2(e_i)$ represents the second endpoint of the *ith* candidate mirror-edge. When we reach the *ith* mirror-edge we compare $v_2(e_i)$ with the *chosen line* for the $(i-1)th$ mirror-edge (the last visited mirror-edge). If $v_2(e_i)$ lies on–or on the left side of–that line, then the $i$th and $(i-1)$th mirror-candidates have the same reflex vertex as their $LBV$. Otherwise, we should compare $v_2(e_i)$ with the line which is not checked yet, and contains the most recent constructed segment of the convex shape.

For example, suppose the convex shape has 3 segments $(j_1, j_2, j_3)$ before we reach the first mirror-edge $(e_1)$ with $v_2(e_1)$. We should check $v_2(e_1)$ with the line, which contains the last constructed segment of the convex hull $(j_3)$, to see if it has $v_2(e_1)$ on its left. If $v_2(e_1)$ lies on the right side of that line, we check $v_2(e_1)$ with $j_2$. We can continue this way, if there is no more line, there is no $LBV(e_1)$. Assume we select the line containing $j_3$ for $e_1$. We should check this line for $e_2$ too, because they may have the same $LBV$.

At the end, suppose for a mirror-edge $e$ we chose line $L$, whose interior contains more than one vertex of the convex shape. If $v_2(e)$ is on the left side of $L$ then we should choose the most recent joined vertex of the convex shape on $L$. But, if $v_2(e)$ itself lies on the $L$, then we choose the first reflex vertex which joined the convex shape on $L$. This reflex vertex is the *closet* one on $L$ to $d$.

However, While we trace $WVP(d)$, facing with any new reflex vertex we should update the convex shape (see Figure 3).

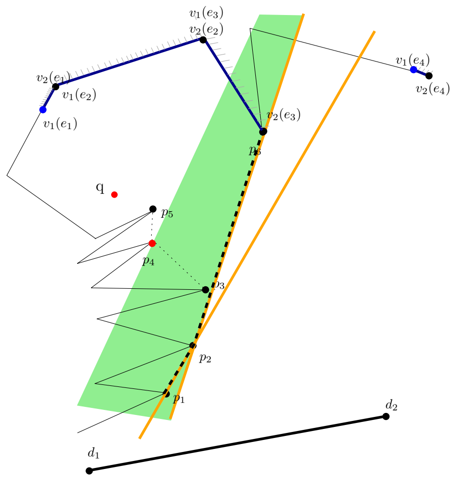


Figure 3: Updating the convex shape while tracing $WVP(d)$ and facing with new reflex vertices. $p_5$ is chosen as $LBV(e_1)$, $p_3$ and $p_2$ as $LBV(e_2)$ and $LBV(e_3)$, respectively. If we had a reflex vertex $p_0$ for the fourth mirror-edge, first we may have selected $p_1$. But later we should change it to $v_1(e_4)$, because $p_1$ cannot block the $e_4$-mirror-visibility.

We trace the $WVP(d)$ in counter-clockwise direction starting from $d_2$ to find all $RBV$ vertices similarly. At the end, since there may be some false chosen $LBV$ or $RBV$ vertices, we should trace $WVP(d)$ again in both directions. First we compare all $LBV$ vertices of the mirror-edges with the corresponding segment $\overline{v_2(e_i), d_1}$. If it was in the left side of the segment, then the chosen vertex is not obstructing the mirror-visibility area, hence, we change the chosen $LBV$ to $v_1(e_i)$ for the $i$th mirror-edge. We proceed similarly for $RBV$ vertices in the other direction.

### 3.1.2 Proof of correctness

First, the following lemma:

**Lemma 2** *For each mirror-edge $e$ the reflex vertex as its $LBV$, is among the reflex vertices before $e$ in the $P$'s order of vertices ($p_i$ $0 \leq i \leq k$). And it is the closet one to $d$ ($p_j$) which the corresponding segment $\overline{p_j v_2(e)}$ holds all the other $p_i$ ($i \neq j$ $0 \leq i \leq k$) reflex vertices on its left side (see Figure 4).*

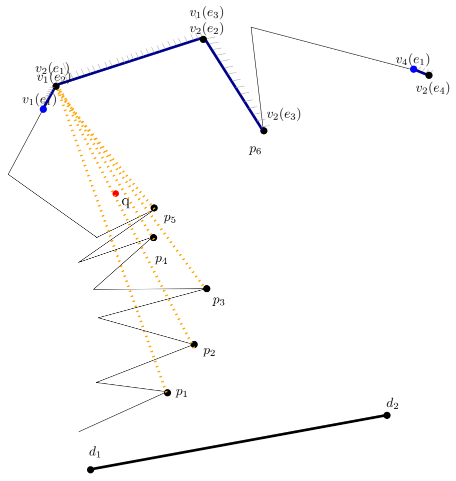


Figure 4: From Lemma 2 point $p_5$ must be the best choice for $LBV(e_1)$.

**Proof.** Suppose $e$ is our mirror-edge and $p_j$ is the chosen $LBV(e)$ using Lemma 2. We will show neither a farther reflex vertex nor a closer one to $d$ is a better choice for being $LBV(e)$. Actually, we will show there are examples of violations for any other reflex vertices, either farther or closer to $d$.

Suppose there is a reflex vertex $p_l$, closer than $p_j$ to $d$. Also assume $L_2$ crosses the polygon on the left of $p_l$ but not on the left side of $p_j$. Therefore, $p_l$ obstructs the $e$-mirror-visible area so that the viewer cannot see $d$ through $e$. This leaves $p_j$, chosen from Lemma 2, not to be $LBV(e)$. We know that $L_2$ should place on the right side of $\overline{d_1 v_2(e)}$, because otherwise the whole mirror-visibility region is on the left side of $d$. But,

$p_j$ is chosen by Lemma 2, hence, it should lie on the right side of $\overline{p_l v_2(e)}$ (see Figure 5). Thus, $L_2$ cannot cross the polygon on the left side of $p_l$ but not on the left side of $p_j$. In the stated analysis if the reflex vertices such as $p_j$ or $p_l$ lie *on* $L_2$, they can block the mirror visibility area. Therefore, we treat them as if they were on the right side of $L_2$.

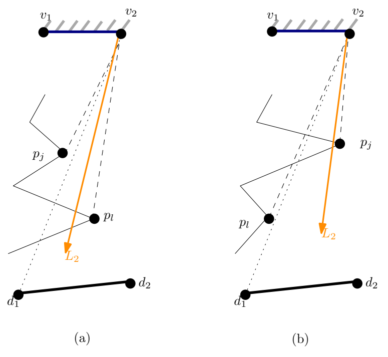


Figure 5: No lower reflex vertex can be a better choice than the one chosen by Lemma 2

Similarly, assume a reflex vertex $p_h$ exists, which is farther than $p_j$ to $d$. And $L_2$ crosses the segment $\overline{p_h v_2(e)}$ and not $\overline{p_j v_2(e)}$. From Lemma 2 $p_h$ should be at the left side of $\overline{p_j v_2(e)}$, and we know that $L_2$ has intersection with $d$ on the right side of $\overline{d_1 v_2(e)}$. Consequently, $L_2$ crosses $d$ on the right side of $p_j$, and crosses $\overline{p_h v_2(e)}$ while it lies on–or on the left side of– $p_j$. Therefore, it should cross $\overline{p_j v_2(e)}$, and we are done. □

Thus, using Lemma 2, when there are $O(n)$ reflex vertices $LBV$s for all the mirror-edges can be computed within $O(n^2)$ time complexity.

Now consider two mirror-edges $e_1$ and $e_2$ and two reflex vertices $p_i$ and $p_{(i-1)}$, which the segment $\overline{p_i v_2(e_1)}$ has all other reflex vertices on its left side for $e_1$. Also, the segment $\overline{p_{(i-1)} v_2(e_2)}$ acts the same for $e_2$. Hence, $p_i$ and $p_{(i-1)}$ are $LBV(e_1)$ and $LBV(e_2)$, respectively. The segment $\overline{p_i p_{(i-1)}}$ is between those lines. Therefore, for all the mirror-edges $(e)$ which have their $v_2$ endpoints on the left side of $\overline{p_i p_{(i-1)}}$ the segment $\overline{p_i v_2(e)}$ is covering all the other reflex vertices on its left. The segments on the right side of $\overline{p_i p_{(i-1)}}$ should be considered one by one. s$\overline{p_{(i-1)} v_2(e)}$ can act the same for those mirror-edges whose $v_2$ vertices lie on the right side of $\overline{p_i p_{(i-1)}}$.

The lines containing the $\overline{p_i p_{(i-1)}}$ segments $(1 \leq i \leq k)$ evidently satisfy the property of being convex hull, the direction of the concave region of which is to the outside of the polygon.

As we mentioned before, each $LBV(e)$ vertex should lie on the right side of $\overline{d_1 v_2(e)}$ of $e$, otherwise, we should exchange the chosen $LBV(e)$ with $v_1(e)$. Therefore, at the end we should check all of $LBV$ vertices. The reason is, in these cases there is nothing to obstruct the visibility region from the left side of the mirror-edge. Clearly, all the stated analysis holds in computing $RBV$ vertices, too.

## 4 Discussion

We dealt with the problem of extending the visibility polygon of a given point in a simple polygon, so that another segment becomes visible to it. For this purpose we convert some of the polygon edges to mirrors. The problem is to find all such kind of edges. Using the algorithm we proposed, it is possible in linear time corresponding the complexity of the simple polygon. We only discussed finding the edges to be mirrors, but it is shown that having two mirrors, the resulting visibility polygon, may not be a simple polygon [6]. The problem can be extended as; put mirrors inside the polygon, a point with a limited visibility area and so on.

## References

[1] D. Avis, G. T. Toussaint. An optional algorithm for determining the visibility of a polygon from an edge. *IEEE Transactions on Computers*, C-30: 910-1014, 1981.

[2] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf Computational Geometry Algorithms and Applications. Springer, third edition *Department of Computer Science Utrecht University*, 13,14 2008.

[3] H. ElGindy, D. Avis. A Linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2: 209-233, 1987.

[4] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2: 209–233, 1987.

[5] B. Kouhestani, M. Asgaripour, S. S. Mahdavi, A. Nouri and A. Mohades. Visibility Polygons in the Presence of a Mirror Edge. *In Proc. 26th European Workshop on Computational Geometry*, 26: 209–212, 2010.

[6] D. T. Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, 22: 207–221, 1983.

# Rotation minimizing frames on monotone-helical PH quintics: approximation and applications to modeling problems

Fatma Şengüler-Çiftçi[*] Gert Vegter[†]

## Abstract

A *rotation minimizing frame* (RMF) $\{\mathbf{t}, \mathbf{f}_1, \mathbf{f}_2\}$ of a curve in 3-space consists of the tangent $\mathbf{t}$ and two normal vectors $\mathbf{f}_1$ and $\mathbf{f}_2$ which rotate as little as possible around $\mathbf{t}$. Having the property of minimum twist makes RMFs attractive in computer graphics, swept surface constructions, motion design and similar applications. Recently we have shown that there is not any rational RMF (RRMF) on monotone quintic helices, so this motivates to develop a rational approximation to RMFs. It is shown that rational approximation to RMFs on monotone-helical Pythagorean-hodograph (PH) quintics is computationally cheap, then it is applied to profile surface modeling and rigid body design.

## 1 Introduction

### 1.1 General context

A parametric curve $r(t) = (x(t), y(t), z(t))$ is called a *Pythagorean-hodograph* (PH) curve if its speed is a polynomial [2]. The theory of PH curves is a much studied research topic in Computer Aided Geometric Design (CAGD) because of their useful properties. An *adapted frame* on a space curve $\mathbf{r}(t)$ is an orthonormal moving frame $\{\mathbf{t}, \mathbf{f}_1, \mathbf{f}_2\}$ such that, $\mathbf{t}$ is the unit tangent $\mathbf{r}'(t)/|\mathbf{r}'(t)|$, and the other two vectors span the normal plane. Rotation minimizing frames (RMFs) have minimum twist that makes them distinguished among adapted frames. RMFs are used in animation, robotics applications, the construction of swept surfaces [10] where the axis of a tool should remain tangential to a given spatial path while minimizing changes of orientation about this axis.

### 1.2 Motivation

It is easy to compute *exact* derivation of RMFs on spatial PH curves [2]. Further, in practical applications, especially *rational* RMFs (RRMFs) are useful for computational purposes. The only curves having rational adapted frames are PH curves, since the only PH curves have rational unit tangent vectors. Besides, the arc lengths of PH curves can be computed precisely, and it can formulate real-time interpolators to drive multi-axis CNC machines along curve paths, at fixed or varying speeds from their exact analytic descriptions [6].

In the family of PH curves, polynomial helices have remarkable interest, particularly quintic helices. The relationship between such curves and some problems in the realm of computer-aided design of curves and surfaces show that the suitable curves are helical PH quintics for real applications [3].

### 1.3 Problem Statement

Having observed that in general PH quintic helices cannot have RRMFs, we aim at making rational approximations to RMFs. We focus on *monotone-helical PH quintics*, i.e. curves whose hodograph has coordinates with a common factor, say $h$. On a monotone-helical PH quintic curve $\mathbf{r}(t)$, RMFs can be computed easily since there is a simplification in the integral giving the angle $\theta$ between *Frenet-Serret frame* (FSF) and RMFs. Because, $\theta$ is given by

$$\theta(t) - \theta_0 = -\int \tau\,\sigma\,dt, \qquad (1)$$

where $\sigma = |\mathbf{r}'(t)|$ and $\tau$ is the torsion, and for monotone-helical PH quintics the integrand $\tau\,\sigma$ turns out to be $\frac{2}{gc}$, with $\sigma = h\,g$ and $c$ is the helicity constant. Applying this idea to related topics, such as sweep surface modeling and rigid body motion design, is the subject of this work.

### 1.4 Related Work

In the previous work [11], we showed that there does not exist RRMFs on monotone-helical PH quintics.

**Theorem 1** *[11] There is not an RRMF on a (regular) monotone-helical PH quintic that is not a straight line.*

We also gave a condition (9) on a polynomial helix of any degree to have an RRMF. This condition leads to a simplification of rational approximation to RMFs on monotone-helical PH quintics. For PH cubic curves and more generally PH curves rational approximation to RMFs was studied in [5, 9].

[*]Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, `F.A.Senguler-Ciftci@rug.nl`

[†]Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, `G.Vegter@rug.nl`

### 1.5 Overview of the Results

The present paper is organized as follows. Section 2 introduces definitions of and basic results on monotone-helical PH quintics, RMFs, RRMFs, and profile surfaces. In Section 3 we discuss a minimax rational approximation to an RMF on a monotone-helical PH quintic. Then we discuss applications to sweep surface modeling in the same section and to rigid body motion planning in Section 4. Subsequently in Section 5 we conclude with remarks about our future considerations.

## 2 Background

In this section we review the preliminary material which we use along the paper.

### 2.1 Monotone-Helical PH Quintics

In Hopf map $\mathbb{C}\times\mathbb{C}\to\mathbb{R}^3$ representation, a PH curve $\mathbf{r}(t)$ is defined by its hodograph

$$\mathbf{r}'(t) = (2\,\alpha(t)\,\bar{\beta}(t), |\alpha(t)|^2 - |\beta(t)|^2), \tag{2}$$

where

$$\alpha(t) = u(t) + v(t)\,\mathrm{i} \text{ and } \beta(t) = q(t) + p(t)\,\mathrm{i}, \tag{3}$$

are some complex polynomials, and the identification $\mathbb{R}^3 \simeq \mathbb{C}\times\mathbb{R}$ is assumed [2].

A *monotone-helical PH curve* $\mathbf{r}(t)$ is a quintic PH curve whose hodograph have components with a common quadratic factor, then $\alpha(t) = h(t)\,a(t)$ and $\beta(t) = h(t)\,b(t)$ for linear complex polynomials $a(t), b(t)$ and $h(t)$. Then the helical PH quintic curve (2) is given by

$$\mathbf{r}'(t) = |h(t)|^2(2\,a(t)\,\bar{b}(t), |a(t)|^2 - |b(t)|^2). \tag{4}$$

**Example [2]:** Let us consider the monotone-helical PH quintic curve $\mathbf{r}(t) = (x(t), y(t), z(t))$, where

$$\begin{aligned} u(t) &= t^2 - 3\,t, & v(t) &= t^2 - 5\,t + 10,\\ p(t) &= -\,2\,t^2 + 3\,t + 5, & q(t) &= t^2 - 9\,t + 10. \end{aligned} \tag{5}$$

Here a common factor of the components $x(t)$, $y(t)$, $z(t)$ is $h(t) = t^2 - 2\,t + 5$. The helicity constant is obtained to be $c = \kappa/\tau = 5\sqrt{2}/3$, where $\kappa, \tau$ are the curvature and torsion, respectively. We will make use of this curve to demonstrate our approximation results.

### 2.2 Rotation Minimizing Frames

The most canonical adapted frame is the FSF $\{\mathbf{t}, \mathbf{n}, \mathbf{b}\}$. There are many other adapted frames associated with a given space curve $\mathbf{r}(t)$, and among them the RMFs are the ones which minimize the amount of rotation along the curve. The variation of a frame $\{\mathbf{t}, \mathbf{f}_1, \mathbf{f}_2\}$ defined on a curve $\mathbf{r}(t)$ is given by its vector angular velocity $\omega = \omega_0\,\mathbf{t} + \omega_1\,\mathbf{f}_1 + \omega_2\,\mathbf{f}_2$ with the relations

$$\mathbf{t}' = \omega\times\mathbf{t}, \quad \mathbf{f}_1' = \omega\times\mathbf{f}_1, \quad \mathbf{f}_2' = \omega\times\mathbf{f}_2. \tag{6}$$

The characteristic property of an RMF is that its angular velocity has no component along $\mathbf{t}$, i.e.,

$$\omega\cdot\mathbf{t} \equiv 0. \tag{7}$$

As we consider a helix $\mathbf{r}(t)$, its FSF is rational [2]. Observe that an RMF is given by a rotation in the normal plane

$$\begin{pmatrix}\mathbf{f}_1\\ \mathbf{f}_2\end{pmatrix} = \begin{pmatrix}-\cos\theta & \sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}\mathbf{n}\\ \mathbf{b}\end{pmatrix}, \tag{8}$$

where (1) with the integration constant $\theta_0$ [2]. Therefore an RMF is not rational in general.

### 2.3 Rational Frames of Quintic PH Helices

A general condition on helices of any degree to have RRMFs is also given in [11].

**Lemma 2** *[11] Let a PH curve $\mathbf{r}(t)$ be a helical curve with $\kappa/\tau = c$ and $c\in\mathbb{R}$. Then $\mathbf{r}(t)$ has an RRMF if and only if there exist relatively prime polynomials $\mu(t)$ and $\nu(t)$ satisfying*

$$\frac{\sqrt{\rho}}{c\,\sigma} = \frac{\mu\,\nu' - \mu'\,\nu}{\mu^2 + \nu^2}, \tag{9}$$

*where*

$$\begin{aligned}\rho = {}&(u\,p' - u'\,p + v\,q' - v'\,q)^2 +\\ &(u\,q' - u'\,q + v\,p' - v'\,p)^2.\end{aligned} \tag{10}$$

The proof of Lemma 2 gives an idea of a simplification of rational approximation to RMFs on monotone-helical PH quintic curves. It will be detailed in the next section.

### 2.4 Profile Surfaces

A *profile surface* is a sweep surface generated by an RMF. More explicitly, it has a parametric representation

$$\mathbf{S}(s,t) = \mathbf{r}(t) + \mathbf{f}_1(t)\,c_1(s) + \mathbf{f}_2(t)\,c_2(s), \tag{11}$$

where $\mathbf{r}(t)$ is the *spine curve* with parameter $t\in[t_0, t_1]\in\mathbb{R}$, $c(s) = (c_1(s), c_2(s))^T$ is the *cross section* or *profile curve* with parameter $s\in[s_0, s_1]\subset\mathbb{R}$, and $\{\mathbf{t}, \mathbf{f}_1, \mathbf{f}_2\}$ is an RMF along $\mathbf{r}(t)$.

If the cross-section curve is a straight line, then the profile surface is a developable surface [9]. This implies that they are flat surfaces, i.e. they have vanishing Gauss curvature $K = 0$. In the next section we obtain a rational approximation of an RMF on a monotone-helical PH quintics.

## 3 Minimax Rational Approximation on monotone-helical PH quintics

In this section we will make a minimax rational approximation on monotone-helical PH quintics by using Mathematica. A $(m,k)$ degree rational function is the ratio of a degree $m$ polynomial to a degree $k$ polynomial. The *error* of minimax rational approximation is the difference between the function and its approximation w.r.t. Euclidean norm. The aim of minimax rational approximation is to minimize the maximum of the relative error from the polynomial curve.

Let $f(t)$ be continuous on a closed interval $[t_0,t_1]$. Then there exists a unique $(m,k)$ degree rational polynomial $\frac{a(t)}{b(t)}$, called the *minimax rational approximation* to exact function $f(t)$, that minimizes

$$\varepsilon(a(t),b(t)) = \max_{t_0<t<t_1} \mid f(t) - \tfrac{a(t)}{b(t)} \mid . \tag{12}$$

### 3.1 Minimax Rational Approximation of RMFs on Monotone-Helical PH Quintics

Nonexistence of RRMFs on a monotone-helical PH quintic curve motivates an approximation of RRMFs which can be done as in [5] with further simplifications as indicated in the following. Standard parametrization of circle is

$$(\sin\theta,\cos\theta) = \left(\frac{2f}{1+f^2},\frac{1-f^2}{1+f^2}\right), \tag{13}$$

where $f=\tan\frac{\theta}{2}$. Then, one can make a rational approximation by

$$f(t) = \tan\frac{\theta(t)}{2} = -\tan\left(\int \frac{\tau\,\sigma}{2}\,dt\right) \simeq \frac{a(t)}{b(t)}, \tag{14}$$

for some relatively prime polynomials $a(t)$ and $b(t)$, which gives a rational frame

$$\begin{pmatrix}\tilde{\mathbf{f}}_1\\ \tilde{\mathbf{f}}_2\end{pmatrix} = -\frac{1}{a^2+b^2}\begin{pmatrix}a^2-b^2 & -2\,a\,b\\ 2\,a\,b & a^2-b^2\end{pmatrix}\begin{pmatrix}\mathbf{n}\\ \mathbf{b}\end{pmatrix}. \tag{15}$$

For a quintic helix, the integrand $\tau\,\sigma$ is a rational function of degree $(2,4)$, while for monotone-helical PH quintic curves this simplifies to $(0,2)$. This is because the monotone-helical PH curve identities hold:

$$\sigma = h\,g \quad \text{and} \quad \rho = h^2, \tag{16}$$

where $h = gcd(x',y',z')$ [4]. Furthermore we put together the curvature of a PH curve [2] $\kappa = 2\,\frac{\sqrt{\rho}}{\sigma^2}$, helicity condition $c=\kappa/\tau$, and monotone-helical conditions (16) in the following computation:

$$\tau\,\sigma = \frac{\kappa}{c}\,\sigma = 2\,\frac{\sqrt{\rho}}{c\,\sigma^2}\,\sigma = \frac{2}{g\,c}. \tag{17}$$

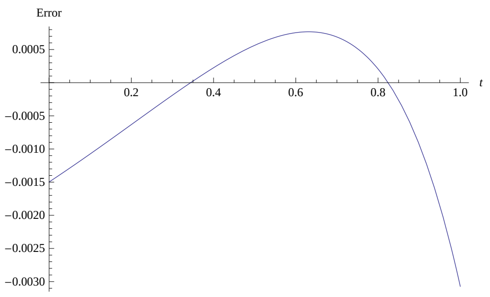


Figure 1: Error for the RMF condition (7).

Then from equation (1), we get

$$\frac{\theta}{2} = -\int \frac{1}{c\,g}\,dt. \tag{18}$$

The integral in (18) gives a simplification of computations when making the approximation given in (14).

**Example:** Let us consider the monotone-helical PH quintic curve (5). After applying the minimax rational approximation with error $\varepsilon = -0.000172464$, the rational approximation to exact function $f(t)$ is then

$$\frac{a(t)}{b(t)} = \frac{-0.469264 + 0.225286\,t}{1-0.304786\,t}. \tag{19}$$

This result yields a good RRMF approximant as can be seen by Figure 1 which shows the error of the RMF condition (7).

### 3.2 Minimax Rational Approximation of Profile Surfaces

Rational approximation of RMF can be used to generate profile surfaces with rational representation. If the profile curve $c(s)$ is chosen to be a straight line then the rational approximation to the profile surface is expected to have Gauss curvature close to zero values.

**Example:** Consider two sweep surfaces,

$$\begin{aligned}\mathbf{S}_1(s,t) &= \mathbf{r}(t) + (-\tfrac{1}{5}\,s+5)\,\tilde{\mathbf{f}}_1 + (10\,s-\tfrac{1}{2})\,\tilde{\mathbf{f}}_2,\\ \mathbf{S}_2(s,t) &= \mathbf{r}(t) + (-\tfrac{1}{5}\,s+5)\,\mathbf{n} + (10\,s-\tfrac{1}{2})\,\mathbf{b},\end{aligned} \tag{20}$$

generated by the rational approximation to the RMF (left) and by the FSF (right) of the monotone-helical PH quintic given in (5), see Figure 2. The Gaussian curvature $\tilde{K}$ can be used as an accuracy criterion. Since the cross-section curve

$$c(s) = (-\tfrac{1}{5}\,s+5, 10\,s-\tfrac{1}{2})^T \tag{21}$$

in this example is a straight line, the Gauss curvature of a profile surface is vanishing. For profile surface $\mathbf{S}_1(s,t)$, minimum and maximum values of the Gauss curvature are

$$\begin{aligned}\tilde{K}_{min}(0.899997,1) &= -\,9.76359\times10^{-11},\\ \tilde{K}_{max}(0.899992,14.7928) &= -\,1.11723\times10^{-11}.\end{aligned} \tag{22}$$

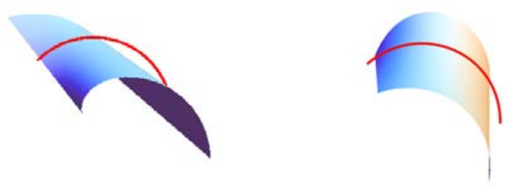


Figure 2: Sweep surfaces $\mathbf{S}_1(s,t)$ and $\mathbf{S}_2(s,t)$, generated by the rational approximation to the RMF (left) and by the FSF (right).

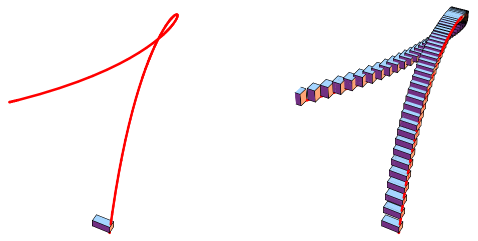


Figure 3: Rigid Body with initial configuration on a monotone-helical PH quintic curve $\mathbf{r}(t)$ (left) and the same curve with rigid body motion (right).

Our approximation $\tilde{K}$ is between the values $\tilde{K}_{min}$ and $\tilde{K}_{max}$ which are close to zero. Therefore this criterion shows us that our approximation gives good results.

## 4 Rigid Body Motion Design

A rigid body motion can be modeled as the motion of an adapted frame. As they make minimum twist, RMFs are very useful in rigid body motion design, however computation of these frames requires to integrate complicates functions. For PH curves the integral in (1) is known to be integrated by elementary functions [2]. We can further see by (18) that this integration is very useful from a computational point of view. This considerable feature of monotone-helical PH quintics can be employed in the following.

Assume that an initial point $p_0$ and a final point $p_1$, and an initial frame at $p_0$ are given. We illustrate that rigid body motion design problem in Figure 3. To find a trajectory satisfying these initial data, a monotone-helical PH quintic can be obtained under some more suitable conditions, then it is an easy task to compute an RMF which aligns with the initial frame at $p_0$. For this purpose interpolation method for monotone-helical PH quintics given in [7] can be used.

## 5 Conclusions and Future Work

Rational approximation of RMFs on monotone-helical PH quintics is studied. It is observed in this work that the integrand (1) which is used to compute RMFs is a rational function of degree (0,2). This leads to a simplification in rational approximation to RMFs as we touch upon in this work. Moreover, it is pointed out that several applications can be done to modeling problems such as sweep surfaces and rigid body design. It is worth to mention here that this distinctive feature is special for monotone-helical PH quintics.

Future work will be to improve and apply the observations outlined above. One concrete question arising is the following. When we are modeling rigid body motion, there exist singular points on the monotone curve. We will search for a method to remove these singularities of monotone-helical PH quintics.

## References

[1] H.I. Choi, S.H. Kwon, and N.S. We, Almost rotation - minimizing rational parameterization of canal surfaces, *Comput. Aided Geom. Design*, 21, 859–881, 2004.

[2] R.T. Farouki, *Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable*, (Berlin: Springer, 2008).

[3] R.T. Farouki, G. Farin, J. Hoschek, and M-S. Kim, *Pythagorean-hodograph curves, Handbook of Computer Aided Geometric Design*, North Holland, 405–427, 2002.

[4] R.T. Farouki, C. Giannelli and A. Sestini, Helical polynomial curves and double Pythagorean hodographs. I. Quaternion and Hopf map representations, *J. Symbolic Comput.*, 44(2), 161–179, 2009.

[5] R.T. Farouki and C.Y. Han, Rational approximation schemes for rotation-minimizing frames on Pythagorean-hodograph curves, *Comput. Aided Geom. Design*, 20, 435–454, 2003.

[6] R.T. Farouki, Y-F. Tsai and B. Feldman, Performance analysis of CNC interpolators for time-dependent feedrates along PH curves, *Comput. Aided Geom. Design*, 18, 245–265, 2001.

[7] C.Y. Han, Geometric Hermite interpolation by monotone helical quintics, *Comput. Aided Geom. Design*, 27, 713–719, 2010.

[8] C.Y. Han, Nonexistence of rational rotation-minimizing frames on cubic curves, *Comput. Aided Geom. Design*, 25, 298–304, 2008.

[9] B. Jüttler, and C. Mäurer, Rational approximation of rotation minimizing frames using Pythagorean-hodograph cubics, *Journal for Geometry and Graphics*, 3, 141–159, 1999.

[10] F. Klok, Two moving coordinate frames for sweeping along a 3D trajectory, *Comput. Aided Geom. Design*, 3, 217–229, 1986.

[11] F. Şengüler-Çiftçi and G. Vegter, Nonexistence of Rational Rotation Minimizing Frames on Quintic Helices, *Computer Graphics and Imaging*, Innsbruck, Austria, 123–128, 2013.

# Layered Reeb Graphs of a Spatial Domain[*]

B. Strodthoff [†], B. Jüttler[†]

**Abstract**

We introduce layered Reeb graphs as a representation for the topological structure of Reeb spaces and sketch a boundary-based algorithm for computing them.

## 1 Introduction

Reeb graphs are topological graphs originating in Morse theory, which represent the topological structure of a Riemannian manifold based on a scalar-valued, sufficiently smooth function defined on it (see [1] for an introduction). The use of more than one function leads to Reeb spaces, which are thus able to capture more features of an object. Reeb spaces were considered in 2008 [4], but appear to be little researched by now, especially Reeb spaces for manifolds with boundary. In the first part of this work, we introduce the layered Reeb graph as a discrete representation for Reeb spaces of 3-manifolds with respect to two scalar-valued functions. After that we present a restricted class of defining functions, for which the layered Reeb graph can be computed from a boundary representation of the spatial domain of interest. This leads to substantial computational advantages if the manifold is given in a boundary description, since no volumetric description has to be constructed.

## 2 Definition of the layered Reeb graph

After defining Reeb graphs and -spaces, we present an approach for computing Reeb graphs or -spaces by a sweep algorithm. This motivates the definition of the layered Reeb graph.

### 2.1 Reeb graphs and -spaces

Consider $n$ scalar-valued functions $f_1, \dots, f_n$ defined on a $d$-manifold (or manifold with boundary) $M$. Any subset of these functions defines *level sets*, where all these functions have constant values. Connected parts of $M$ on the same level set are called *level set components.*

**Definition 1** *Points where level set components of all functions $f_1, \dots, f_n$ or components of their boundary meet or disappear are called* critical, *all other points are regular. Contracting every level set component to a point gives the* Reeb space *of $M$ with respect to $f_1, \dots, f_n$. In the special case of $n = 1$ we speak of a* Reeb graph.

In the following, we will only consider three special cases of this definition:

- Reeb graph of a 2-manifold (Figures 1 and 2b),
- Reeb graph of a 3-manifold (Figures 2c-d), and
- Reeb space of a 3-manifold with respect to two functions (Figure 3).

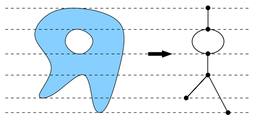


Figure 1: Reeb graph of a 2-manifold with boundary in the plane with respect to the height function, which maps each point to its last Cartesian coordinate.

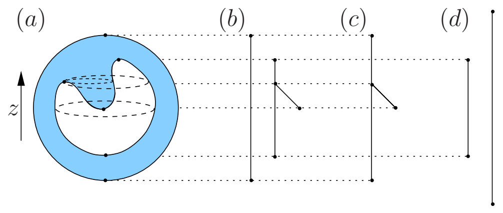


Figure 2: Reeb graphs of manifolds in space with respect to the height function. (a) The object: sphere with a bowl-shaped void. (b) Reeb graph of the surface considered as 2-manifold in space. (c) Reeb graph of the object. (d) Reeb graph of the air-volume surrounding the object.

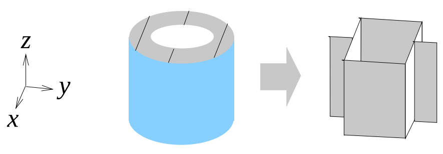


Figure 3: Reeb space of a vertical pipe with respect to the $z$ and $y$- coordinate. Middle: Lines in the horizontal cut show common level sets. Right: Structure of the resulting Reeb space.

[*]Supported by ESF Programme EuroGIGA-Voronoi

[†]Johannes Kepler University, Linz, Austria

### 2.2 Sweep algorithm for computing Reeb spaces

In the remainder of this paper, we consider two sufficiently smooth, scalar-valued functions $f$ and $g$ on $\mathbb{R}^3$, and a 3-manifold $M$ with boundary embedded in $\mathbb{R}^3$. We are interested in the Reeb space with respect to $(f_1, f_2)$ with $f_1 = f|_M$ and $f_2 = g|_M$.

Consider a level set of $f$ at function value $f = c$, and the restriction $g_c$ of $g$ onto this level set. The Reeb graph of the level set $f = c$ (considered as a 2-manifold with boundary) with respect to $g_c$ contracts all those points of the level set with the same $g_c$-value. We denote these Reeb graphs as *level set Reeb graphs.*

A point of a level set Reeb graph thus represents a connected component of points of $M$ which are mapped to the same value both by $f$ and $g$. Sweeping through the level sets of $f$ and continually connecting subsequent level set Reeb graphs gives the Reeb space of $M$ with respect to $(f, g)$.

This motivates the following *algorithm* for the computation of Reeb spaces: First, identify those points of $M$ where changes occur in the structure of level sets of $f$ or of their level set Reeb graphs. We will call these points *events* in the following. Then, sweep through the level sets of $f$. At each event, find the level set components in which changes occur, and update their level set Reeb graphs. These can be computed by a sweep inside the level set.

### 2.3 The layered Reeb graph

Consider the Reeb graph of $M$ with respect to $f$. An arc of this graph represents an evolving level set component of $f$. This component's level set Reeb graph goes through structural changes only at certain event points. In the layered Reeb graph, the original Reeb graph arc is divided at these events, and each part stores its level set Reeb graph, see Figure 4.

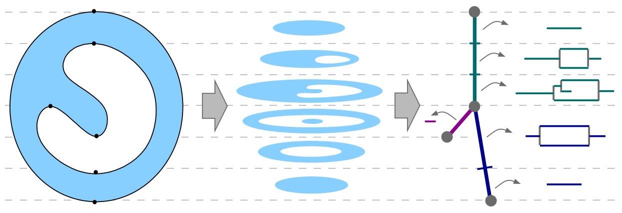


Figure 4: Layered Reeb graph. Left: vertical cut through a 3-manifold. Middle: level sets of $f$. Right: layered Reeb graph.

**Definition 2** *The layered Reeb graph of a 3-manifold $M$ with boundary with respect to two sufficiently smooth, scalar-valued functions $f$ and $g$ is obtained as follows. Take the Reeb graph with respect to the first function $f$ and subdivide each arc into parts of level set Reeb graphs with equivalent topological structure. Then enhance these parts by adding their level set's Reeb graphs with respect to $g$ as a secondary structure.*

## 3 Boundary-based computation of layered Reeb graphs

Several algorithms are described in the literature which compute the Reeb graph of a surface for a given surface description or the Reeb graph of a three-dimensional domain for a given volumetric description (like e.g. [8, 2, 6]). These algorithms typically allow for a rather general choice of defining functions. In this section we will restrict the defining functions such that the layered Reeb graph of a 3-manifold with boundary can be computed using only a boundary description of this manifold. This leads to computational advantages if the manifold is given in a boundary description, since no volumetric description has to be constructed. Additionally, it is thus possible to compute the layered Reeb graph of an unbounded manifold, for which the construction of a volumetric description may impose problems.

### 3.1 Feasible functions

The defining function for a Reeb graph has to meet two basic requirements:

(i) Firstly, it has to be possible to identify all the critical points, where structural changes in the level sets occur, using only function values on the boundary. This excludes the existence of local extrema or saddle points inside the considered manifold, see Figure 5.

(ii) Secondly, it should be possible to reconstruct the full structure of a level set knowing only the intersection of the level set with the manifold's boundary. This will become clearer in the following sections.

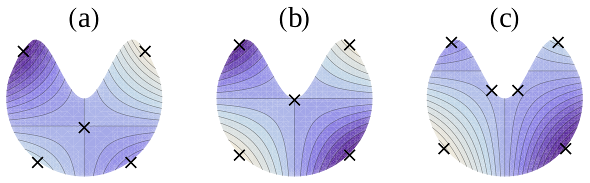


Figure 5: Contours of the function $(x, y) \to xy$ considered on planar 2-manifolds $M$ with boundary. Critical points are marked by crosses. (a) forbidden: the origin is a critical point inside $M$. (b) forbidden: the origin is a critical point, but a regular point on the boundary. (c) allowed: all critical points on $M$ are local minima or maxima on the boundary.

## Feasibility of the first function

The first function $f$ has to allow the computation of a Reeb graph of the 3-manifold $M$. For the Reeb graph of a 3-manifold, the considered level sets are, in general, surfaces.

Critical points (or curves) that are not induced by the boundary may occur where these level set surfaces touch each other or contract to a point. Since these cases are characterized by a zero gradient, we prescribe $\nabla f \neq 0$ to meet requirement (i).

The intersection of the level set surface with the manifold's boundary consists, in general, of closed curves. For requirement (ii), we need to be able to identify the relative positions of such boundary curves. Additionally, we need to know whether the inside or outside of such a curve is part of the considered level set. This can be deduced from the curve's orientation, provided that the boundary curves be computed in a certain orientation derived from the manifold's surface orientation. Using these two basic operations, the full structure of a level set can be reconstructed from the computed boundary curves. This is also a prerequisite of computing the Reeb graph of a level set component with respect to the second function.

This second requirement is fulfilled by functions for whose level sets a regular parametrization is known. Then, the two basic operations mentioned in the last paragraph can be reduced to operations in the level set's parameter domain. For instance, if another sufficiently smooth functions $h$ is given such that $\nabla g \times \nabla h = \lambda \ \nabla f$ for some scalar field $\lambda > 0$, then $g$ and $h$ can be used as parameter functions on any level set of $f$, see Figure 6a.

## Feasibility of the second function

The second function $g$ is used to compute the Reeb graph of a level set surface of $f$. So, for every value $c$ of $f$, the restriction $g_c$ of $g$ to the level set $f = c$ needs to meet the two basic requirements for the computation of a Reeb graph.

Consider requirement (i), i.e., that critical points are determined by the function values on the manifold's boundary. The level sets of $g_c$ are now, in general, curves. They are formed by the intersection of level set surfaces of $f$ and $g$, and thus have the tangential direction $\nabla f \times \nabla g$. Critical points inside the manifold occur if these level set curves touch or collapse to a point, which may happen only if the gradients of $f$ and $g$ are linearly dependent. So in order to meet the first requirement, we assume $\nabla f \times \nabla g \neq 0$.

For requirement (ii), we need to be able to reconstruct the structure of a level set from its boundary, e.g. in order to determine which level set component a critical point belongs to, while sweeping through the level sets of $g_c$. The level sets of $g_c$ consist of segments on a curve, and their boundaries are simply points. We consider again the auxiliary function $h$, which is then monotonic along the level set of $g_c$, see Figure 6b. Then the curve endpoints can be sorted by their $h$-values, and intervals between them form level set components.

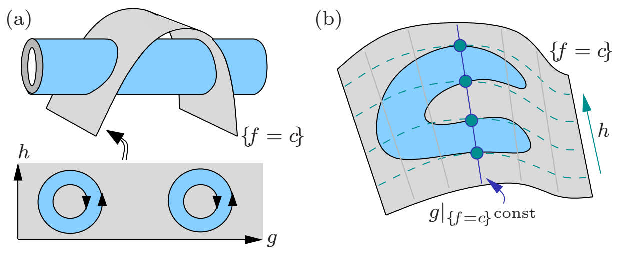


Figure 6: (a) Parametrization $(g, h)$ of $f$-level sets. (b) Monotonic function $h$ on level sets of $g|_{\{f=c\}}$.

## Summary

Summing up we arrive at the following observation.

**Lemma 1** *Assume $\nabla f \times \nabla g \neq 0$. Additionally, assume that another function $h$ is available such that $\nabla h \times \nabla g = \lambda \ \nabla f$ for some scalar field $\lambda > 0$. Then, the layered Reeb graph with respect to $f$ and $g$ is determined by function values of $f, g$ and $h$ on the boundary of $M$.*

**Proof.** According to the observations in the previous sections, critical points of $f$ and $g_c$ on $M$ that are not determined by function values on the boundary are excluded by $\nabla f \neq 0$ and $\nabla f \times \nabla g \neq 0$, respectively. Using the additional function $h$, the level sets of $f$ can be equipped with a regular parametrization $(g, h)$. Additionally, $h$ is monotonic along the level sets of $g_c$. So all in all, requirements (i) and (ii) are met both for $f$ and $g|_{\{f=c\}}$ $\square$

## 3.2 Relation to Jacobi sets

Consider the critical points induced by the boundary. A boundary point $p$ can only cause a change in the level set component if the functions' common level set touches the boundary in $p$ without intersection. With $N$ denoting the surface normal vector of $M$, these points are characterized by $(\nabla f \times \nabla g) \cdot N = 0$, since $\nabla f \times \nabla g$ is the tangent direction of the level set of $f$ and $g$. This leads to a connection to the Jacobi set of the functions' restrictions to the manifold's surface, as defined in [3].

**Definition 3 ([3])** *The* Jacobi set *of two smooth functions $\varphi$ and $\psi$ on a surface consists of all points with $\nabla\varphi \times \nabla\psi = 0$, with $\nabla$ denoting the gradient operator on the surface.*

We get the following relation between Reeb spaces and Jacobi sets.

**Lemma 2** *Let $\bar{f}$ and $\bar{g}$ denote the restrictions of $f$ and $g$ to the boundary of $M$. Then, the critical points of $(f, g)$ which are induced by the boundary of $M$ form the Jacobi set of $\bar{f}$ and $\bar{g}$.*

**Proof.** Let $\bar{\nabla}$ denote the gradient operator with respect to the boundary surface. For points on the Jacobi set, the gradients $\bar{\nabla}\bar{f}$ and $\bar{\nabla}\bar{g}$ are linearly dependent vectors in the boundary surface's tangent plane. Together with the surface normal vector $N$ they define a plane $\epsilon$, see Figure 7. Since $\nabla f$ and $\nabla g$ consist of $\bar{\nabla}\bar{f}$ and $\bar{\nabla}\bar{g}$, respectively, and components in the direction of $N$, they are also contained in $\epsilon$. Therefore, $(\nabla f \times \nabla g) \cdot N = 0$, which characterizes a critical point of $(f, g)$ induced by the boundary of $M$. The argumentation works analogously in the other direction. □

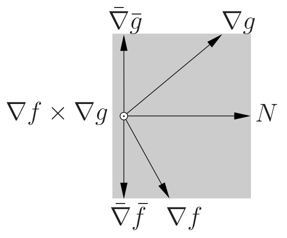


Figure 7: In critical points, gradients of $f$ and $g$ are coplanar with the surface normal vector $N$.

According to [3], the Jacobi sets of two functions defined on a surface are in general curves, so the critical points of $f$ and $g$ form curves on the manifold's surface. While sweeping through level sets of $f$, changes in the level set Reeb graphs thus occur at crossing points of these Jacobi curves, or at local extrema of the surface, which form monotonicity changes of Jacobi curves with respect to $f$. So the events to be considered in the sweep are crossing- and monotonicity-changing points of the Jacobi curves of $\bar{f}$ and $\bar{g}$.

## 3.3 Realization

The double-layered sweep algorithm sketched in the last paragraph of section 2.2 has been implemented for manifolds given by a triangular surface mesh, extending the algorithm for Reeb graphs as presented in [7]. We consider the piecewise linear approximations of the defining functions, as induced by the triangular mesh. If the mesh is fine enough compared to curvatures of the surface and the defining functions, the gradients of the piecewise linear approximation will approximate the smooth gradients sufficiently well.

In this setting, all critical points occur on edges of the surface mesh, so the Jacobi curves consist of mesh-edges. Using the criterion presented in [3], each edge is tested whether it is part of the Jacobi set or not. Additionally, boundary curves of the level sets of $f$ are polygons, which can be computed and handled efficiently.

## Conclusion

We presented a discrete representation for Reeb spaces of 3-manifolds with boundary with respect to two scalar-valued functions, the layered Reeb graph. Furthermore we introduced restrictions on the defining functions, which allow the layered Reeb graph to be computed from a boundary description of the manifold. In the next step, we seek to find a geometrically meaningful embedding of the Reeb space of a 3-manifold with respect to two functions into space. This leads to a topological skeleton of the 3-manifold, which promises to be more efficiently computable than, for example, the manifold's medial axis. The freedom in the choice of defining functions allows a customization of such a Reeb skeleton to give optimized results for given manifold.

## References

[1] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theor. Computer Science*, 392(1-3):5–22, 2008.

[2] H. Doraiswamy and V. Natarajan. Efficient algorithms for computing Reeb graphs. *Computational Geometry*, 42(6-7):606–616, 2009.

[3] H. Edelsbrunner and J. Harer. Jacobi sets of multiple Morse functions. *Foundations of Computational Mathematics, Minneapolis 2002*, pages 35–57, 2004.

[4] H. Edelsbrunner, J. Harer, and A. K. Patel. Reeb spaces of piecewise linear mappings. In *Proc. Sympos. on Comput. Geom.*, pages 242–250. ACM, 2008.

[5] H. Edelsbrunner and M. Kerber. Alexander duality for functions: the persistent behavior of land and water and shore. In *Proc. Sympos. on Comput. Geom.*, pages 249–258. ACM, 2012.

[6] G. Patané, M. Spagnuolo, and B. Falcidieno. A minimal contouring approach to the computation of the Reeb graph. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):583–595, 2009.

[7] B. Strodthoff, M. Schifko, and B. Jüttler. Horizontal decomposition of triangulated solids for the simulation of dip-coating processes. *Computer Aided Design*, 43:1891–1901, 2011.

[8] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Trans. on Visualization and Computer Graphics*, 15(6):1177–1184, 2009.

# Collapsing Rips complexes*

Dominique Attali† André Lieutier‡ David Salinas†

## Abstract

Given a finite set of points that samples a shape in Euclidean space, the Rips complex of the points provides an approximation of the shape which can be used in manifold learning. Indeed, it suffices to compute the proximity graph of the points to encode the whole Rips complex as the latter is an example of flag completion. Recently, it has been proved that the Rips complex reflects the homotopy type of the shape when sufficiently densely sampled by the points. Unfortunately, the Rips complex is generally high-dimensional. In this paper, we focus on the simplification of Rips complexes that approximate manifolds with the goal of reducing the dimension of the complex to the one of the manifold. We first propose an algorithm that iteratively applies elementary operations that preserve both the homotopy type and the property of the complex to be a flag completion and then show how our algorithm performs on real datasets.

## 1 Introduction

Manifold learning aims at recovering low-dimensional structures hidden in high-dimensional data [7]. An example of data might be a collection of $m$ by $m$ images of a rigid body taken under different poses. The collection of images can be thought of as a point cloud in $\mathbb{R}^{m\times m}$. Assuming the space of images is equipped with a reasonable metric (possibly the Euclidean metric), we expect the points to be distributed over a 6-dimensional manifold corresponding to the group of rigid displacements (e.g. rotations and translations). A manifold learning algorithm should be able, given as input the points, to output a topologically consistent representation of that manifold. Typically, the representation can encode a simplicial complex which, in the ideal case, is homeomorphic to the underlying manifold.

Given a finite set of points that samples a shape in Euclidean space, a classical approach for building an approximation of the shape consists in returning the Rips complex of the points [4]. Formally, the *Rips complex* of a set of points $P$ at scale $\alpha$ is the simplicial complex whose simplices are subsets of points in $P$ with diameter at most $2\alpha$. Recently, it has been proved that the Rips complex reflects the homotopy type of the shape, assuming the shape has a positive reach and is sufficiently densely sampled by the points [2]. The Rips complex has the computational advantage to be a flag completion: it suffices to compute its 1-skeleton to encode the whole complex. Unfortunately, the Rips complex is generally high-dimensional so that the true dimension of the shape remains elusive in the representation. To retrieve the intrinsic dimension of the shape, we propose to simplify Rips complexes by repeatedly applying generalized vertex and edge collapses (see definition bellow). We propose and compare several heuristics for finding such a sequence of collapses.

## 2 Collapses

Given a simplicial complex $K$, an *elementary collapse* is the operation that removes a pair of simplices $(\sigma_{\min}, \sigma_{\max})$ assuming $\sigma_{\max}$ is the unique proper coface of $\sigma_{\min}$. The result is the simplicial complex $K \setminus \{\sigma_{\min}, \sigma_{\max}\}$ to which $K$ deformation retracts. The reverse operation, which adds back the two simplices $\sigma_{\min}$ and $\sigma_{\max}$ is called an *elementary anti-collapse* and is clearly also a homotopy-preserving operation. A simplicial complex is said to be *collapsible* if it can be reduced to a single vertex by a finite sequence of elementary collapses. For instance, the closure of a simplex $\sigma$, $\mathrm{Cl}(\sigma) = \bigcup_{\emptyset\neq\tau\subset\sigma}\{\tau\}$, is collapsible. $\mathrm{Cl}(\sigma)$ is an example of cone. A *cone* is a simplicial complex $K$ which contains a vertex $o$ such that the following implication holds: $\sigma \in K \implies \sigma \cup \{o\} \in K$. Cones are also collapsible. Another elementary operation that we shall use is the *edge contraction*. The edge contraction $ab \mapsto c$ is the operation that identifies the two vertices $a \in K$ and $b \in K$ to the vertex $c$. It preserves the homotopy-type whenever $\mathrm{Lk}_K(ab) = \mathrm{Lk}_K(a) \cap \mathrm{Lk}_K(b)$ where $\mathrm{Lk}_K(\sigma) = \{\tau \in K, \tau\cap\sigma = \emptyset \text{ and } \tau\cup\sigma \in K\}$ designates the link of $\sigma$ in $K$ [1, 5].

We now list several possible generalizations of elementary collapses. To do so, we call the collection of simplices of $K$ having $\sigma$ as a face the *star* of $\sigma$ and denote it as $\mathrm{St}_K(\sigma)$. Finally, we call the operation that removes $\mathrm{St}_K(\sigma_{\min})$ from $K$:

*Research partially supported by the French "Agence nationale pour la Recherche" under grant ANR-09-BLAN-0331-01 "Giga".

†Gipsa-lab – CNRS UMR 5216, Grenoble, France. Firstname.Lastname@gipsa-lab.grenoble-inp.fr

‡Dassault systèmes, Aix-en-Provence, France. andre.lieutier@3ds.com

• a *(classical) collapse*: if the star of $\sigma_{\min}$ has a unique inclusion-maximal element $\sigma_{\max} \neq \sigma_{\min}$ ; the reverse operation is called a *(classical) anti-collapse*.

• an *(extended) collapse*: if the link of $\sigma_{\min}$ is a cone [1]; the reverse operation is called an *(extended) anti-collapse*.

• a *(generalized) collapse*: if the link of $\sigma_{\min}$ can be reduced to a point by a sequence of collapses, anti-collapses and homotopy-preserving edge contractions; the reverse operation is called a *(generalized) anti-collapse*.

All three collapses (classical, extended, generalized) preserve the homotopy-type. Deciding whether the operation that removes $\mathrm{St}_K(\sigma_{\min})$ from $K$ is a classical or extended collapse can be done efficient (*i.e. in polynomial time*) using the data structure described in [1]. We will see shortly that deciding whether the operation that removes $\mathrm{St}_K(\sigma_{\min})$ from $K$ is a generalized collapse is computationally more involved. Here, we will focus on Rips complexes which will allow us to design specific reduction sequence.

## 3 Simplifying Rips complexes whoses vertices approximate a manifold

The goal of this section is to present and compare strategies for simplifying a Rips complex whose vertex set samples a manifold. In the ideal case, we would like to get a complex homeomorphic to the manifold or at least whose dimension is as close as possible to that of the manifold. Throughout the section, we will assume that $P$ is a point cloud that samples a $d$-dimensional manifold $A$ embedded in $D$-dimensional Euclidean space and suppose that we can find a value of $\alpha$ such that $\mathrm{Rips}(P, \alpha)$ and $A$ has the same homotopy type.

### 3.1 Simplification algorithm

The Rips complex is simplified in two stages: the first stage iteratively collapses vertices and the second stage iteratively collapses edges; see Algorithm 1. During the simplification, the complex remains a flag completion, since this property is not altered by collapsing vertices and edges. For $k \in \{0, 1\}$, stage $k$ proceeds as follows. Initially, all $k$-dimensional simplices are stored in a priority queue $Q$. Each $k$-simplex receives as priority its diameter. During stage $k$, we iteratively take the $k$-simplex $\sigma$ with highest priority and remove it from the current complex $K$ together with all its cofaces whenever REDUCIBLE($\mathrm{Lk}_K(\sigma)$) returns true; see Algorithm 2. Ideally, we would like the function REDUCIBLE($\mathrm{Lk}_K(\sigma)$) to be true if and only if the operation that removes $\sigma$ and all its cofaces is a generalized collapse. This means that ideally, we would like the function REDUCIBLE to take as input a simplicial complex $L$ and returns true whenever there exists a sequence of homotopy-preserving elementary operations (collapses, anti-collapses and edge-contractions) that goes from $L$ to a point and false otherwise. Unfortunately, the problem of deciding whether a complex $L$ is reducible to a point by a sequence of elementary operation is NP-complete, even when we limit ourselves to elementary collapses and 3-dimensional complexes [6]. Instead, we will propose four more or less sophisticated heuristics to find such a sequence, drawing inspiration from the constructive proofs of [3] and sometimes taking advantage of the fact that $L$ is a flag complex.

**Algorithm 1** SIMPLIFY(Simplicial complex $K$)

```
SIMPLIFY($K$,0,**true**) {Collapse the vertices of $K$}
SIMPLIFY($K$,1,**false**) {Collapse the edges of $K$ }
```

**Algorithm 2** SIMPLIFY(Simplicial complex $K$, integer $k$, boolean reinsert)

```
$Q = K^{(k)} \setminus K^{(k-1)}$
while $Q \neq \emptyset$ do
  Remove the simplex $\sigma$ from $Q$ with highest priority
  if REDUCIBLE ($\mathrm{Lk}_K(\sigma)$) then
    $K = K \setminus \mathrm{St}_K(\sigma)$
    if reinsert then
      Insert in $Q$ the $k$-dimensional simplices whose link have changed
    end if
  end if
end while
```

Observe that the only difference between the two stages is that at stage 0, when we collapse a vertex, we reintroduce the vertices of its link in the priority queue. We do not do the same at stage 1 because, in our experiments, it slows the computation and does not improve the number of times we get a complex either homeomorphic to or with the same dimension as the manifold $A$.

### 3.2 Finding reduction sequences

We present four possible procedures that can be used in place of REDUCIBLE in Algorithm 2. Each procedure takes as input a simplicial complex $L$. The third strategy is the only one that requires its input to be a flag completion. The pseudo-code for each procedure can be found either in [1] or [3]. For later reference, strategies are numbered from (S1) to (S4). The intuition behind strategies (S2) and (S3) is that if the vertices of $L$ sample a convex set densely enough then results in [3] ensure that those two strategies will succeed.

(S1) ISCONE: returns true if and only if $L$ is a cone.

(S2) REDUCIBLE_BY_SWEEP: starts by picking a vertex $o$ of $L$ and tries to apply vertex and edge extended collapses to reduce $L$ to $o$. To do so, the strategy computes a priority $\varphi$ which is evaluated to $\varphi(a) = d(o, a) - \alpha$ for each vertex $a$ of $L$ and $\varphi(ab) = d(o, B(a, \alpha) \cap B(b, \alpha))$ for each edge $ab$ of $L$. Then, we put all vertices of $L$ except $o$ in a priority queue $Q$ and put all edges $ab$ of $L$ in $Q$ iff the priority of $ab$ is greater than the priority of its vertices $a$ and $b$ that is $\varphi(ab) > \min\{\varphi(a), \varphi(b)\}$. Finally the strategy tries to reduce $L$ to $o$ by performing extended collapses of all simplices in $Q$ in the order of decreasing priority $\varphi$. The strategy returns true iff it manages to do so.

(S3) REDUCIBLE_BY_COMPLETION: applies a sequence of edge extended anti-collapses in the order of increasing length. If at some point, the result is a cone, returns true. If at some point, an edge could not be inserted, returns false.

(S4) REDUCIBLE_BY_EDGE_CONTRACTIONS: simplify $L$ by applying a sequence of edge contractions $ab \mapsto \frac{a+b}{2}$ in the order of increasing length assuming $\mathrm{Lk}_L(ab) = \mathrm{Lk}_L(a) \cap \mathrm{Lk}_L(b)$ as explained in [5, 1]. Returns true iff the simplex $L$ after simplification consists of a single vertex.

If we assume that $P$ is initially a dense sampling of $A$, the vertices in the link of a simplex are likely to be close to a convex (at least at the beginning of the simplification). In this situation, it is proved that the complex is always reduced to a point with the two strategies (S2) and (S3) [3]. We now describe various computational experiments and the results we obtained.

## 4 Experiments

### 4.1 Data-sets

We present the four data-sets used in our experiments.

`Cat`. A collection of 72 images of a toy cat placed on a turntable and observed by a fixed camera. Each image has size $128^2 = 16384$ and can be thought of as a point-cloud that samples a 1-manifold in $\mathbb{R}^{16384}$. For this data-set, $d = 1$ and $D = 16384$.

`Sphere`. A sampling of a 2-sphere with 2646 points corrupted by noise. In this case, $d = 2$ and $D = 3$.

`Ramses`. A 3D scan data consisting of 193252 points measured on the surface of a statue representing Ramses II. The surface of the statue is homeomorphic to $\mathbb{S}^2$. For this data-set, $d = 2$ and $D = 3$.

`SO3`. A point set `SO3` $\subset \mathbb{R}^9$ with size 10000 that samples the special orthogonal group. Recall that this group is diffeomorphic to $\mathbb{RP}^3$ which is a 3-dimensional manifold that can be embedded in $\mathbb{R}^9$ by representing each rotation in 3D by a $3\times 3$ matrix. We have $d = 3$ and $D = 9$.

| $P$ | $\sharp P$ | size of $\mathrm{Rips}(P,\alpha)$ | dimension of $\mathrm{Rips}(P,\alpha)$ |
|---|---|---|---|
| `Cat` | 72 | $> 10^6$ | 19 |
| `Sphere` | 2646 | $> 2 \times 10^9$ | $> 12$ |
| `Ramses` | $193,252$ | $> 6 \times 10^6$ | 14 |
| `SO3` | $10,000$ | $> 2.8 \times 10^8$ | 16 |

Table 1: For each data-set $P$, we indicate the number of points in $P$, the number of simplices in $\mathrm{Rips}(P,\alpha)$ and the dimension of $\mathrm{Rips}(P,\alpha)$.

Table 1 gives for each data-set the number of points, the number of simplices in $\mathrm{Rips}(P,\alpha)$ and the dimension of $\mathrm{Rips}(P,\alpha)$.

### 4.2 Results

To compare our four strategies, we first perform the following experiment. We apply our simplification algorithm (Algorithm 2), using for REDUCIBLE the function which returns true iff one of the four strategies returns true. Let $\sigma_j$ be the $k$-simplex removed at step $j$ during the simplification. Let $s_k^x(i)$ be the number of times (S$x$) returns false when applied to $\sigma_j$ for $j$ ranging over $\{1, \ldots, i\}$. In other words, $s_k^x(i)$ counts the number of times the strategy (S$x$) has failed to find a sequence of reduction while another strategy had succeeded during the $i$ first steps of the simplification process. In Figure 1, we plot $s_0^x$ for $x \in [1, 4]$ (that is for all strategies), and for all data-sets. When collapsing edges, all strategies give the same answer for all data-sets except for `Ramses` and `SO3` thus we plot $s_1^x$ only for these two data-sets, see Figure 1. We observe that (S4) seems to be the most efficient strategy : it finds a sequence of reduction whenever another strategy finds one when simplifying our four data-sets.

We now use our simplification algorithm with a fixed strategy for finding reduction sequences. In Table 2, we describe the complex $K_{\mathrm{out}}$ obtained after simplification using each of our 4 strategies in turn to find reduction sequences. As suggested by the previous experiment, the best results are obtained when using (S4). Indeed, using (S4), the result of the simplification $K_{\mathrm{out}}$ is a flag complex homeomorphic to the sampled manifold $A$, except for the data-set `SO3`. Still, in that case, we get a complex with the correct dimension. Future work will include a better understanding of the performances of strategy $(S4)$ together with the search of a condition ensuring that our algorithm outputs a complex homeomorphic to $A$.

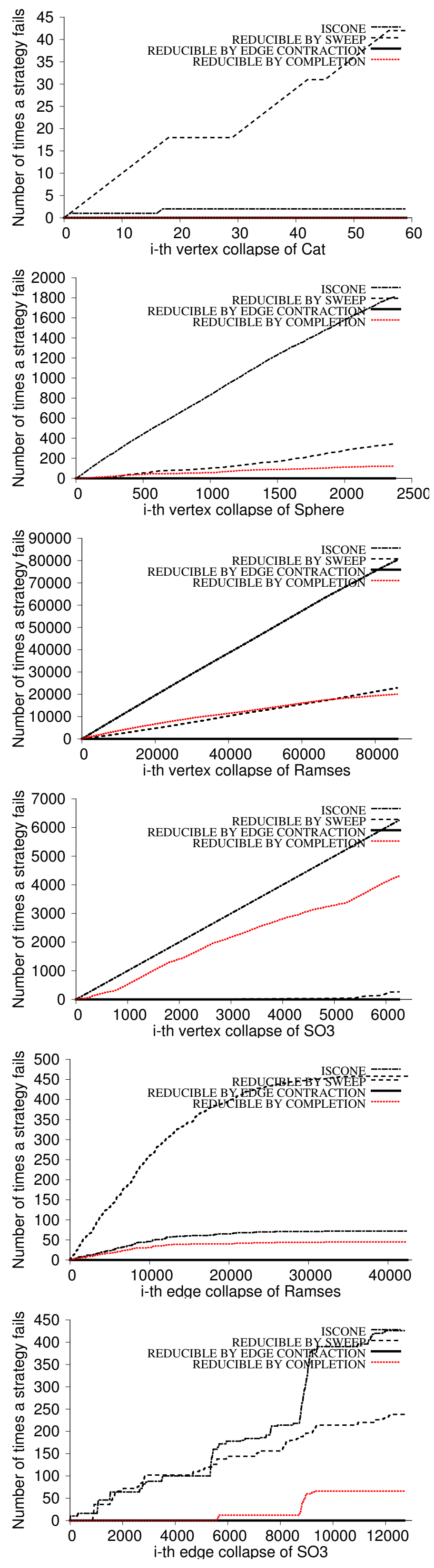


Figure 1: Number of failures for each strategy while performing vertex collapse (first four figures) and edge collapse (last two figures).

| **(S1)** | dim($K_{\text{out}}$) | $K_{\text{out}} \approx A$ | running time |
|---:|---:|---:|---:|
| `Cat` | 1 | YES | 1 s |
| `Sphere` | 5 | NO | 1 min |
| `Ramses` | 3 | NO | 25 min |
| `SO3` | 6 | NO | 94 s |
| **(S2)** | dim($K_{\text{out}}$) | $K_{\text{out}} \approx A$ | running time |
| `Cat` | 2 | NO | 3 min |
| `Sphere` | 3 | NO | 6 min |
| `Ramses` | 3 | NO | 180 min |
| `SO3` | 4 | NO | 10 min |
| **(S3)** | dim($K_{\text{out}}$) | $K_{\text{out}} \approx A$ | running time |
| `Cat` | 1 | YES | 1 s |
| `Sphere` | 2 | YES | 2 min |
| `Ramses` | 2 | NO | 160 min |
| `SO3` | 4 | NO | 33 min |
| **(S4)** | dim($K_{\text{out}}$) | $K_{\text{out}} \approx A$ | running time |
| `Cat` | 1 | YES | 2 s |
| `Sphere` | 2 | YES | 2 min |
| `Ramses` | 2 | YES | 150 min |
| `SO3` | 3 | NO | 7 min |

Table 2: Description of the simplicial complex output $K_{\text{out}}$ when using each of the four strategies for finding sequences of reductions. We indicate the dimension of $K_{\text{out}}$ together with the fact that it is homeomorphic or not to $A$ and the computation time. All computation are done with a 2.8 GHz processor and 8 GB RAM.

## References

[1] D. Attali, A. Lieutier, and D. Salinas. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. *International Journal of Computational Geometry and Applications (IJCGA)*, 22(4):279–303, 2012.

[2] D. Attali, A. Lieutier, and D. Salinas. Vietoris-Rips complexes also provide topologically correct reconstructions of sampled shapes. *Computational Geometry: Theory and Applications (CGTA)*, 2012.

[3] D. Attali, A. Lieutier, and D. Salinas. Collapsing rips complexes. In *29th Ann. Sympos. Comput. Geom.*, Rio de Janeiro, Brazil, 2013. Submitted.

[4] V. de Silva and R. Ghrist. Coverage in sensor networks via persistent homology. *Algebraic & Geometric Topology*, 7:339–358, 2007.

[5] T. K. Dey, H. Edelsbrunner, and S. Guha. Computational topology. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*. AMS, Providence, 1999.

[6] M. Tancer. Recognition of collapsible complexes is np-complete. *CoRR*, abs/1211.6254, 2012.

[7] J. Tenenbaum, V. De Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

# Constructing Complicated Spheres

Mimi Tsuruga* Frank H. Lutz*

**Abstract**

Fast and efficient homology algorithms are in demand in the applied sciences for analyzing solid materials and proteins, processing digital imaging data, or pattern classification among others. Recent advances employ discrete Morse theory as a preprocessor. Research in this area has lead to the need to find complicated test examples. We present an infinite series of examples that have been constructed to test some of the latest algorithms under development. This family of 4-spheres (known as the Akbulut–Kirby spheres) is based on a handlebody construction via finitely presented groups.

## 1 Motivation

### 1.1 Homology Algorithms

Computing the homology of simplicial or more general cell complexes can quickly become too cumbersome by hand. Today, however, computers can be used to analyze even amazingly large complexes [5, 6]. Speeding up homology calculations is now an active field of research in computational topology.

Computing homology involves the computation of the Smith normal form of matrices that encode geometric information about the complex. Current algorithms for computing the Smith normal form, though polynomial [13], are too slow for explicit computations on large examples. However, computation time can be dramatically improved by employing a topological preprocessor.

Discrete Morse theory provides an excellent foundation for reducing a large complex to a simplified complex of the same topological (PL) type [10, 12]. We can then replace the matrix for which we want to find the Smith normal form with a much smaller one. The problem of finding an optimal discrete Morse function, however, is $\mathcal{NP}$-hard [11, 14].

Benedetti and Lutz [3] propose using random discrete Morse theory to search for discrete Morse functions with few critical cells. Surprisingly, this approach frequently finds the optimum even for large examples. In fact, it has been difficult to provide complicated examples on which the random heuristics performs poorly.

*Technische Universität Berlin

The aim of this article is to provide an infinite series of such examples.

### 1.2 Complicated Spheres

Simplicial complexes can pose challenges for discrete Morse algorithms for various reasons. For example, if the underlying topological space is complicated or if the triangulation itself is complicated. Here we look at the latter case and construct a series of triangulations of the 4-dimensional sphere $S^4$, which has trivial topology.[1] The examples in the series, however, are composed in a rather non-trivial manner.

## 2 Akbulut–Kirby Spheres

### 2.1 Background

Topologists in recent years have been working on finding counterexamples for the smooth Poincaré conjecture in dimension 4. These so-called exotic spheres are smooth manifolds that are homeomorphic, but not diffeomorphic to the standard sphere $S^4$, i.e., the round sphere with the usual differentiable structure. Various candidates have been introduced, many of which were later shown to be standard.

We have built explicit triangulations of one such candidate series proposed by Akbulut and Kirby [2], which goes back to an example by Cappell and Shaneson. Gompf [9] showed that one sphere from the series is standard, i.e., diffeomorphic to $S^4$. Akbulut [1] later proved that all examples in the whole infinite series are standard.

In the following section, we will give a detailed description of our construction of triangulations of the Akbulut–Kirby spheres.

### 2.2 Construction

Before going into the details of our construction of triangulations for the Akbulut–Kirby spheres, let us start with a quick overview of what's to come. The main idea for the spheres begins with finitely presented groups

$$G = \langle g_1, \ldots, g_l \mid r_1, \ldots, r_m \rangle ,$$

where $g_i$ are generators and $r_i$ are relators in the $g_i$'s. It is undecidable whether a given finitely presented

[1]We study spheres because spheres can admit perfect Morse functions, i.e., with only two critical cells.

group is the trivial group [16]. In fact, there are even group presentations with only two generators and two relators that are not obviously trivial.

Let $G = \langle x, y \mid xyx = yxy; x^r = y^{r-1} \rangle$. To see that this group is trivial, rearrange the first relator to get

$$y = x^{-1}y^{-1}xyx.$$

Then $y^r = x^{-1}y^{-1}x^r yx = x^{-1}y^{r-1}x = x^r$ by substituting the second relator twice. This yields $y^r = y^{r-1}$ or $y = e$ and hence $x = e$. Thus $G$ is the trivial group.

Next, take a 5-ball and attach two 1-handles representing each of the generators $x$ and $y$. Then glue in two 2-handles along thickened curves representing the relators $xyx = yxy, x^r = y^{r-1}$. The resulting object is a topological 5-ball.

Gompf [9] proved that for all the presentations $G = \langle x, y \mid xyx = yxy, x^r = y^{r-1} \rangle$ the resulting boundary 4-spheres are, in fact, standard 4-spheres.

We built simplicial complexes following this recipe. But we begin our construction in dimension 3.

Start with a 3-ball and attach two 1-handles. Then choose two curves in the resulting handlebody that represent the relators. Next we take the product of the 3-dimensional handlebody $H^3$ with an interval $I$ twice. We then find (copies of) the two curves in the boundary of the 5-dimensional handlebody $H^3 \times I \times I$.

Consider Fig. 1 in which the red curve lies in the interior of the handlebody $H^3$. After taking the product $H^3 \times I$, we find (a copy of) the red curve on the boundary of $H^3 \times I$. On the boundary of $H^3 \times I \times I$ we thicken each of the two curves representing the generators $xyx = yxy$, $x^r = y^{r-1}$ to a solid 4-torus along which the two 2-handles are glued in. While the two curves are twisted in $H^3$, the two solid 4-tori untwist in the boundary of $H^3 \times I \times I$.

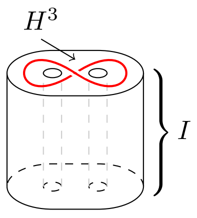


Figure 1: The red curve in the interior of $H^3$ is on the boundary of $H^3 \times I$.

By gluing in the two 2-handles, we end up with a topological 5-ball whose boundary (if smoothed appropriately) is a homotopy 4-sphere [2]. For 4-manifolds, the categories DIFF (differentiable) and PL (piecewise linear) coincide, so our triangulations provide model spaces for the smooth setting.

## Step 1: The Space.

We begin in dimension 3. In particular, we want to have a 3-ball with two 1-handles which represent our two generators $x$ and $y$, see Fig. 2.

Notice that we have used three colors to distinguish the different parts of our space. Keep this picture in mind when we move on to the next steps. The green section in the center is our 3-ball. To this 3-ball we attach one 1-handle which we label $x$ (in purple) and another 1-handle which we label $y$ (in orange).

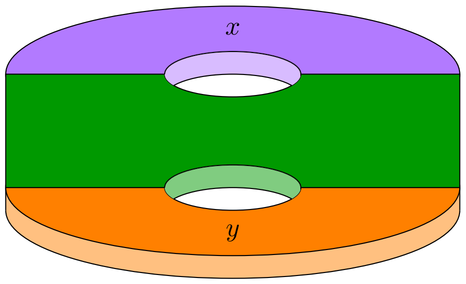


Figure 2: The handlebody $H^3$ consisting of a 3-ball (in green) with two 1-handles (purple for the generator $x$ and orange for the generator $y$).

We will choose two curves that represent the two relators inside this space.

## Step 2: Planning the Curves.

We draw the two curves which represent the relators. The two curves are $xyx = yxy$, which we call the blue curve, and $x^r = y^{r-1}$, which we call the red curve (see Fig. 3 for the Akbulut–Kirby sphere with $r = 5$).

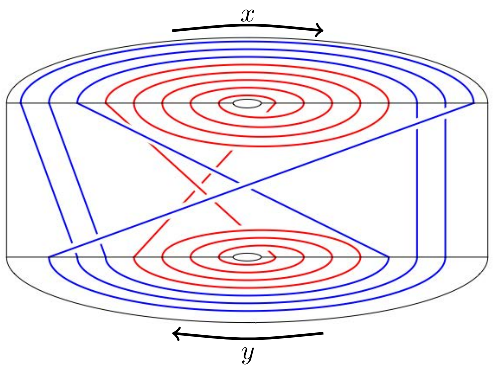


Figure 3: The two curves of the Akbulut–Kirby sphere $xyx = yxy$ (blue) and $x^5 = y^4$ (red) in dimension 3.

The crossings indicate which sections of the curves run over or under. The embedding of the curves can be arbitrarily chosen as long as they run over the two 1-handles $x$ and $y$ as specified by the relators. When we move up in dimension in Step 6, the thickened curves will untangle no matter how twisted or knotted they were in dimension 3. The particular arrangement shown in Fig. 3 was chosen to simplify the construction in Step 5.

To understand Fig. 3, begin with the blue curve read as $xyxy^{-1}x^{-1}y^{-1} = e$. Remember that we are on a 3-ball with two 1-handles $x$ and $y$. These handles are oriented as indicated by the black arrows.

For the blue curve, we begin at the blue crossing in the center of Fig. 3. Follow the bottom curve at the crossing and go up (and left) towards the $x$-handle. Then along $x$, back through the ball, then $y$, then $x$, then all the way across to the far bottom left, along

$y$ in the reverse direction, then $x$ in reverse, then $y$ in reverse and then finally return to our starting point to close the curve. The red curve $x^5y^{-4} = e$ is represented similarly.

## Step 3: Thickening of the Curves to Solid Tori.

Fig. 3 displays the blue and red curves in $H^3$ which will be thickened to solid 3-tori. We choose triangular prisms to compose the solid 3-tori. In Fig. 4, we have

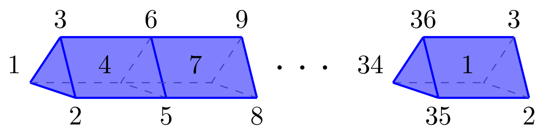


Figure 4: The blue curve in Fig. 3 is thickened to a chain of triangular prisms.

the prisms for the blue curve for the Akbulut–Kirby sphere with $r = 5$ (Fig. 3). The vertex labels repeat at the ends as they are identified to form a loop.

Notice that the blue curve in Fig. 3 can be cut up into 12 strands (3 on the $x$-handle, 3 on the $y$-handle, and 6 inside the middle ball) every time the curve crosses into a different section of the handlebody (recall Fig. 2). One triangular prism is used for each of these strands. So we used $(12 \times 3 =)36$ vertices to build the blue solid torus as indicated in Fig. 4.

We can systematically and coherently break down these prisms into tetrahedra by using product triangulations as described in [15] and references therein. This method guarantees that neighboring simplices match up nicely. For example, we do not want to introduce different diagonals for a rectangular face of one of the prisms.

## Step 4: Fill the Handles.

The construction of the 1-handles is simple since they only have triangular prisms that run parallel along each other. However, we want to avoid having any unwanted identifications along the handles. So we add buffers between neighboring triangular prisms using rectangular prisms. Fig. 5 gives a front view of the

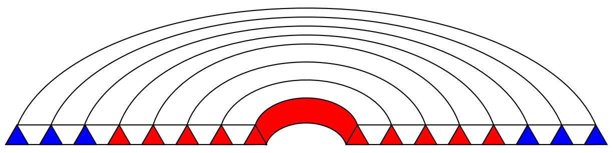


Figure 5: Filling the $x$-handle of Fig. 3.

$x$-handle. The three blue triangles on the two ends are part of the blue solid 3-torus running over the $x$-handle in Fig. 3. The rectangular prisms between the triangular prisms are white. We need 7 rectangular buffer prisms for the $x$-handle and 6 rectangular buffer prisms for the $y$-handle.

## Step 5: Fill the Ball.

First we explain the reason for choosing the particular arrangement of the curves in Fig. 3. Let's look only at the rectangular area representing what will become our 3-ball, see Fig. 6.

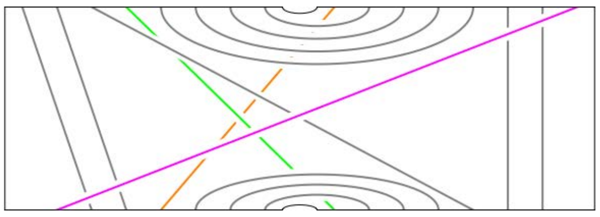


Figure 6: The portion that will be filled to become the 3-ball. The three colored lines go above or below.

The (curves that represent) prisms in the middle ball section are not all parallel and some cross each other. The gray lines have no crossings and we place them in, say, Level 0. We then let the pink line go above (Level +1), green goes below (Level -1) and orange further below (Level -2).

To fill the floors, we first glue in rectangular buffer prisms between the parallel strands of the ground floor. Then we glue in 2-dimensional (triangulated) membranes to close the holes between the remaining strands of the ground floor, then two additional membranes to close an upper cupula including the pink strand and further membranes to obtain another two cavities for the two lower floors. The three cavities are then closed by filling in three cones.

The main part of our construction is now complete.

## Step 6: Go Up In Dimension.

Having composed $H^3$, we can now take a direct product of it with the unit interval $I = [0, 1]$ once to go to 4-dimensions. In this step, we thicken the 3-dimensional solid tori to 4-dimensional solid tori. We then take the direct product again to go to 5 dimensions. This time, we keep the solid 4-tori in one of the two boundary components. The product triangulation procedure [15] is used each time.

## Step 7: Glue In the 2-Handles.

We glue in the two 2-handles along the two solid 4-tori in a canonical way by using four additional vertices each.

Now we have a 5-dimensional ball.

## Step 8: Take Its Boundary.

Finally, we take the boundary to obtain our (homotopy) 4-sphere. Notice that all the vertices have been pushed out onto the boundary at Step 6.

The final steps use techniques that are mostly canonical and will be described in more detail in a full version of this article (in preparation).

## 3 Results

**Theorem 1** *The series of Akbulut–Kirby spheres can be triangulated with face vectors* $f = (176 + 64\,r, 2390 + 1120\,r, 7820 + 3840\,r, 9340 + 4640\,r, 3736 + 1856\,r)$ *for* $r \geq 3$, *where the triangulations respect the defining handlebody decompositions as described in the above construction.*

As we hoped, these triangulated spheres have shown to be substantially complicated. For example for $r = 5$, the best discrete Morse vector found, $(1, 2, 4, 2, 1)$, was reached only 5 times out of 1,000 random runs. On average, we certainly need more than just two critical cells; see Table 1.

| $r$ | $c_\mathcal{T}$ | $r$ | $c_\mathcal{T}$ |
|---|---|---|---|
| 5 | 29.26 | 8 | 40.07 |
| 6 | 33.124 | 9 | 42.432 |
| 7 | 37.026 | 10 | 46.946 |

Table 1: Average number of critical cells $c_\mathcal{T}$ needed for $r = 5, \ldots, 10$ in 1,000 runs.

It is important to note that there does not exist an algorithm that can determine whether a (triangulated) manifold of dimension $d > 4$ is a sphere and existence of such an algorithm is still open for $d = 4$. One heuristic way, however, is to perform (random) bistellar flips, or Pachner moves [4, 17] to show that a given triangulation is PL-equivalent to the boundary of a $(d+1)$-simplex. We have shown using `polymake` [8] that at least for $r = 3$ the corresponding Akbulut–Kirby sphere is standard thus reconfirming the statement in this case experimentally.

Another remarkable result of our experiments is that the initial nontrivial group presentation can show up when we compute the fundamental group as the edge-path group according to Seifert and Threlfall [19] and then simplifying the presentation using `GAP` [7]. In fact, we found that the fundamental group of the complex for $r = 5$ after some bistellar and (FP) group simplification had two generators $x$ and $y$ and two relators $xyx = yxy$ and $x^5 = y^4$. This is exactly the presentation with which we started!

These spheres have recently been used as test examples for the Perseus homology algorithm by Mischaikow and Nanda [18]. The techniques designed to generate these spheres will also be used to construct other PL versions of topologically interesting objects.

## Acknowledgments

Many thanks to John M. Sullivan for helpful discussions and remarks. We are grateful to Joel Hass and to William P. Thurston for directing us to the Akublut–Kirby spheres as non-trivial constructions of 4-spheres.

## References

[1] S. Akbulut. Cappell-Shaneson homotopy spheres are standard. *Ann. Math. (2)*, 171:2171–2175, 2010.

[2] S. Akbulut and R. Kirby. A potential smooth counterexample in dimension 4 to the Poincaré conjecture, the Schoenflies conjecture, and the Andrews–Curtis conjecture. *Topology*, 24:375–390, 1985.

[3] B. Benedetti and F. Lutz. Random discrete Morse theory I: Complicatedness of triangulations. In preparation.

[4] A. Björner and F. H. Lutz. Simplicial manifolds, bistellar flips and a 16-vertex triangulation of the Poincaré homology 3-sphere. *Exp. Math.*, 9:275–289, 2000.

[5] CAPD::RedHom. `http://redhom.ii.uj.edu.pl`.

[6] CHomP. `http://chomp.rutgers.edu`.

[7] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.12*, 2008. `http://www.gap-system.org`.

[8] E. Gawrilow and M. Joswig. `polymake`: a framework for analyzing convex polytopes. In G. Kalai and G. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000.

[9] R. Gompf. Killing the Akbulut–Kirby 4-sphere, with relevance to the Andrews–Curtis and Schoenflies problems. *Topology*, 30:97–115, 1991.

[10] M. Joswig. Computing invariants of simplicial manifolds. `arXiv:math.AT/0401176`, 2004, 13 pages.

[11] M. Joswig and M. Pfetsch. Computing optimal Morse matchings. *SIAM J. Discr. Math.*, 20:11–25, 2006.

[12] T. Kaczynski, K. Mischaikow, and M. Mrozek. *Computational Homology.* Applied Mathematical Sciences vol. 157. Springer-Verlag, New York, NY, 2004.

[13] R. Kannan and A. Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8:499–507, 1979.

[14] T. Lewiner, H. Lopes, and G. Tavares. Optimal discrete Morse functions for 2-manifolds. *Comput. Geom.*, 26:221–233, 2003.

[15] F. Lutz. Triangulated Manifolds with Few Vertices: Geometric 3-Manifolds. `arXiv:math.GT/0311116`, 2003, 48 pages.

[16] P. S. Novikov. *On the algorithmic unsolvability of the word problem in group theory*, volume 44 of *Trudy Matematicheskogo Instituta imeni V. A. Steklova.* Izdatel'stvo Akademii Nauk SSSR, Moscow, 1955.

[17] U. Pachner. P.L. homeomorphic manifolds are equivalent by elementary shellings. *Eur. J. Comb.*, 12:129–145, 1991.

[18] Perseus. `http://www.math.rutgers.edu/~vidit/perseus.html`.

[19] H. Seifert and W. Threlfall. *Lehrbuch der Topologie.* BG Teubner, 1934.

# Selecting a Small Covering from a Double Covering

Peter Brass*

**Abstract**

The main result is that from any double covering of the plane by unit discs we can select a subset using at most 3/4 of the discs that still forms a covering.

## 1 Introduction

A $k$-fold covering of the plane is a family of sets such that each point of the plane belongs to at least $k$ of the sets. Such multiple coverings have been studied in discrete geometry at least since 1962 [7], see [5] section 2.1, and recently also found interest in the theory of sensor- and ad-hoc networks [2, 3, 8, 10, 11, 17, 18]. Reduction of multiple coverings of the plane by unit discs, or more generally by translates of a convex body, has up to now been studied only as a decomposition problem: Given a multiple covering, of sufficiently high multiplicity, can it be decomposed into two subfamilies which are still coverings? This problem has been answered for some classes of convex sets, most recently in [15] (improving [13, 14, 1, 16]), where it is showed that for any convex polygon $P$ there is a $k(P)$ such that any $k(P)$-fold covering by translates of $P$ can be decomposed in two coverings (the multiplicity depends on the shape of $P$, and is not even known to be bounded for, e.g., all fourgons). But the most natural case of the circle must be considered still open, for the much-cited manuscript [12] is still unpublished. It is the aim of this note to analyze selection instead of decomposition: Given a multiple covering, how small a subset can we select that is still a covering. We give bounds for double coverings and a corresponding result for double packings of unit discs.

## 2 Results

**Theorem 1** *From any given family of unit discs that is a double covering of the plane, we can select a subfamily using at most $\frac{3}{4}$ of the discs that is still a covering.*

To prove this theorem, we dualize the setting and consider the set $C$ of centers of the unit discs of the double covering. This is then a set of points such that each unit disc around any point $p$ in the plane contains at least two points of $C$. We consider the Delaunay-Triangulation of $DT(C)$. Since each unit disc contains at least two points, we can shrink that disc until it contains exactly two points of $C$, which define a Delaunay-edge. So each unit disc in the plane contains an edge of $DT(C)$. Since $DT(C)$ is planar, we can four-color it, and remove the largest color-class. By this we remove at most one of the two endpoints of each edge; so each unit disc still contains at least one of the remaining points. Thus we found a subset of $C$, so that the unit discs around that subset still cover the plane, and the subset contains at most $\frac{3}{4}$ of the original points.

A concept related to multiple coverings are multiple packings: a $k$-fold packing in the plane is a family of sets such that no point of the plane belongs to more than $k$ sets of the family. The double covering result has a 'dual' for double packings.

**Theorem 2** *From any given family of unit discs that is a double packing in the plane, we can select a subfamily using at least $\frac{1}{4}$ of the discs that is still a packing.*

The proof is similar: the set of centers $C$ of the unit discs in a double packing is a set of points with the property that any unit disc in the plane contains at most two points of $C$. We again use the Delaunay-Triangulation; if a disc contains two points of $C$, these are joined by a Delaunay-edge. So if we four-color the planar graph given by the Delaunay-Triangulation, and select one color-class, we have at most one point in each circle. So the unit discs around these centers generate a packing.

These theorems suggests two families of related questions:

- What is the smallest $a_k$ such that we can select from any $k$-fold covering a subfamily of at most an $a_k$ fraction of the discs that is still a covering?
- What is the largest $b_k$ such that we can select from any $k$-fold packing a subfamily of at least a $b_k$ fraction of the discs that is a packing?

Clearly $a_k \geq \frac{1}{k}$ and $b_k \leq \frac{1}{k}$, since the $k$-fold covering or packing could be a union of $k$ normal coverings or packings. We showed $a_2 \leq \frac{3}{4}$ and $b_2 \geq \frac{1}{4}$. It is known that the thinnest double covering by unit discs is thinner than a union of two coverings, and the

*Author's address: Department of Computer Science, City College of New York, 160 Convent Ave, New York, NY 10031, USA, e-mail peter@cs.ccny.cuny.edu. Supported by NSF award 1017539 same for double packings [6, 9]; these examples show that $a_2 > \frac{1}{2}$ and $b_2 < \frac{1}{2}$; but I have no construction giving an interesting bound. Indeed, double coverings are somewhat mysterious, and there is no conjectured structure for the thinnest double covering of the plane by unit discs (see also [5] Section 2.1 Problem 1).

The method does not extend in any simple way to higher multiplicities of coverings: there are point sets such that each unit disc contains $k$ points, but for some discs, the graph induced by the Delaunay-Triangulation is a star $K_{1,k-1}$. In that case, it is possible that in a four-coloring of this planar graph all but one points are in the same color class.

We can replace the Delaunay-Triangulation by any other triangulation, but we need the property that any unit disc which contains $k$ points has a nontrivial subgraph induced on these points. The minimum-weight triangulation is not useful here: there are examples in which the $k$ points in a unit disc induce an independent set in the triangulation. For the Delaunay-Triangulation, the induced graph on the points in a disc is always connected. If there were a triangulation for which the induced graph always contained a triangle, this would answer also the partition problem: choosing two color classes of a four-coloring would then guarantee at least one point in each disc.

## References

[1] G. Aloupis, J. Cardinal, S. Collette, S. Langerman, D. Orden, P. Ramos: Decomposition of Multiple Coverings into More Parts, *Discrete & Computational Geometry* **44** (2010) 706–723.

[2] X. Bai, Z. Yun, D. Xuan, B. Chen, W. Zhao: Optimal Multiple-Coverage of Sensor Networks, in: *IEEE International Conference on Computer Communications (INFOCOM)* (2011) 2498–2506.

[3] J. Beaudaux, A. Gallais, T. Razafindralambo: Multiple Coverage with Controlled Connectivity in Wireless Sensor Networks, *PE-WASUN* 2010, 9–16

[4] P. Brass: Optimal Relations between Coverage, Connectivity, Radius Ratio, and Region Size, manuscript 2012.

[5] P. Brass, W. Moser, J. Pach: Research Problems in Discrete Geometry, Springer, 2005.

[6] L. Danzer: Drei Beispiele zu Lagerungsproblemen, *Arch. Math.* **11** (1960) 159–165.

[7] P. Erdős, C.A. Rogers: Covering Space with Convex Bodies, *Acta Arithmetica* **7** (1962) 281–285.

[8] M.Hefeeda, M.Bagheri: Randomized k-Coverage Algorithms For Dense Sensor Networks, in: *IEEE International Conference on Computer Communications (INFOCOM)* (2007), 2376–2380.

[9] A. Heppes: Über mehrfache Kreislagerungen, *Elemente Math.* **10** (1955) 125–127.

[10] W. Jianzhen, G. Zhiyan: Research on Multiple Coverage and Connectivity for Wireless Sensor Network, in: *ICCSE 2011*, 1058–1062.

[11] S. Kumar, T.H.Lai, J.Balogh: On $k$-coverage in a mostly sleeping sensor network, *Wireless Networks*, **14**(2008), 277–294.

[12] P. Mani-Levitska, J. Pach: Decomposition Problems for Multiple Coverings with Unit Balls, manuscript 1986.

[13] J. Pach: Covering the Plane with Convex Polygons, *Discrete & Computational Geometry* **1** (1986) 73–81.

[14] J. Pach, G. Tóth: Decomposition of Multiple Coverings in Many Parts, in: *SoCG 07* (ACM Symposium on Computational Geometry 2007) 133–137.

[15] D. Pálvölgyi, G. Tóth: Convex Polygons are Cover-Decomposable, *Discrete & Computational Geometry* **43** (2010) 483–496.

[16] G. Tardos, G. Tóth: Multiple Coverings of the Plane with Triangles, *Discrete & Computational Geometry* **38** (2007) 443–450.

[17] B. Wang: Coverage Problems in Sensor Networks: A Survey, *ACM Computing Surveys* **43** (2011) Article 32, 53 pages.

[18] S.Yang, F.Dai, M.Cardei, J.Wu: On Connected Multiple Point Coverage in Wireless Sensor Networks, *International Journal of Wireless Information Networks*, **13(4)**(2006), 289–301.

# Smart-grid Electricity Allocation via Strip Packing with Slicing *

**(Extended Abstract)**

Soroush Alamdari† Therese Biedl† Timothy M. Chan† Elyot Grant‡ Krishnam Raju Jampani§
S. Keshav† Anna Lubiw† Vinayak Pathak†

## Abstract

One advantage of smart grids is that they can reduce the peak load by distributing electricity-demands over multiple short intervals. Finding a schedule that minimizes the peak load corresponds to a variant of a strip packing problem. Normally, for *strip packing problems*, a given set of axis-aligned rectangles must be packed into a fixed-width strip, and the goal is to minimize the height of the strip. The electricity-allocation application can be modelled as *strip packing with slicing:* each rectangle may be cut vertically into multiple slices and the slices may be packed into the strip as individual pieces. The *stacking constraint* forbids solutions in which a vertical line intersects two slices of the same rectangle.

We give a fully polynomial time approximation scheme for this problem, as well as a practical polynomial algorithm that slices each rectangle at most once and yields a solution of height at most 5/3 times the optimal height.

## 1 Introduction

The conventional approach to generating and distributing electricity relies on sizing infrastructure to support the peak load, when demand for electricity is highest. However, this peak is rarely reached, so much of the expensive infrastructure is idle most of the time. For example, in 2009, 15% of the generation capacity in Massachusetts was used less than 88 hours per year [5]. Reducing the infrastructure size is not practical since unsupported demand can cause blackouts. Therefore, there is considerable benefit to reducing the peak load itself.

Peak load occurs when many consumers use power-hungry appliances simultaneously. However, there is often flexibility in scheduling the use of particular appliances. For example, a water heater requires a certain amount of electricity to heat the water, but can equally well heat the water in one continuous interval or in multiple short intervals.[1] It is anticipated that future smart grids would obtain (at each substation) daily "demand schedules" for appliance use from the consumers in its local area, and then automatically re-schedule appliance use to minimize peak load [14].

The demand schedule can be modelled as a set of rectangles, one for each appliance, with power consumption as height, and desired running time as width. The re-scheduling should cover a given length of time, which corresponds to a strip of given width. The objective is then to pack slices of the rectangles into the strip so as to minimize the maximum power consumption, i.e., the maximum height of the packing. Because appliances cannot be powered at double the usual power, we have the additional *stacking constraint* requiring that no vertical line may intersect two slices from the same rectangle. Slicing with the stacking constraint is new, but strip packing has been well-studied, as we review in the following section.

**Strip packing problems.** In the *two-dimensional strip packing problem* (abbreviated 2SP), a set of axis-aligned rectangles of specified dimensions must be packed, without rotation, into a rectangular strip of fixed width, with the goal of minimizing the height of the strip. The 2SP problem is very well-studied [11], and generalizes the *bin packing problem*, which is equivalent to the case in which all rectangles have unit height. The current best approximation algorithm for 2SP has an approximation factor of $5/3+\varepsilon$ for any $\varepsilon > 0$ [7], and was achieved after a long sequence of successive improvements [1, 3, 12, 13, 15]. Many other authors have proposed algorithms with *additive* approximation guarantees [10, 8].

Motivated by the electricity-allocation problem, we study a variant called *two-dimensional strip packing with slicing* (hereafter 2SP-S). In 2SP-S, we are allowed to cut each rectangle vertically into multiple slices, which may be packed into the strip as individual rectangles. Formally, the input consists of a number $W$ and a set of rectangles $r_1, r_2, \ldots, r_n$. Here

*This work was done as part of an Algorithms Problem Session at the University of Waterloo. Research of TB, TC, SK and AL supported by NSERC.

†Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada {s26hosse,biedl,tmchan,alubiw,keshav,vpathak}@uwaterloo.ca

‡Massachusetts Institute of Technology, Cambridge, USA, elyot@mit.edu

§University of Guelph, Guelph, Canada, rjampani@uoguelph.ca

[1]To simplify the modelling we presume that no extra electricity is needed for re-starting the appliance.

$W$ is the width of the strip, which consists of two vertical sides at $x = 0$ and $x = W$, and the "base" at $y = 0$. Rectangle $r_i$ has width $w_i$ and height $h_i$; let $h_{\max} = \max_{i=1}^{n} h_i$ be the maximum height. A solution to 2SP-S consists of a partition of each rectangle $r_i$ into vertical slices and an assignment of positions to the slices so that the interiors of the slices are pairwise disjoint. Slices must not be rotated. The *height* of a solution, denoted by $H$, is the minimum $y$-coordinate above which the strip is empty.

Existing results: Strip packing with slicing has been studied for a variant in which the width of each rectangle represents a demand for a number of concurrently running processors [2]. However, this problem differs substantially from 2SP-S because the slices must have integer widths and must be horizontally aligned due to concurrency, and results for it do not carry over. 2SP-S also relates to the minimum makespan scheduling problem on $m$ parallel machines, if we slice each rectangle into strips of width $W/m$. Unfortunately we must make $m$ big to approximate well if rectangles have uneven widths, which makes the many existing algorithms for this scheduling problem too slow.

A second reason that existing results do not apply to electricity-allocation (at least not as far as we can prove) is that in our application we have the additional stacking constraint requiring that no vertical line may intersect two slices from the same rectangle. The version of 2SP-S with the stacking constraint is denoted by 2SP-SSC.

**Results for 2SP-S and 2SP-SSC.** The freedom to slice rectangles can be highly beneficial. It is easy to construct an example where slicing reduces the required height by a factor of $2 - \varepsilon$. Slicing also makes a difference in the complexity of the problem. Standard 2SP generalizes bin packing and is thus strongly NP-complete. Also, a simple reduction from the *Partition problem* [4] shows that 2SP admits no $(3/2-\varepsilon)$-approximation for any $\varepsilon > 0$ unless P=NP. In contrast, 2SP-S and 2SP-SSC are NP-hard (we omit details of the reduction from Partition), but not hard to approximate: we will give a fully polynomial-time approximation scheme (FPTAS) for these problems.

The FPTAS is based on solving a linear program with exponentially many constraints, and hence is mostly of theoretical interest. We also develop simpler, more practical algorithms, and also limit the number of times a rectangle may be sliced (which is of interest in the electricity-allocation problem to avoid start-up costs for the appliance.) We give two simple 2-approximation algorithms based on the well-known First Fit and Shelf paradigms. Then, building on these algorithms, and splitting the problem into two halves, we give two practical polynomial-time algorithms, one with approximation factor 3/2, and one that uses at most one cut per rectangle and has approximation factor 5/3.

Our paper is organized as follows. The First Fit and Shelf algorithms are in Section 2. Section 3 contains the FPTAS, and Section 4 develops practical algorithms. We only have space to describe/sketch the algorithms, leaving analyses and proofs of correctness to the long version.

## 2 Basic Algorithms

This section describes the *First Fit* and *Shelf* heuristics for 2SP-S and 2SP-SSC. Both algorithms achieve an approximation factor of 2, which is noteworthy given that, for the standard strip packing problem, 2-approximation algorithms are difficult to obtain [12, 15]. Both algorithms in fact achieve a height of $H_{OPT} + h_{\max}$, and hence are asymptotically optimal if $h_{\max}$ is considered an additive constant.

**First Fit Algorithm** Given a list of rectangles $r_1, r_2, \ldots, r_n$, the *First Fit* algorithm processes them in order, repeatedly finds the lowest point in the current solution where a slice of $r_i$ can be placed, and places the widest possible slice of $r_i$ there, breaking ties arbitrarily. Repeat with the remainder of $r_i$, and continue until all rectangles have been processed. In the case of 2SP-SSC, the stacking constraint must be respected when placing slices. See Figure 1.

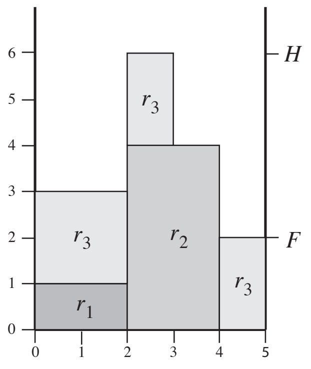


Figure 1: An execution of the First Fit algorithm on a 2SP-SSC instance. Note that $r_3$ is being sliced twice, and a smaller height would be achieved without the stacking constraint.

It is not hard to show that after placing each rectangle, the difference between the maximum height $H$ and the floor $F$ (the maximum height to which the entire strip is filled) is at most $h_{\max}$. Since by area-consideration $H_{\text{OPT}} \geq F$, First Fit achieves height at most $H_{\text{OPT}} + h_{\max}$.

**Shelf Algorithm** Given a set of rectangles, the Shelf algorithm for strip packing with slicing works as follows. Sort the rectangles by decreasing height so that $h_1 \geq h_2 \geq \ldots \geq h_n$. Pack the rectangles in this order on "shelves". The first shelf is the base of the strip.

Place rectangles on the current shelf from left to right. When we reach a rectangle $r_i$ that is too wide for the remaining space, we pack the widest possible slice of $r_i$. The rest of $r_i$ goes back in the list of remaining rectangles. Then we place a horizontal line across the strip to form a new shelf at the current maximum height of the packing, and continue on the new shelf with the remaining rectangles. See Figure 2. Note that the stacking constraint is automatically satisfied, and each rectangle is sliced at most once.

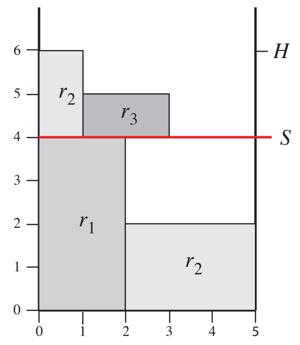


Figure 2: An execution of the Shelf algorithm on the same instance as Figure 1 (but rectangles have been sorted by height.)

Observe that (with $h_{n+1} := 0$) the empty space below height $H$ has area are most $\sum_{i=1}^{n}(h_i - h_{i+1}) \cdot W$. To see this, partition the empty space into rectangles by cutting it horizontally, and assign each empty rectangle to the $r_i$ that has a slice below it in the same shelf. Therefore, the empty space is at most $h_1 \cdot W = h_{\max} \cdot W$, which proves that the Shelf-algorithm achieves height at most $H_{\text{OPT}} + h_{\max}$.

## 3 Approximation Schemes

In this section, we sketch the FPTAS for 2SP-S and 2SP-SSC. The approach uses a linear programming relaxation and is relatively standard in the literature; in particular it resembles the classic work of Karmarkar and Karp concerning the bin-packing problem [9]. The main steps are as follows:

1. Guess the height $H_{\text{OPT}}$ of an optimal solution (using, for example, binary search).
2. Round the height of each rectangle down to the nearest multiple of some small value (dependent on $\varepsilon$, $n$, and our guess for $H_{\text{OPT}}$).
3. Formulate the rounded problem as a linear program, with one variable for each possible configuration of rectangles in a vertical slice of the packing. There are exponentially many variables, but only polynomially many constraints.
4. The dual program of this has polynomially many variables and exponentially many constraints. Solve it using the ellipsoid method, without explicitly listing the constraints, but instead solving in each iteration a Knapsack-problem (in polynomial time since we rounded the heights) to confirm feasibility or find a violated constraint.
5. From the solution of the dual problem, obtain a solution of the primal, reconstruct a packing of the original problem, and argue that its height is at most $(1+\varepsilon)H_{\text{OPT}}$.

The linear program we solve is similar to the one used to obtain fractional strip packings in [10], though our full algorithm requires different searching and rounding routines since the variables in our linear program must correspond to vertical configurations rather than horizontal ones.

## 4 2SP-SSC with Limited Cuts

Although the approximation scheme from the previous section may be more practical if the simplex method is used, it is still unsuitable for electricity-allocation applications both due to its runtime and because it may result in rectangles that have been sliced numerous times. We hence developed other algorithms that are simple to implement, run quickly, have decent approximation guarantees, and do not slice rectangles too often.

The approach is to partition the bin vertically, slice each rectangle once, and pack the two slices in the two parts with either First Fit or Shelf. With a judicious choice of where to partition and slice, this gives a 3/2-approximation if we use First Fit and a 5/3-approximation that slices each rectangle at most once if we use Shelf. (We note that both results are achieved without any partitioning if $h_{\max} \leq \frac{1}{2}H_{\text{OPT}}$ (resp. $h_{\max} \leq \frac{2}{3}H_{\text{OPT}}$).)

These algorithms work as follows:

1. Guess the height $H_{\text{OPT}}$ of an optimal solution (using, for example, binary search).
2. Sort the rectangles by decreasing height.
3. Fix a value $t \geq H_{\text{OPT}}/2$.
4. Find all rectangles $r_1, \ldots, r_j$ with height $> t$.
5. Let $\alpha W$ be the total width of $r_1, \ldots, r_j$; by $t \geq H_{\text{OPT}}/2$ we know $\alpha \leq 1$.
6. Divide the strip into two parts: the *left side* has width $\alpha W$ and the *right side* has width $(1-\alpha)W$.
7. Split each rectangle into a *left piece* and a *right piece* as follows.
   (a) Each of $r_1, \ldots, r_j$ obtains only a left piece (the right piece is empty.)
   (b) For $i = n, n-1, \ldots,$ the right piece of rectangle $r_i$ has width $\min\{w_i, (1-\alpha)W\}$, i.e., we add as much as possible of $r_i$ to the right piece. This continues until $i = j$ or until we

reach a rectangle $r_x$ such that the total area of the right pieces exceeds $(1-\alpha)WH_{\mathrm{OPT}}$.

(c) If $r_x$ exists, slice it so that the total area of the right pieces is exactly $(1-\alpha)WH_{\mathrm{OPT}}$.

8. Apply one of the basic algorithms in Section 2 to pack the left (resp. right) pieces in the left (resp. right) side of the strip.

**Theorem 1** *There exists a $3/2$-approximation for 2SP-SSC that runs in $O(n^2)$ time.*

*There exists a $5/3$-approximation for 2SP-SSC that slices every rectangle at most three times and runs in time $O(n\log(nM))$, where $M$ is an upper bound on the integer heights of the rectangles.*

*There exists a $5/3$-approximation for 2SP-SSC that slices every rectangle at most once and runs in time $O(n\log^2 n\log(nM)/\log\log n)$.*

**Proof.** [Sketch] Since $r_1,\dots,r_j$ have total width $\alpha W$ and are packed in the left strip of width $\alpha W$, the left strip has no empty space below height $t$. In the right strip, all rectangles have height at most $t$. Using this, and the bound on the area in the right strip, we can show that the right strip has height at most $H_{\mathrm{OPT}}+t$. If we use First Fit in Step 8, then the left strip has height at most $H_{\mathrm{OPT}}+t$ and using $t=H_{\mathrm{OPT}}/2$ gives the first result. If we use Shelf, then the left strip can be shown to have height at most $3H_{\mathrm{OPT}}-2t$. Notice that using Shelf on both sides gives a packing with at most 3 cuts (one to cut into the left and right piece, and one by each application of Shelf), and the only rectangle that may have 3 cuts is $r_x$. So using $t=2H_{\mathrm{OPT}}/3$ gives the second result, and the third result is obtained from the second by doing some extra work to align the pieces on the shelves. □

## 5 Conclusions

Motivated by an application in electricity allocation, this paper explored variants of the strip packing problem in which rectangles could be sliced vertically as long as no two slices of the same rectangle are stacked atop each other. We provided simple 2-approximation algorithms, an FPTAS of mostly theoretical interest, and practical approximation algorithms that slice rectangles only a few times.

The main remaining open problem is to find practical algorithms with better approximation factors. For example we conjecture that First Fit Decreasing, i.e., First Fit applied to rectangles in decreasing height, is a 4/3-approximation. Without the stacking constraint, this follows from Graham's 4/3-approximation bounds for multiprocessor scheduling [6], but with the stacking constraint the best bound we can prove is 3/2 (details of this will be in the full paper.) Also, is there a simple PTAS for strip-packing with slicing (with or without stacking constraint)?

## References

[1] B. S. Baker, E. G. Coffman, and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[2] M. Bougeret, P.-F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximating the non-contiguous multiple organization packing problem. In *IFIP TCS*, pages 316–327, 2010.

[3] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM J. Comput.*, 9(4):808–826, 1980.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman & Co Ltd, 1979.

[5] P. Giudice. Our energy future and smart grid communications. Testimony before the FCC Field Hearing on Energy and Environment. `www.broadband.gov/fieldevents/fh_energy_environment/giudice.pdf`, 2009.

[6] R. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.

[7] R. Harren, K. Jansen, L. Prädel, and R. van Stee. A $(5/3+\varepsilon)$-approximation for strip packing. In *Algorithms and Data Structures Symposium, WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 475–487. Springer, August 2011.

[8] K. Jansen and R. Solis-Oba. New approximability results for 2-dimensional packing problems. In *Mathematical Foundations of Computer Science 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2007.

[9] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Symposium on Foundations of Computer Science*, pages 312–320. IEEE, 1982.

[10] C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Math. Oper. Res.*, 25(4):645–656, 2000.

[11] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241 – 252, 2002.

[12] I. Schiermeyer. Reverse-Fit: A 2-optimal algorithm for packing rectangles. In *European Symp. Algorithms*, volume 855 of *Lecture Notes in Computer Science*, pages 290–299. Springer, 1994.

[13] D. D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, Feb. 1980.

[14] P. Srikantha, C. Rosenberg, and S. Keshav. An analysis of peak demand reductions due to elasticity of domestic appliances. In *Proc. Energy-Efficient Computing and Networking (e-Energy '12)*, number 28. ACM, 2012.

[15] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

# Packing Identical Simply Polygons of Constant Size is NP-hard[*]

Ning Xu[†]

## 1 Introduction

The decision version of the *identical simple polygon packing problem* can be described as: given a small simple polygon $S$, a large simple polygon $P$, and a positive integer $k$, can one place $k$ copies of $S$ inside $P$ with translation and rotation and without overlap?

If $S$ is nonconvex and polynomial in size, this decision problem is NP-hard even if both $S$ and $P$ are orthogonal [1].

This decision problem is related to the problem 56 in the Open Problems Project [2], in which $S$ is restricted to an axis-parallel unit square (rotation is forbidded). In a variation of the problem 56, $P$ may contain holes. The decision problem for this variation is NP-complete [3, 4] even if $P$ is an orthogonal polygon with all coordinates being multiples of 1/2.

In this paper, we improve the result in [2] by restricting $S$ in constant size.

**Theorem 1** *The decision version of the identical simple polygon packing problem is NP-hard, even if only translation is permitted, $S$ has constant size, and both $S$ and $P$ are orthogonal.*

## 2 Proof

We reduce an instance of a known NP-hard problem, the 0-1 knapsack problem [5], to an instance of our problem. The decision version of the 0-1 knapsack problem is: given $n$ items $I_1, \ldots, I_n$, in which the item $I_i$ has value $V_i \in \mathbb{Z}^+$ and the weight $W_i \in \mathbb{Z}^+$, and a total weight limit $W \in \mathbb{Z}^+$ and a desired value $V \in \mathbb{Z}^+$, can one carry some items with value at least $V$ and total weight at most $W$, that is, does there exist a sequence $\{b_1, \ldots, b_n\}$ with $b_i \in \{0, 1\}$ such that $\sum_{i=1}^{n} w_i b_i \le W$ and $\sum_{i=1}^{n} v_i b_i \ge V$? Here we assume that $W < \sum_{i=1}^{n} w_i$; otherwise, one can carry all items.

### 2.1 The identical simple polygon $S$

Figure 1 shows the small identical simple polygon $S$. This identical polygon is formed by two rectangles of width 1 and height $H$, a rectangle of width $X$ and height 1, and a rectangle of width $L$ and height 1. Here $H$, $X$ and $L$ are positive integers whose value will be discussed later.

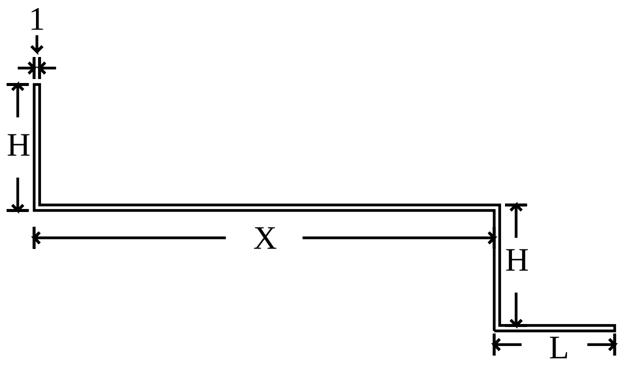


Figure 1: Identical simple polygon $S$

### 2.2 The gadget

A *gadget* $G_i (1 \le i \le n)$, showed in Figure 2, is a simple polygon corresponding to the item $I_i$ in the 0-1 knapsack problem. The gadget can be decomposed into two components: the red one is called *the handle*, and has width $v_i$; the blue one is called *the body*. The left bottom corner of the body, the point $r$, is called the *root point*.

If $X$ is sufficient large, one can place at most $v_i$ identical polygons in the gadget $G_i$. Here $c$ and $d_i$ are positive integers, whose value will be discussed later.

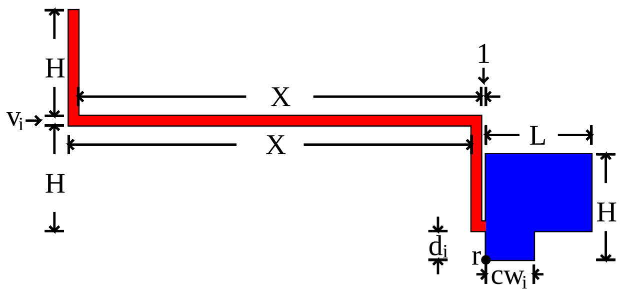


Figure 2: Gadget $G_i$.

### 2.3 Reduction

As illustrated in Figure 3, we construct a large simple polygon $P$ by mounting $n$ gadgets $G_1, \ldots, G_n$ on the

[*]The research is funded by the NSF grant CCF-1017539

[†]The Graduate Center, The City University of New York, New York, USA, nxu@gc.cuny.edu

top of another simple polygon called the *base*. The base consists of a rectangle of width $X$ and height $T$, a rectangle of width $X$ and height $H+T$ and a rectangle of width $L$ and height $T$. Here $T$ is a positive integers whose value will be discussed later. Without loss of generality, we set the point $o$ to be the origin on the plane and set the top side of the base to be the $x$-axis. The gadget $G_i$ is mounted on the base such that the root point of $G_i$ locates at the point $((i-1)(L+c+1),0)$.

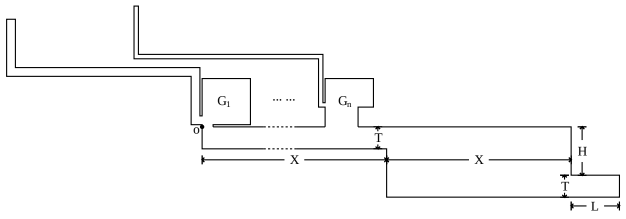


Figure 3: Large simple polygon $P$, constructed by reduction

Now, we discuss the values used in the reduction.

The horizontal distance between two neighbor gadgets is $L+c+1$. We set

$$c = \max\{v_1,\ldots,v_n\}+1 \tag{1}$$
$$H > T \tag{2}$$
$$X = (L+c+1)n \tag{3}$$

such that there are only two ways to place an identical polygon $S$, as illustrated in Figure 4.

- Place $S$ in $G_i$. We say $S$ is *private* in $G_i$.
- Place a part of $S$ in $G_i$ and place the other part of $S$ in the base. We say $S$ is *shared* in $G_i$.

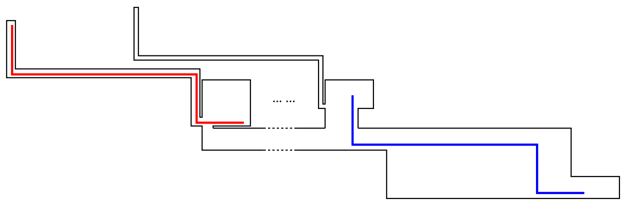


Figure 4: The two ways to place an identical polygon. The red one on left is a private identical polygon, and the blue one on right is a shared identical polygon.

We set

$$d_i = \begin{cases} 1 & i=1 \\ d_{i-1}+v_{i-1}+1 & i>1 \end{cases} \tag{4}$$

to let $P$ be a simple polygon. From the reduction, all gadgets are mounted on the base, and the body of any two gadgets are separated. For a gadget $G_i$, because $d_i>0$ and $c=\max\{v_1,\ldots,v_n\}+1>v_i$, the handle of $G_i$ is separated from the body of any other gadget. When $i>1$, because $d_i>d_{i-1}+v_{i-1}$, the handle of $G_i$ is separated from $G_{i-1}$ by translating to upper-right. This implies that the handle of any two gadgets are separated. From the reduction, $P$ is a simple polygon.

Since the gadget $G_i$ corresponds to the item $I_i$, we assume that a private identical polygon and a shared identical polygon cannot both exist in the same gadget in any packing. This corresponds that one can either choose or not choose an item. To valid this assumption, we set $H$ and $L$ as:

$$H = T+d_n+v_n+1 \tag{5}$$
$$L = c(\max\{w_1,\ldots,w_n\}+1)+2 \tag{6}$$

Note that (5) satisfies (2). In any gadget $G_i$, if one places a shared identical polygon, because $H-T>d_n+v_n\geq d_i+v_i$, no private identical polygon can be placed in $G_i$. On the other hand, if one places a private identical polygon, because $L-v_i-1>L-c-1>cw_i$, no shared identical polygon can be placed in $G_i$.

Finally, we set

$$T = c(\sum_{i=1}^{n} w_i - W) \tag{7}$$

All values used in the reduction can be calculated in $O(n)$ time. Each gadget has $O(1)$ vertices. Because there are $n$ gadgets. The reduction can be done in $O(n)$ time.

For any gadget $G_i$, there are at most $cw_i$ shared identical polygons in $G_i$. For any packing, if there is no shared identical polygon in $G_i$, $G_i$ is *unoccupied*; if there are $cw_i$ shared identical polygons in $G_i$, $G_i$ is *occupied*; otherwise, $G_i$ is *partially occupied*. One can place a private identical polygon in $G_i$ if and only if $G_i$ is unoccupied. Observing the base, the total number of shared identical polygons is at most $T$.

The *packing number* of a given packing is the number of identical polygons placed inside $P$. We will show that there exists a packing with packing number at least $V+T$ if and only if there is a sequence $b_1,\ldots,b_n$ with $b_i\in\{0,1\}$ such that $\sum_{i=1}^{n} w_k b_k \leq W$ and $\sum_{i=1}^{n} v_k b_k \geq V$. Note that $T$ is a function of $v_1,\ldots,v_n,w_1,\ldots,w_n$ and $W$.

We first prove the "if" part. Suppose such a sequence $b_1,\ldots,b_n$ exists. We place identical polygons for gadgets in the order $G_1,\ldots,G_n$. If $b_i=1$, we place $v_i$ private identical polygons in $G_i$. If $b_i=0$, we try to place $cw_i$ shared identical polygons in $G_i$ compactly from left bottom. If the space in the base is insufficient for $cw_i$ shared identical polygons, we place as many shared identical polygons as possible.

Because $\sum_{i=1}^{n} v_i b_i \geq V$, the number of private identical polygons is at least $V$. Because $\sum_{i=1}^{n} w_i b_i \leq W$,

$$\begin{aligned}&\sum_{i=1}^{n} cw_i(1-b_i)\\=&c(\sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i b_i)\\\geq&c(\sum_{i=1}^{n} w_i - W)\\=&T\end{aligned} \tag{8}$$

Thus, the number of shared identical polygons in this packing is exact $T$. Therefore, the packing number is at least $V+T$.

Then, we prove the "only if" part. Suppose that there exists a packing with packing number at least $V+T$.

**Lemma 2** *If there are at least two partially occupied gadgets in a packing, then there exists another packing with same packing number, same number of shared identical polygons, and at most one partially occupied gadget.*

**Proof.** Suppose $G_i$ and $G_j$ are two partially occupied gadgets. One can obtain another packing with the same packing number by moving one shared identical polygon from $G_i$ to $G_j$. Continue this operation until one of $G_i$ and $G_j$ is either unoccupied or occupied. This decreases the number of partially occupied gadgets by at least 1. Note that the total number of shared identical polygons does not change.

Recursively eliminating partially occupied gadgets, one can finally obtain a packing with at most one partially occupied gadget. □

**Lemma 3** *If there are less than $T$ shared identical polygons in a packing, there exists another packing with larger or equal packing number having exact $T$ shared identical polygons.*

**Proof.** Suppose the packing contains $m < T$ shared identical polygons.

By Lemma 2, we first obtain a packing with same packing number, $m$ shared identical polygons, and at most one partially occupied gadget.

If the packing number is at most $T$, one can obtain another packing by only placing $T$ shared identical polygons, then we are done.

If $m = 0$, all gadgets are unoccupied. Because $T = c(\sum_{i=1}^{n} w_i - W) > c > v_1$, we can obtain a new packing with larger packing number by removing all private identical polygons from $G_1$ and add $c$ shared identical polygons in $G_1$. Thus, we can assume $m > 0$.

If there is one partially occupied gadget $G_i$, which contains $m_i$ shared identical polygons, one can add more shared identical polygons. If $cw_i - m_i \geq T - m$, one can add $T-m$ shared identical polygons in the gadget $G_i$. The new packing has larger packing number and $T$ shared identical polygons, and we are done. If $cw_i - m_i < T - m$, one can add $cw_i - m_i$ shared identical polygons in the gadget $G_i$, letting $G_i$ become occupied.

If there is no partially occupied gadget, because every gadget is either unoccupied or occupied now, the total number of shared identical polygons can be divided by $c$. Because $T$ can also be divided by c, there is still some space in the base for at least $c$ shared identical polygons. One can choose an unoccupied gadget, say $G_j$, remove all $v_j$ private identical polygons, and add $c$ shared identical polygons. Because $v_j < c$, the new packing has larger packing number, and $G_j$ is a partially occupied or occupied gadget now.

We can recursively add shared identical polygons, until we obtain a packing with exact $T$ shared identical polygons. □

By Lemma 2 and Lemma 3, we can assume that there is at most one partially occupied gadget and exact $T$ shared identical polygons in the packing. We obtain a sequence $b_1, \dots, b_n$ by setting $b_i = 1$ if the gadget $G_i$ is unoccupied, or $b_i = 0$ otherwise.

Suppose $G_j$ is the partially occupied gadget which has $m_j \leq cw_j$ shared identical polygons (If there is no partially occupied gadget, we select an arbitrary occupied gadget). Because the packing has exact $T$ shared identical polygons, we have:

$$\begin{aligned}&\sum_{i=1}^{n} cw_i(1-b_i) - cw_j + m_j = T\\\Rightarrow&(\sum_{i=1}^{n} cw_i(1-b_i) \geq T\\\Rightarrow&c(\sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i b_i) \geq c(\sum_{i=1}^{n} w_i - W)\\\Rightarrow&\sum_{1=1}^{n} w_i b_i \leq W\end{aligned} \tag{9}$$

The total number of private identical polygons is $\sum_{i=1}^{n} v_i b_i$. Because the packing number is at least $V+T$, the number of private identical polygons is at least $V$, i.e., $\sum_{i=1}^{n} v_i b_i \geq V$ holds.

This completes the proof of the Theorem 1.

## 3 Conclusion

We proved that it is NP-hard to determine whether a given number of simple identical polygons of constant size can be packed into a simple polygon.

However, there is still a gap between our problem and the problem 56 in the Open Problem Project. If we further restrict $S$ to be convex, the complexity of our problem is still unknown.

**Acknowledgment**
The author thanks Peter Brass for his valuable comments and helpful suggestions related to the paper.

## References

[1] S.R. Allen and J. Iacono. Packing identical simple polygons is np-hard. In *Fall Workshop on Computational Geometry*, pages 19–20, 2012.

[2] Mitchell J.S.B Demaine, E.D. and O'Rourke J. The open problems project. *http://cs.smith.edu/orourke/TOPP/Welcome.html*.

[3] El-Khechen D. Iacono J. Dulieu, M. and N. van Omme. Packing $2 \times 2$ unit squares into grid polygons is np-complete. In *the Canadian Conference on Computational Geometry*, pages 33–36, 2009.

[4] Paterson M.S. Fowler, R.J. and Tanimoto S.L. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3), 1981.

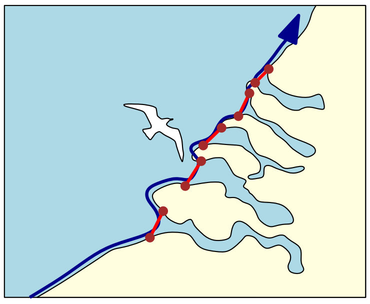
[5] M.R. Garey and Johnson D.S. *Computers and Intractability: A Guide to the Theory of NPCompleteness*. W.H. Freeman, 1979.

# Computing the Fréchet distance with shortcuts is NP-hard

Maike Buchin[*] Anne Driemel[†] Bettina Speckmann[*]

## Abstract

We study the *shortcut Fréchet distance*, a natural variant of the Fréchet distance that allows us to take shortcuts from and to any point along one of the curves. We show that, surprisingly, the problem of computing the shortcut Fréchet distance exactly is NP-hard. Furthermore, we give a 3-approximation algorithm for the decision version of the problem.



## 1 Introduction

Measuring the similarity of two curves is an important problem which occurs in many applications. A popular distance measure, that takes into account the continuity of the curves, is the Fréchet distance. Imagine walking forwards along the two curves simultaneously. At any point in time, the two positions have to stay within distance $\varepsilon$. The minimal $\varepsilon$ for which such a traversal is possible is the Fréchet distance. In general, the Fréchet distance can be computed by the algorithm of Alt and Godau [1] in $O(n^2 \log n)$ time. Despite its versatility, the Fréchet distance has one serious drawback: it is a bottleneck distance. Hence it is quite sensitive to outliers, which are frequent in real world data sets. To remedy this Driemel and Har-Peled [3] introduced a variant of the Fréchet distance, namely the *shortcut Fréchet distance*, that allows shortcuts from and to any point along one of the curves. The shortcut Fréchet distance is then defined as the minimal Fréchet distance over all possible such shortcut curves.

The shortcut Fréchet distance automatically cuts across outliers and allows us to ignore data specific "detours" in one of the curves. Hence it produces significantly more meaningful results when dealing with real world data than the classic Fréchet distance. Consider the following example. Birds are known to use coastlines for navigation, e.g., the Atlantic flyway for migration. However, when the coastline takes a "detour", like a harbor or the mouth of a river, the bird ignores this detour, and instead follows a shortcut across. See the example of a seagull in the figure, navigating along the coastline of Zeeland while taking shortcuts between the islands. Using the shortcut Fréchet distance, we can detect if the trajectory of the bird is similar to the coastline. The shortcut Fréchet distance can be interpreted as a partial distance measure. Note that a different notion of a partial Fréchet distance was developed by Buchin et al. [2].

**Definitions.** A ***curve*** $T$ is a continuous mapping from $[0,1]$ to $\mathbb{R}^2$, where $T(t)$ denotes the point on the curve parameterized by $t \in [0,1]$. Given two curves $T$ and $B$ in $\mathbb{R}^2$, the ***Fréchet distance*** between them is

$$\mathsf{d}_{\mathcal{F}}(T,B) = \min_{f:[0,1]\to[0,1]} \max_{\alpha\in[0,1]} \|T(f(\alpha)) - B(\alpha)\|,$$

where $f$ is an orientation-preserving reparameterization of $T$. We call the line segment between two arbitrary points $B(y)$ and $B(y')$ on $B$ a ***shortcut*** on $B$. Replacing a number of subcurves of $B$ by the shortcuts connecting their endpoints results in a shortcut curve of $B$. Thus, a ***shortcut curve*** is an order-preserving concatenation of non-overlapping subcurves of $B$ that has straight line segments connecting the endpoints of the subcurves. Our input are two polygonal curves: the ***target curve*** $T$ and the ***base curve*** $B$. The ***shortcut Fréchet distance*** $\mathsf{d}_{\mathcal{S}}(T,B)$ is now defined as the minimal Fréchet distance between the target curve $T$ and any shortcut curve of the base curve $B$.

**Results.** In this paper we study the complexity of computing the shortcut Fréchet distance. Driemel and Har-Peled [3] described approximation algorithms for the shortcut Fréchet distance in the restricted case where shortcuts have to start and end at input vertices. Specifically, they gave a $(3+\varepsilon)$-approximation algorithm for the vertex-restricted shortcut Fréchet distance between $c$-packed polygonal curves that runs

[*]Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands, m.e.buchin@tue.nl and speckman@win.tue.nl. Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.106. and 639.022.707.

[†]Department of Information and Computing Sciences Utrecht University, the Netherlands, a.driemel@uu.nl. Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.065.823.

in $O(c^2 n \log^3 n)$ time for two $c$-packed polygonal curves of complexity $n$. Firstly, we outline how to combine the algorithmic layout of Driemel and Har-Peled [3] with a line stabbing algorithm of Guibas et al. [4] to obtain a 3-approximation algorithm for the decision version of the general shortcut Fréchet distance which runs in $O(n^3 \log n)$ time. This result is described in the full version of this paper.

Secondly, we show that, surprisingly, in the general case, where shortcuts can be taken at any point along a curve, the problem of computing the shortcut Fréchet distance exactly is NP-hard. An important observation is that the reachable free space of the matchings may fragment into an exponential number of components. We use this fact in our reduction together with a mechanism that controls the sequence of free space components that may be visited. In this abstract we describe the construction, we prove the correctness in the full version.

## 2 NP-Hardness

Assume an instance of SUBSET-SUM is given to us as $n$ positive integers $S = \{s_1, s_2, \ldots, s_n\}$ and a positive integer $\sigma$. Recall that the problem is to decide whether there exists an index set $I$, such that $\sum_{i \in I} s_i = \sigma$. We now describe a construction of a target curve $T$ and base curve $B$ such that there exists a shortcut curve of the base curve that is within Fréchet distance 1 to the target curve if and only if there exists a subset of $S$ of which the total sum is $\sigma$.

**Basic Idea.** The base curve has several horizontal edges within distance 1 of the target curve. A feasible shortcut curve has to visit a well-defined subsequence of these edges. Any possible visiting sequence will encode a different subset of $S$. Let $I$ be an index set which defines such a subset $S' \subseteq S$, we call $s_i = \sum_{1 \le j \le i, j \in I} s_j$ the $i$th ***partial sum*** of $S'$. The incremental partial sums encoded by a shortcut curve are encoded on certain edges of the base curve by the particular point where the shortcut visits the edge. We can restrict the solutions to the Fréchet problem to have this specific form by using what we call projection centers. These are certain points on the target curve, which have to be visited by any curve that is within Fréchet distance 1 to the target curve. Intuitively a shortcut of a feasible shortcut curve has to start and end at particular edges of the base curve and intersect a projection center in between. Hence we can think of the shortcut as a projection. The construction is such that a shortcut that enters a gadget will have two edges of the base curve available as possible destinations. The corresponding projections will then cascade through the projection centers of the gadget and are bundled again on the last edge of the gadget, where they have a certain distance to each other, which encodes one of the input values.

**General layout and notation.** We denote with $H_0$, $H_1$, $H_{-1}$ and $H_\alpha$ the horizontal lines at $0, 1, -1$ and $\alpha$. All relevant vertices of the construction lie on these lines (see Figure 1) and hence it is usually sufficient to specify their $x$-coordinates. We slightly abuse notation by denoting the $x$-coordinate of a point and the point itself with the same variable, albeit using a different font.

The target curve consists entirely of edges that lie on the $x$-axis. In Figure 1, the curve drawn below the $x$-axis illustrates the topology of the target curve. Clearly all feasible shortcut curves have to lie within the hippodrome of radius 1 around the target curve. The $i$th gadget defines a subcurve $T_i$. The vertices of $T_i$, except for the initialization and the terminal gadget, are defined by the parameters $c_j^{(i)}$ for $1 \le j \le 4$. These are $(c_1^{(i)}+1, 0), (c_1^{(i)}-1, 0), (c_2^{(i)}+1, 0), (c_2^{(i)}-1, 0), (c_3^{(i)}+1, 0), (c_3^{(i)}-1, 0), (c_4^{(i)}+1, 0), (c_4^{(i)}-1, 0)$ in this order. Thus, the edges of the target curve are generally running in positive $x$-direction, except for some edges of length two, which are centered at the points $(c_j^{(i)}, 0)$. We call these points ***projection centers***. The construction of the base curve is such that any feasible shortcut curve has to go through the projection centers. In particular, this is enforced by the fact that we place all edges of the base curve at distance at least 2 away from the projection centers.

The base curve has relevant edges $\mathsf{e}_j^{(i)}$, for $1 \le j \le 7$ and $0 \le i \le n$, where $j$ defines the order along the base curve. These edges lie on the horizontal lines $H_1$, $H_{-1}$ and $H_\alpha$ at $1, -1$ and $\alpha \in (0,1)$. We call these edges ***docking edges***, since they are the edges visited by the feasible shortcut curves. The docking edges run in negative $x$-direction. The remaining edges of the base curve are outside the hippodrome, except for ***connector edges***, which vertically connect to docking edges on $H_\alpha$ and run in positive $y$-direction.

**Global variables.** The construction uses four global variables $\alpha \in (0,1), \beta > 0$, and $\delta > 0$. The parameter $\alpha$ is besides 1 and $-1$ the $y$-coordinate of the horizontal lines that support the docking edges. The parameter $\beta$ controls the minimal horizontal distance between docking edges that lie in between two consecutive zones. The function of the parameter $\delta$ is two-fold. Firstly, it is the minimum difference of two partial sums. This can be ensured by scaling the instance by $\delta$, such that $s_i/\delta \ge 1$ for $1 \le i \le n$ and $\sigma/\delta \ge 1$. Secondly, we choose $\delta$ sufficiently large to ensure that a feasible shortcut curve cannot visit any edges other than docking edges and only in the prescribed visiting order.

**Encoding of a solution.** A shortcut curve $B_\Diamond$ of the base curve encodes a subset $S' \subseteq S$ as follows: The value $s_i$ is included in $S'$ if and only if $B_\Diamond$ visits $\mathsf{e}_1^{(i)}$. Any feasible shortcut curve $B_\Diamond$ also encodes an approximation of its incremental partial sums in the distance between the point where $B_\Diamond$ visits the edge $\mathsf{e}_7^{(i)}$ to the endpoint of this edge $\mathsf{a}_7^{(i)}$. By choosing the global parameter $\delta$ carefully, we can ensure that two distinct partial sums have a minimum difference that exceeds the approximation error. By construction of the terminal gadget any feasible shortcut curve has to visit $\mathsf{e}_7^{(n)}$ at a point that is in distance $\sigma + \delta$ to the point $\mathsf{a}_7^{(i)}$. This implies that only shortcut curves that encode a subset that sums to $\sigma$ can be feasible.

**Construction of the gadgets.** We now describe the part of the construction of the gadgets that is specific to the instance of the problem. That is, we give exact choices of the coordinates of the two curves. The construction is incremental. Given the endpoints of edge $\mathsf{e}_7^{(i-1)}$, as defined by the $(i-1)$th gadget and the value $s_i$, we describe how to construct the subcurves of the intermediate gadget $G_i$. We describe the initialization and the terminal gadget afterwards. Since all relevant vertices of the base and target curve lie on horizontal lines as indicated in Figure 1, we need to choose only their $x$-coordinates. The construction goes through several rounds of fixing the position of the next projection center and then projecting an endpoint $\mathsf{a}_j^{(i)}$ of one edge to obtain the endpoint $\mathsf{a}_{j+1}^{(i)}$, $\mathsf{a}_{j+2}^{(i)}$, or $\mathsf{a}_{j+3}^{(i)}$ of another edge. The endpoint $\mathsf{b}_j^{(i)}$ is projected in the same way. Thus, we obtain the first point of one edge by projecting the last point of another and the other way around.

The detailed construction is described in the full version of the paper. Here, we only describe how to pick the first and the last projection center. From $G_{i-1}$, we are given the values of $a_7^{(i-1)}$ and $b_7^{(i-1)}$. Let $\ell = b_7^{(i-1)} - a_7^{(i-1)}$ and let $h_i = b_7^{(i-1)} + \beta + \ell$. We choose $c_1^i$ as the $x$-coordinate where the line through $(h_i, -\alpha)$ and $(b_7^{(i-1)}, -1)$ passes through $H_0$. We obtain $a_j^{(i)}$ and $b_j^{(i)}$ for $1 \leq j \leq 6$ from the subsequent projections through the constructed projection centers as shown in the figure. Now, Let $p_i = a_6^{(i)} - s_i$. We choose $c_4^{(i)}$ as the $x$-coordinate where the line through $(p_i, 1)$ and $\mathsf{a}_5^{(i)}$ passes through $H_0$. And finally we project the points $\mathsf{a}_5^{(i)}, \mathsf{b}_5^{(i)}, \mathsf{a}_6^{(i)}$ and $\mathsf{b}_6^{(i)}$ through the last projection center $\mathsf{c}_4^i$ onto $H_{-1}$. We then choose $a_7^{(i)}$ as the minimum of the obtained $x$-coordinates and $b_7^{(i)}$ as the maximum of the obtained $x$-coordinates.

In this manner we obtain the docking edges $\mathsf{e}_j^{(i)}$ for $1 \leq j \leq 7$. We connect $\mathsf{e}_7^{(i-1)}$ to $\mathsf{e}_1^{(i)}$ using edges that lie outside the hippodrome. Similarly we connect the remaining edges in the order of $j$ using vertical connector edges for the edges lying on $H_\alpha$ and otherwise edges that lie outside the hippodrome.

**Initialization.** We place the first vertex of the target curve at $(a_0^{(0)}, 0) = (0, 0)$ and the first vertex of the base curve at $(a_0^{(0)}, 1) = (0, 1)$. The base curve then continues to the left on $H_1$ while the target curve continues to the right on $H_0$. $G_0$ has one projection center $(c_1^{(0)}, 0)$, we define it by $c_1^{(0)} = \delta + 2$. Then we define $\mathsf{e}_7^{(0)}$ such that $\mathsf{a}_0^{(0)}$ projects onto the center of this edge and such that the projection is in distance $\delta$ to both endpoints. That is, we define $a_7^{(0)} = c_1^{(0)} + 2$ and $b_7^{(0)} = c_1^{(0)} + 2\delta + 2$. Now, the next gadget $G_1$ can be constructed as described above.

**Terminal gadget.** We choose the very last projection center by setting $c_1^{(n+1)} = b_7^{(n)} + 2$. Let $p_\sigma = (a_7^{(n)} + \delta + \sigma)$ and project the point $(p_\sigma, -1)$ through this projection center onto $H_1$ to obtain a point $(a_\sigma, 1)$. We finish the construction by letting both the target curve and the base curve end on a vertical line at $a_\sigma$. The target curve ends on $H_0$ approaching from the left, while the base curve ends on $H_1$ approaching from the right.

**Proof Idea.** Consider the following construction of a shortcut curve that encodes a given subset $S' \subseteq S$. We start in $B(0)$, and subsequently project through all projection centers. In the intermediate gadget for $s_i$, we visit $\mathsf{e}_1^{(i)}$ if $s_i \in S'$, otherwise we visit $\mathsf{e}_2^{(i)}$. Finally, we choose $B(1)$ as the last vertex of our shortcut curve. We claim that this curve is feasible if and only if $S'$ is a solution. This can be proven by a repeated application of the intercept theorem. Note that this curve visits any edge of the base curve in at most one point. Clearly not all feasible shortcut curves have this property. However, they have to be approximately monotone by the construction of the target curve. This helps us to bound the error in the encoding of the partial sums.

**Acknowledgements.** We thank Maarten Löffler for insightful discussions on the topic of this paper.

## References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. of Comp. Geometry & Applications*, 5:75–91, 1995.

[2] K. Buchin, M. Buchin, and Y. Wang. Exact algorithm for partial curve matching via the Fréchet distance. In *Proc. 20th ACM-SIAM Symp. on Discrete Algorithms*, pages 645–654, 2009.

[3] A. Driemel and S. Har-Peled. Jaywalking your dog – computing the Fréchet distance with shortcuts. In *Proc. 23rd ACM-SIAM Symp. on Discrete Algorithms*, pages 318–337, 2011.

[4] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *Proc. 2nd Int. Symp. on Algorithms*, pages 151–162, 1991.

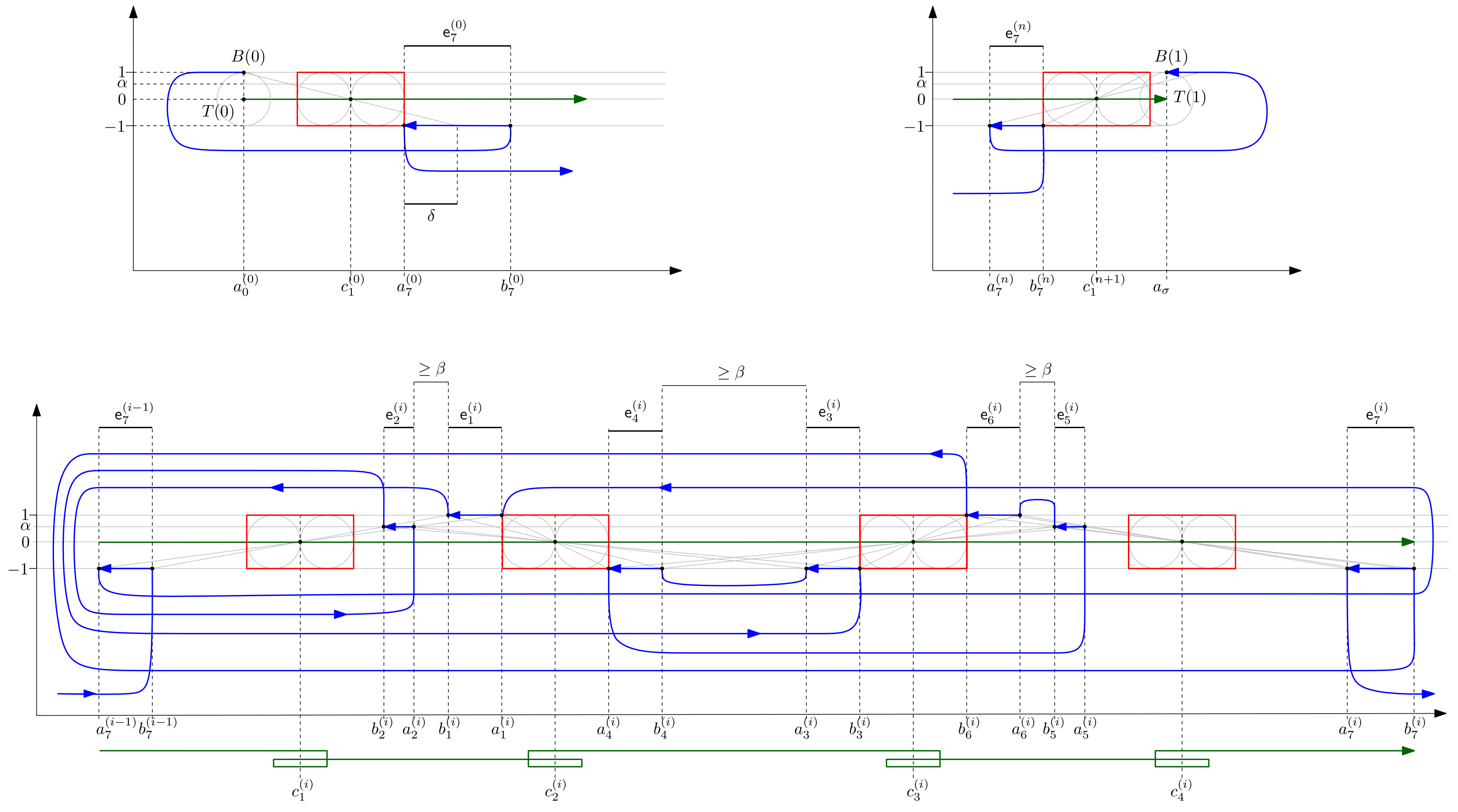


Figure 1: Initialization gadget, Terminal gadget and intermediate gadgets $G_i$ with global parameters $\alpha$ and $\beta$. The target curve is shown in green, the base curve in blue. For the sake of presentation the lengths of the docking edges have been assumed smaller.

# Parallel computation of the Hausdorff distance between shapes *

Helmut Alt† Ludmila Scharf†

## Abstract

We show that the Hausdorff distance for two two-dimansional shapes, modeled as sets of $n$ non-intersecting line segments, can be computed in parallel performing $O(n \log^2 n)$ total work, even in time $O(\log^2 n)$ on $n$ processors. We discuss how some parts of the sequential algorithm can be performed in parallel using previously known parallel algorithms; and identify the so-far least efficiently solved part of the problem, which is the following: Given two sets of $x$-monotone, non-intersecting curve segments, red and blue, for each red segment find its extremal intersection points with the blue set, i.e. points with the minimal and maximal $x$-coordinate. The algorithm presented here improves the best known theoretical time and space performance while still being practically feasible.

## 1 Introduction

Evaluating the similarity of two geometric shapes is an important problem in different fields of computer science including computer vision and pattern recognition. One of the most natural similarity measures is the *Hausdorff distance* which is defined for any two compact sets. The directed Hausdorff distance between two compact point sets $P$ and $Q$ is defined as $d_H(P,Q) = \max_{p\in P} \min_{q\in Q} d(p,q)$, where $d(p,q)$ denotes the Euclidean distance between the points $p$ and $q$. The (undirected) Hausdorff distance $D_H(P,Q)$ is defined as maximum of the two directed distances: $D_H(P,Q) = \max\{d_H(P,Q), d_H(Q,P)\}$. Efficient sequential algorithms are known for the Hausdorff distance computation for $P$ and $Q$ being discrete point sets, sets of non-intersecting straight line segments [2] and for algebraic parameterized curve segments [3].

In this paper we show that the Hausdorff distance for two sets of non-intersecting line segments can be computed efficiently in parallel within $O(\log^2 n)$ time using $O(n)$ processors in the CREW-PRAM computation model.

Due to the current trend in hardware development, where the performance increase is achieved through additional computing units (processor kernels) instead of increase in CPU speed, parallel algorithms have gained new popularity in the algorithm development. Some graphic cards support general computations on graphics hardware (GPGPU) which comprises up to several hundreds of parallel processing units, which even more motivates for development of parallel algorithms.

In this paper we use the PRAM as model of computation for evaluating the theoretical performance of the algorithm, since it exposes the principal parallelism of the problem instead of concentrating on specific technical details of some hardware. The main contribution of this paper – the algorithm for the red-blue segment intersection problem – can be efficiently implemented on most of the currently available parallel hardware platforms, such as GPGPU or multicore CPUs. Section 3 gives a brief note on practical aspects of the algorithm.

We briefly recapitulate here the key steps of the sequential algorithm for computing the directed Hausdorff distance between two sets $P$ and $Q$ of non-intersecting line segments from [2]: (1) Construct the Voronoi diagram $VD(Q)$ of the set $Q$; (2) For each endpoint $p$ of a segment in $P$ find its closest segment in $Q$ using $VD(Q)$ and compute the distance from $p$ to that segment; (3) Determine the so-called "critical points" on the edges of $VD(Q)$; (4) For each critical point $q$ compute the distance from $q$ to its nearest segment in $Q$; (5) Return the maximal distance of endpoints and critical points.

The authors show that the critical points, i.e., the points where the directed Hausdorff distance $d_H(P,Q)$ can be attained, besides the endpoints of the segments in $P$, are the intersection points of $P$ with the Voronoi edges of $Q$. Furthermore, they prove that for each edge of $VD(Q)$ only the extreme intersection points, i.e., the first and the last intersection point along the curve segment, are critical points (s. Figure 1), thus reducing the total number of critical points to $O(n)$. For $x$-monotone curves the extreme intersection points are the points with the minimal and maximal $x$-coordinate. A non-$x$-monotone parabola segment can be split into two $x$-monotone segments at the point with the vertical tangent. Thus, in the following we can assume that all Voronoi edges are $x$-monotone.

Considering the steps of the sequential algorithm we observe that the parallelization of steps (4) and

*This research was performed in scope of the DFG-project "Parallel algorithms in computational geometry with focus on shape matching" under the contract number AL 253/7-1.

†Institute of Computer Science, Freie Universität Berlin, {alt,scharf}@mi.fu-berlin.de

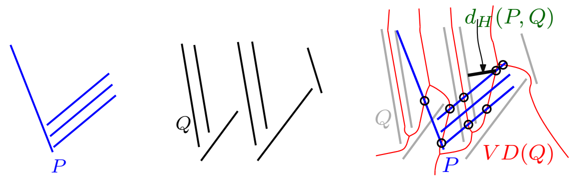


Figure 1: Two sets of line segments representing trademark images. The Hausdorff distance is attained at a critical point on a Voronoi edge.

(5) is straightforward. A parallel algorithm for computing the Voronoi diagram of a set of line segments (Step (1) of the algorithm) is given in [9]. It runs in $O(\log^2 n)$ time on a $O(n)$ processor CREW-PRAM and uses the divide-and-conquer technique. For a given planar subdivision with $n$ vertices a point location data structure supporting $O(\log n)$-time queries can be constructed in $O(\log n)$ time on an EREW-PRAM with $O(n)$ processors [11]. Thus, step (2) can be performed efficiently in parallel.

For determining the critical points, i.e., intersection points between segments and Voronoi edges, in Step (3) the sequential algorithm uses the plane sweep technique. Clearly, the plane sweep technique is inherently sequential, and we therefore need different tools to compute the critical points in parallel. For the related intersection detection and intersection reporting problems, i.e., given $n$ line segments in the plane, determine if any two of them intersect, or find all pairwise intersections, there exist efficient parallel algorithms see e.g., [5, 8, 10].

We call the intersection problem arising in the Hausdorff distance computation, which also may be of independent interest, the *first-last intersection problem* (defined below). Since Voronoi edges of a set of line segments can be line or parabola segments, we want our algorithm for this intersection problem to work for more general sets of curve segments:

**Definition 1** *Two sets of curve segments $A$ and $B$ are called* well-behaved *if every segment in $A \cup B$ is $x$-monotone; no two segments of the same set have a common point except possibly common endpoints; any two segments from different sets intersect at most twice; all intersections between any two segments can be computed in constant time; and for every segment we can compute in constant time for a given $x$-coordinate the corresponding $y$-coordinate.*

Observe that the set $P$ of line segments and the possibly split Voronoi edges of $Q$ are two well-behaved sets. The problem of finding critical points for the Hausdorff distance can then be formulated as:

**Problem 1 (First-Last Intersection Problem)** *Given two well-behaved sets $A$ and $B$ of curve segments in the plane, for each segment $a \in A$ find the intersection points of $a$ with the segments from the set $B$ with the smallest and with the largest $x$-coordinate.*

All above mentioned line segment intersection parallel algorithms utilize the *segment tree* data structure (see e.g. [1]), which is also used in this paper.

The first-last intersection problem arises also as a part of Hausdorff Voronoi Diagrams computation for non-crossing objects. Dehne et al. present in [7] a parallel algorithm for this sub-problem for coarse grained parallel architectures with total work of $O(n \log^3 n)$ and memory requirement of $O(n \log^2 n)$. Their algorithm works with segment trees and builds additionally a secondary segment tree structure.

Here we present an algorithm with a total work of $O(n \log^2 n)$ and a memory requirement of $O(n \log n)$ on an arbitrary number of processors. Our algorithm utilizes a different secondary data structure – the interval tree, which allows us to save a $O(\log n)$ factor in memory and time bounds. We can show that for a given set of $n$ intervals an interval tree can be constructed in $O(\log n)$ time on $O(n)$ processors in the CREW-PRAM model using an efficient sorting algorithm.

The theoretical analysis of our algorithm is summarized in the following theorem:

**Theorem 1** *Let $A$ and $B$ be two well-behaved sets of curve segments in the plane with $|A| + |B| = n$. The first-last intersection problem for $A$ and $B$ can be solved on a CREW-PRAM using $O(\log n)$ time, $O(n \log^2 n)$ operations, and $O(n \log n)$ space.*

Clearly, the total work of $O(n \log^2 n)$ claimed in Theorem 1 can alternatively be performed in $O(\log^2 n)$ time on $O(n)$ processors, which corresponds to processor and time requirements for the Voronoi diagram computation. Thus, Theorem 1 together with the above mentioned previous work completes the proof of Theorem 2:

**Theorem 2** *Given two sets $P$ and $Q$ of $n$ line segments, such that no two segments of the same set intersect, except possibly at the endpoints, the Hausdorff distance $D_H(P,Q)$ can be computed on a CREW-PRAM using $O(\log^2 n)$ time, $O(n \log^2 n)$ operations, and $O(n \log n)$ space.*

## 2 A Parallel Algorithm for the First-Last-Intersection Problem

Let $A$ and $B$ be two well-behaved sets (red and blue resp.) of curve segments in the plane with $|A|+|B| = n$. Here we describe how to find for each segment $a \in A$ the intersection point with $B$ with the minimal $x$-coordinate, the intersections with the maximal $x$-coordinate can be determined symmetrically.

Our algorithm begins with the construction of a segment tree $T$ for the set $A \cup B$. Recall that each node

$v$ of a segment tree represents a horizontal interval and stores a so-called *cover-list* – the list of segments that cover completely the interval of $v$ but not the interval of the parent of $v$; and an *end-list* – the list of segments that have an endpoint in the interval of $v$.

**Step 1.** *Build a segment tree $T$ for $A \cup B$. For each node $v$ of $T$ construct separate cover-lists $C_A(v), C_B(v)$ for the sets $A$ and $B$ respectively, and separate end-lists $E_A(v), E_B(v)$. Sort $C_A(v)$ and $C_B(v)$ by $y$-coordinate for all nodes $v$ in parallel.*

Since the initial sets are intersection free, the $y$-order within the cover-lists of each color is well-defined.

Chazelle showed in [6] that if two line segments of a set intersect, then in the corresponding segment tree there must be a node $v$ such that either both segments are in $C(v)$ or one is in $C(v)$ and the other in $E(v)$. It is easy to see that a similar statement holds for well-behaved curves as defined in Section 1.

In the two-set setting this means that if a red segment $a$ and a blue segment $b$ intersect, then there must be a node $v$ in $T$ such that either (1) $a \in C_A(v)$ and $b \in C_B(v)$, (2) $a \in E_A(v)$ and $b \in C_B(v)$, or (3) $a \in C_A(v)$ and $b \in E_B(v)$. The following steps of the algorithm deal with each of these three cases. Whereas the handling of the first two cases (Step 2) is a modification of the corresponding steps of the algorithm in [10], the third case demands additional processing (Step 3).

**Step 2.** *For each node $v$ of $T$ and each segment $a \in C_A(v) \cup E_A(v)$ do in parallel: Find the neighbors of $a$ in $C_B(v)$ with respect to the $y$-order at the $x$-coordinate of the leftmost point of $a$ within the interval of $v$. Compute the intersections of $a$ with its neighbors, if any exist, and record the one with the minimum $x$-coordinate.*

For type (3) of intersections we observe that for a given segment $b \in E_B(v)$ we can easily determine the lowest, $a_1$, and the highest, $a_2$, red segment in $C_A(v)$ intersected by $b$ within the interval of $v$, using binary search on $C_A(v)$. Clearly, all red segments in $C_A(v)$ between $a_1$ and $a_2$ are also intersected by $b$ forming a set of consecutive ranks in $C_A(v)$ with respect to its ascending order in $y$-direction. This set we call the *rank interval* of $b$, which has a constant size representation by $a_1, a_2$.

In the following we are going to find for each node $v$ of $T$ the set $I(v)$ of the rank intervals for all $b \in E_B(v)$. Then we process and narrow the rank intervals in $I(v)$ with the purpose to avoid multiple intersections of a blue segment with a red one, and thus, to include at most $O(\log n)$ intersection points for each $a \in C_A(v)$. A detailed consideration shows that all possible configurations between $b$ and its rank interval are the ones shown in Figure 2.

**Step 3.1.** *For each node $v$ in $T$ and each $b \in E_B(v)$ do in parallel: Find the rank interval of $C_A(v)$ intersected by $b$. Split $b$ if necessary, so that each red segment is intersected at most once by each subsegment of $b$. Record the resulting interval(s) in the interval set $I(v)$.*

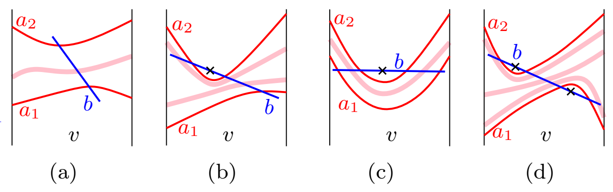


Figure 2: Configurations of the intersection points of $b$ with the first and last red segments of its rank interval.

**Step 3.2.** *For each node $v \in T$ in parallel construct an interval tree $T_I(v)$ for the set of rank intervals $I(v)$.*

The blue segments whose rank intervals are stored in the same node $u$ of $T_I(v)$ all intersect the same red segment $a \in C_A(v)$ – the segment with the reference rank of $u$, hereafter called the *reference segment* of $u$. Thus, we can order them by the $x$-coordinate of their intersection with $a$. If two blue segments $b_1, b_2$ intersect some red segments the order of the intersection points will be the same on all of these red segments. Therefore, if $b_1$'s intersection with $a$ has a lower $x$-coordinate than that of $b_2$, we can remove from the rank interval of $b_2$ those elements that are in the interval of $b_1$ without losing significant intersection points. Thus, by sorting the rank intervals with respect to the reference segment and computing prefix-minima and maxima, we can prune the blue rank intervals so that the remaining intervals have the following properties: The rank intervals stored in one node of $T_I(v)$ are disjoint, i.e., the rank of every segment $a \in C_A(v)$ is contained in at most one interval of a single node. The number of intervals in $T_I(v)$ is at most twice the original number. The rank of every $a \in C_A(v)$ is contained in at most $O(\log n)$ intervals in $T_I(v)$.

**Step 3.3.** *For each node $v$ in $T$ and each node $u$ in $T_I(v)$ in parallel prune the rank intervals in $u$.*

Finally, we can reorganize $T_I(v)$ and compute the intersection points:

**Step 3.4.** *For each node $v \in T$ in parallel rebuild the interval tree $T_I(v)$ for the new intervals.*

**Step 3.5.** *For each red segment $a \in C_A(v)$ in parallel find all $b \in E_B(v)$ whose intervals in $T_I(v)$ contain the rank of $a$. Record the intersection point with the minimal $x$-coordinate.*

The last step is to select the intersection points to report from the candidate intersections:

**Step 4.** *For each $a \in A$ in parallel find the intersection point with the minimal $x$-coordinate from the candidate points computed in Steps 2 and 3.*

Concerning the proof of Theorem 1: The correctness of the algorithm is already explained along the description of the algorithm. A segment tree uses

$O(n \log n)$ space and the total size of all interval trees is $O(n)$. Step 1 can be performed in $O(\log n)$ time on an $O(n)$ processor CREW-PRAM, see [1]. In Step 2 we assign a processor to each occurrence of a segment $a$ in cover- and end-lists of $T$ and find all candidates of type (1) and (2) in $O(\log n)$ time with $O(n \log n)$ processors using binary search. Similarly, we can show that Steps 3.1 – 3.5 can be performed within the same time and processor bounds, where the construction of all interval trees is performed in parallel. Finally, finding the maximum of the $O(n \log n)$ candidate values in Step 4 can clearly be performed within the claimed resource bounds.

## 3 Practical Notes

The algorithm described in Section 2 breaks down to the following basic (global) operations on arrays: sorting, prefix minimum and maximum computation (also known as scan operation), and a so-called compact operation, which is used to remove elements (e.g., intervals). Additionally, for each segment the following (local) operations are performed independently and conflict free: binary search and arithmetic computations of constant complexity.

All of these operations can be efficiently implemented for the following currently available parallel architectures with shared memory: GPGPU – parallel computations on graphics hardware, and the parallel external memory model (PEM) [4] – one of the models attempting to reflect the demands of modern parallel multicore CPUs. Thus, our algorithm can be directly implemented efficiently for both of these architectures.

Another popular parallel architecture is cluster computing, usually modeled by the *coarse grained multicomputer* (*CGM*) model (as in [7]). In this model the communication between processors is assumed to be very expensive and the communication cost is typically expressed in the number of global sort operations performed by the algorithm. We can show that the data structure used by our algorithm can be distributed over $p$ processors in a similar manner as it is done in [7]. Thus, our algorithm performs a total work of $O(n \log^2 n)$ and has $O(1)$ global communication rounds in the CGM model.

## 4 Concluding Remarks

The algorithm for the first-last intersection problem presented here can further be accelerated to perform Step 2 in $O(\log n)$ time using $O(n)$ processors by applying the fractional cascading technique [5]. This technique simplifies the iterative search for a key in multiple ordered lists and allows to perform the search of a segment $a$ in all $C_B$-lists of $T$ in $O(\log n)$ time, i.e., taking $O(\log n)$ time on $O(n)$ processors for Step 2. Steps 1 and 4 have already these time and processor bounds. But it is not clear whether the performance of Steps 3.1 to 3.5, i.e., finding the first intersection of a red sub-segment from a cover-list with blue sub-segments from an end-list in a node of the segment tree, can be improved.

A further interesting related problem is matching geometric shapes under transformations (e.g., translations, rotations, scalings) with respect to the Hausdorff distance, i.e., find a transformation of one of the shapes such that the Hausdorff distance is minimized, for sequential algorithms see [2].

## References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3(1):293–327, 1988.

[2] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13:251–265, 1995.

[3] H. Alt and L. Scharf. Computing the Hausdorff distance between curved objects. *Int. J. Comput. Geometry Appl.*, 18(4):307–320, August 2008.

[4] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *Proc. 20th symposium on Parallelism in algorithms and architectures*, SPAA'08, pages 197–206, 2008.

[5] M. J. Atallah, R. Cole, and M. T. Goodrich. Cascading divide-and-conquer: a technique for designing parallel algorithms. *SIAM J. Comput.*, 18(3):499–532, 1989.

[6] B. Chazelle. Intersecting is easier than sorting. In *STOC'84: Proc. 16th annual ACM symposium on Theory of computing*, pages 125–134, 1984.

[7] F. Dehne, A. Maheshwari, and R. Taylor. A coarse grained parallel algorithm for Hausdorff Voronoi diagrams. In *Int. Conf. on Parallel Processing (ICPP)*, pages 497–504, 2006.

[8] M. T. Goodrich. Intersecting line segments in parallel with an output-sensitive number of processors. In *Proc. 1st ACM symposium on Parallel algorithms and architectures*, SPAA '89, pages 127–137, 1989.

[9] M. T. Goodrich, C. Ó'Dúnlaing, and C.-K. Yap. Constructing the Voronoi diagram of a set of line segments in parallel. *Algorithmica*, 9(2):128–141, 1993.

[10] M. T. Goodrich, S. B. Shauck, and S. Guha. Parallel methods for visibility and shortest path problems in simple polygons (preliminary version). In *Proc. 6th symposium on Computational geometry*, SCG '90, pages 73–82, 1990.

[11] R. Tamassia and J. S. Vitter. Optimal parallel algorithms for transitive closure and point location in planar structures. In *Proc. 1st ACM symposium on Parallel algorithms and architectures*, SPAA'89, pages 399–408, 1989.

# Hardness Results on Curve/Point Set Matching with Fréchet Distance

Paul Accisano* Alper Üngör *

**Abstract**

Let $P$ be a polygonal curve in $\mathbb{R}^d$ of size $n$, and $S$ be a point set of size $k$. We consider the problem of finding a polygonal curve $Q$ on $S$ such that all points in $S$ are visited and the Fréchet distance from $P$ is less than a given $\varepsilon$. We show that two versions of this problem are NP-complete: one in which points of $S$ are allowed to be revisited and one in which they are not.

## 1 Introduction and Background

We consider the problem of finding a polygonal curve $Q$ on a point set $S$ such that all points in $S$ are visited and the Fréchet distance from a given polygonal curve $P$ is less than a given $\varepsilon$. Figure 1 shows an example problem instance and its solution.

The Fréchet distance problem has been well studied. Alt and Godau [1] showed that, given two polygonal curves of length $n$ and $m$, deciding whether they have Fréchet distance less than a given $\varepsilon$ can be accomplished in $O(nm)$ time. They also showed that finding the exact Fréchet distance between the two curves can be done in $O(nm\log(nm))$ time.

Maheshwari et al. [5] examined the following variant of the Fréchet distance problem, which we refer to as the Curve/Point Set Matching (CPSM) problem. Given a polygonal curve $P$ of length $n$, a point set $S$ of size $k$, and a number $\varepsilon > 0$, determine whether there exists a polygonal curve $Q$ on a subset of the points of $S$ such that $\delta_F(P,Q) \le \varepsilon$. They gave an algorithm that decides this problem in time $O(nk^2)$, as well as an algorithm to compute the curve of minimal Fréchet distance in time $O(nk^2\log(nk))$ using parametric search.

Wylie and Zhu [6] also explored the CPSM problem from the perspective of discrete Fréchet distance, which only takes into account the distance at the vertices along the curves. They formulated four versions of the CPSM problem depending on whether or not points in $S$ were allowed to be visited more than once (Unique vs. Non-unique) and whether or not $Q$ was required to visit all points in $S$ at least once (All-Points vs. Subset). They showed that, under the discrete Fréchet distance metric, both non-unique versions were solvable in $O(nk)$ time, and both unique versions were NP-complete.

*Dept. of Computer & Info. Sci. & Eng., University of Florida, {accisano, ungor}@cise.ufl.edu

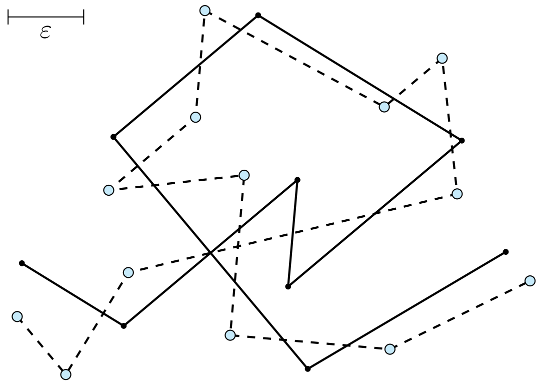


Figure 1: A problem instance and its solution. The input is the solid line and the circle points, and the solution is the dotted line.

Driemel and Har-Peled [3] studied a similar problem in which two curves are given, and the goal is to find an optimal set of "shortcuts" that minimize the Fréchet distance. Our problem differs from theirs in that points of $S$ can be visited in any order, whereas in their problem, the order is prescribed.

In this paper, we show that the Continuous All-Points versions of the CPSM problem, both Unique and Non-unique, are NP-complete (Table 1).

| | | Discrete | Continuous |
|---|---|---|---|
| Subset | Unique | NP-C [6] | Open |
| | Non-Unique | P [6] | P [5] |
| All-Pts | Unique | NP-C [6] | NP-C* |
| | Non-Unique | P [6] | NP-C* |

Table 1: Eight versions of the CPSM problem and their complexity classes. New results are starred.

## 2 Preliminaries

Given two curves $P, Q : [0,1] \to \mathbb{R}^d$, the *Fréchet distance* between $P$ and $Q$ is defined as $\delta_F(P,Q) = \inf_{\sigma,\tau} \max_{t\in[0,1]} \|P(\sigma(t)), Q(\tau(t))\|$, where $\sigma, \tau : [0,1] \to [0,1]$ range over all continuous non-decreasing surjective functions [4].

For a given point $p \in \mathbb{R}^d$ and a real number $\varepsilon > 0$, let $\mathcal{B}(p,\varepsilon) \equiv \{q \in \mathbb{R}^d : \|pq\| \le \varepsilon$ denote the *ball* of radius $\varepsilon$ centered at $p$, where $\|\cdot\|$ denotes Euclidean distance. For a line segment $L \subset \mathbb{R}^d$, let $\mathcal{C}(L,\varepsilon) \equiv \bigcup_{p\in L} \mathcal{B}(p,\varepsilon)$

denote the *cylinder* of radius $\varepsilon$ around $L$. Note that a necessary condition for two curves $P$ and $Q$ to have Fréchet distance at most $\varepsilon$ is that the vertices of $Q$ must all lie within the cylinder of some segment of $P$.

Our NP-completeness result is obtained via a reduction from a restricted version of the well-known 3SAT problem. The 3SAT problem takes as input a boolean formula with clauses of size 3, and asks whether there exists an assignment to the variables that makes the formula evaluate to TRUE. If we restrict the input to formulas in which each literal occurs exactly twice, the problem becomes the (3,B2)-SAT problem. Furthermore, we make the restriction that no two clauses have two literals in common. The problem was shown to be NP-complete in [2].

## 3 The Reduction

Let $\Phi$ be a formula given as input to the (3,B2)-SAT problem. We construct a polygonal curve $P$ and a point set $S$ such that $\Phi$ is satisfiable if and only if there exists polygonal curve $Q$ whose vertices are exactly $S$ with Fréchet distance at most $\varepsilon$ from $P$.

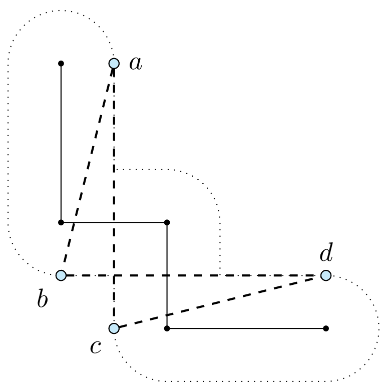
First, we construct a gadget consisting of components of $P$ and $S$ that will force any algorithm to choose between two possible polygonal paths. The gadget is constructed in such a way that these two choices are the only possible polygonal paths along the gadget's component of $S$ with Fréchet distance at most $\varepsilon$ from $P$. Then, we create a series of points in $S$ to represent the clauses in $\Phi$, one point for each clause. For each variable, a gadget will be placed that goes through four points, which represent the four clauses in which the variable appears. The gadget will be placed such that only the clauses in which the variable's positive or negative instances occur are reachable, but not both. Once this has been done for each variable in $\Phi$, any polygonal curve $Q$ whose vertices are exactly $S$ with $\delta_F(P,Q) \leq \varepsilon$ will correspond to an assignment to the variables of $\Phi$ in which every clause is satisfied, thus making the formula evaluate to TRUE. Furthermore, if no such curve exists, then there can be no such satisfying assignment for $\Phi$.

### 3.1 Separation Gadget

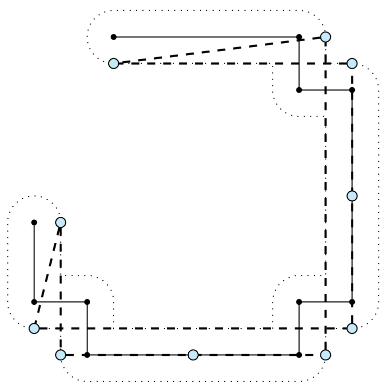
We begin the description of our main gadget with an example, which we later generalize. Consider the problem instance shown in Figure 2a, with $S = \{a, b, c, d\}$. It is clear that the answer to this instance is "no"; no polygonal curve on $S$ with $\delta_F(P,Q) \leq \varepsilon$ can visit both $b$ and $c$. However, suppose this $P$ and $S$ were part of a larger problem instance. Then suppose that other segments of $P$ come within $\varepsilon$ of $b$ and $c$. The answer to the problem instance is no longer so obvious. Even if both points cannot be reached the first time they are encountered, it is possible that whichever point was skipped could be covered in the future. This creates the fundamental difficulty that leads to our reduction.

Figure 2b shows an extension of the previous configuration, with more corners, all symmetrically the same as the first. Note how we have not increased the number of options; there are still only two possible paths to take. We can add as many of these corners as we like without breaking this property, as long as they all bend in the same direction.

The corner points must be placed very precisely to ensure the above properties hold. Because their position is so constrained, using them to represent elements of $\Phi$ in our construction would be difficult. At each corner, the two path possibilities alternate between the boundary of the cylinders and the interior. As shown in Figure 2b, extra points in the cylinder interior are still only visible from the other interior points, and therefore we can add as many as we like



(a) The two dashed curves are the only possible curves on $S$ with Fréchet distance at most 1 from the given (solid) curve.


(b) More corners have been added. The two interior points are only reachable on different curve possibilities.

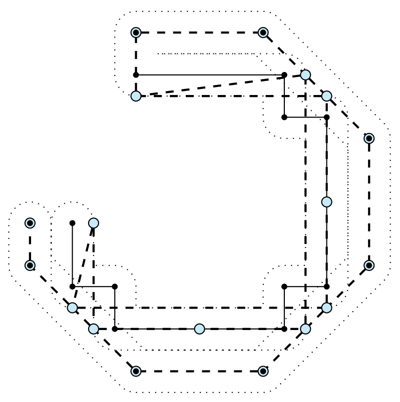

(c) All except one of the two interior points are covered, regardless of which path is chosen.

Figure 2: The separation gadget, step by step.

without affecting the path possibilities. Thus, by extending the segments between the corners, we can create large regions in which we can place points that are only reachable along one of the two possibilities. These points will represent clauses in our construction.

There is still a problem to be addressed; as more corners are added, more points are created that would be skipped by the chosen path. We would like to create a construction that forces a choice among only the points in the cylinder interiors and that ensures all the corner points will be visited regardless of which path is chosen. To accomplish this, after the last corner of the gadget, we can have $P$ loop back around along the outer edge, covering all the corner points without covering any of the interior points (Figure 2c.)

Figure 3 shows an example usage of the gadget. The points in the cylinder interiors represent the clauses in which the variable appears. Only one set of clause points, either the clauses in which the positive or negative literals appear, can be reached. However, all the corner points will always be covered. The full construction will include one of these gadgets for every variable, with each one passing through the points corresponding to clauses containing the variable's positive and negative literals. Note that, while our initial construction used only right angles, our corner constructs can be modified to bend at any non-acute angle $\alpha$. In the following sections, we refer to these constructs as $\alpha$-corners. In our full paper, we give a more precise definition of the construct.

## 3.2 Construction

We begin by adding an initial set of points to $S$ which we refer to as "clause points" $C_1, \ldots, C_n$, one for each clause in $\Phi$. We position these points so that they lie on a circle, equally spaced. We then perform the following procedure for each variable $v_i$ in $\Phi$, building the construction incrementally. Let $x$ and $y$ be the clauses in which the positive literals of $v_i$ occur, and $z$ and $w$ be the clauses in which the negative literals occur. We begin by positioning an $\alpha$-corner so that the extension of the last segment of the forward path passes through $C_x$ and $C_y$. An extra point, which we refer to as a split point, is added on the boundary of the first forward path segment to allow the splitting of the two possible paths. From there, both the forward and return paths are extended through the clause ring, with the forward path crossing through $C_x$ and $C_y$.

On the opposite side, outside the convex hull of all points in $S$ so far, another $\alpha$-corner is added, bending the path toward $\overline{C_z C_w}$. More $\alpha$-corners, all bending in the same direction, are added as needed until one can be placed such that the forward path passes through $C_z$ and $C_w$. Note that there must be an odd number of $\alpha$-corners in order to ensure that $C_x$, $C_y$ and $C_w$, $C_z$ are reachable on different curve possibilities. Once

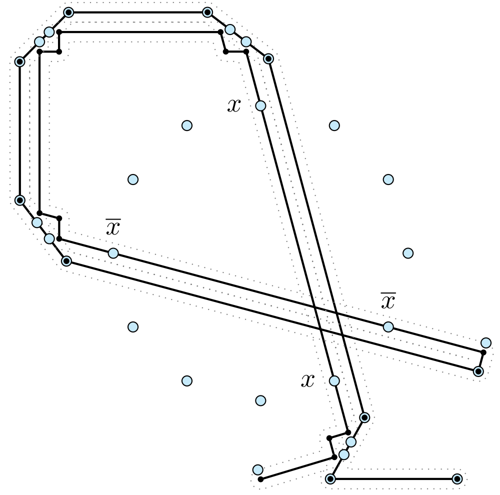


Figure 3: A partial construction for a formula with 12 clauses, showing the gadget for a single variable. The only two valid paths visit either the positive literal's clauses or the negative literal's clauses.

the paths have been extended through the clause ring and outside the convex hull, another split point is added on the boundary to collapse the curve possibilities. Finally, the forward path is linked to the return path, and the joint is added to $S$. At the end of the return path, more segments of $P$ are added, with each joint being added to $S$, in order to move to the next variable's clause points. To avoid interfering with previously placed gadgets, we place all new segments and points outside the convex hull of all previously placed points in $S$. Figure 4 shows a completed construction for a simple formula.

To ensure our construction is always possible, we must enforce certain properties. First, the circle on which the clause points are placed must have a radius of at least $n^2\varepsilon$. Let the *clause strip* along clauses $i$ and $j$ denote the region within $7\varepsilon$ of the line $\overline{C_i C_j}$. Our choice of radius ensures that, if two clause strips are parallel, $\alpha$-corners placed inside will not interfere with each other.

We also require that, with the exception of those directly adjacent to clause points, all $\alpha$-corners are placed entirely outside all strips along all clause pairs, so as not to block future pieces. Those $\alpha$-corners that are adjacent to clause points will lie entirely inside the corresponding clause strip, but must be placed outside all other clause strips. Note that this is always possible; beyond $4n^2\varepsilon$ units from the center of the clause ring, no clause strip intersects any other. Strips of different angles will grow further and further apart, creating regions of arbitrary size between them.

## 4 Result

**Lemma 1** *There exists a polygonal path $Q$ on $S$ with $\delta_F(P, Q) \leq \varepsilon$ that visits every point in $S$ if and only* *if $\Phi$ is satisfiable.*

**Proof.** For the forward direction, assume $\Phi$ has a satisfying assignment. It is easy to see that our construction always has a polygonal path $Q$ on $S$ with $\delta_F(P,Q) \leq \varepsilon$ that will visit every non-clause point; $\alpha$-corners are constructed specifically to ensure this. If $\Phi$ has a satisfying assignment, then one of the two path possibilities in each variable construct will cover the clause points corresponding to the clauses satisfied by that variable, resulting in all clause points being visited.

For the backward direction, let $Q$ be a polygonal path whose vertices are exactly $S$ with $\delta_F(P,Q) \leq \varepsilon$. By constructing each variable construct completely outside the convex hull of all previously placed points of $S$, we have ensured that any $Q$ with $\delta_F(P,Q) \leq \varepsilon$ must follow the path we have laid out. Each variable construct forces a choice between two paths, representing a true or false value for that variable. Since $Q$ visits each clause point, the path taken in each variable construct represents an assignment to the variables that satisfies $\Phi$. $\square$

It is straightforward to show that five $\alpha$-corners are sufficient to move between any two strips, which means the construction is of polynomial size. This leads to the following result.

**Theorem 2** *The Non-unique All-points Continuous CPSM Problem is NP-complete.*

In the construction, the only points that occur more than once are the clause points and the inner $\alpha$-corner points. In all occurrences of both cases, the next point is always reachable from the previous point. Thus, for this class of problem instances, any solution to the Non-unique version of this problem can be converted to a solution to the Unique version by simply skipping the points that have already been visited. This shows that the same reduction applies to the Unique version.

**Corollary 3** *The Unique All-points Continuous CPSM Problem is NP-complete.*

## 5 Approximation Algorithm

We have recently developed a 3-approximation algorithm for the Non-unique All-points Continuous CPSM problem. However, due to space limitations, that algorithm will be presented elsewhere. Our approximation algorithm relies on an $O(nk^2)$ time algorithm that solves the following restricted version of the problem: Given a polygonal curve $P$ and a point set $S$, find the curve of minimal Fréchet distance whose vertices are exactly $S$, with the additional restriction that each point $s \in S$ is visited *at its closest segment* (as well as possibly at other segments).

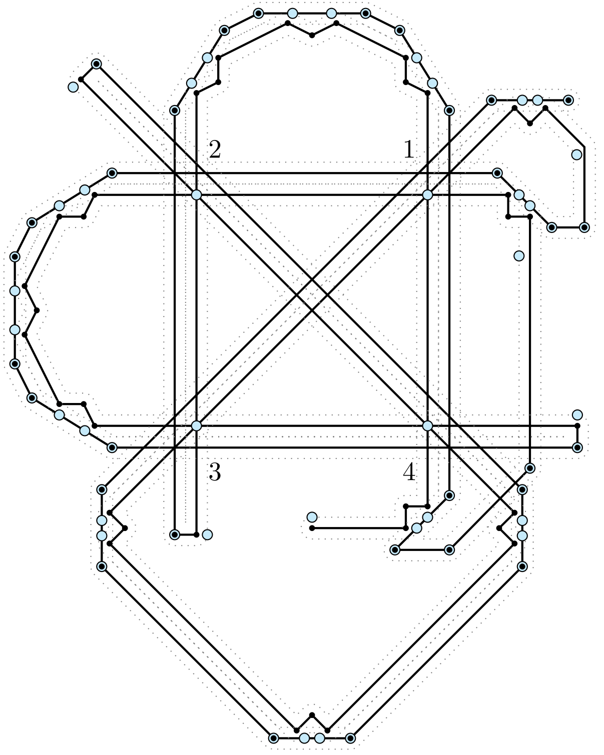


Figure 4: A completed construction for the formula $\Phi = (x \vee y \vee z) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee z) \wedge (x \vee \overline{y} \vee \overline{z})$. The upper right clause point represents the first clause, and the second, third, and fourth follow counterclockwise.

## References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. of Comp. Geom. & Appl.*, 5(01n02):75–91, 1995.

[2] P. Berman, M. Karpinski, and A. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *El. Coll. on Comp. Complex.*, (049), 2003.

[3] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the fréchet distance with shortcuts. In *Proc. of Symp. on Disc. Alg.*, pages 318–337, 2012.

[4] George Ewing. *Calculus of variations with applications.* Dover Publications, New York, 1985.

[5] A. Maheshwari, J. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Staying close to a curve. In *Canadian Conf. on Comp. Geom.*, pages 55–58, 2011.

[6] T. Wylie and B. Zhu. Discretely following a curve (short abstract). In *Computational Geometry: Young Researchers Forum (CG:YRF)*, 2012.

# Differential-Based Geometry Modeling

Konrad Polthier[*]

**Abstract**

We present an interactive modeling framework for 3D shapes and for texture maps. The technique combines a differential-based deformation method with the idea of geometry brushes that allow to interactively apply modifications by painting on the geometry. Whereas most other deformation techniques demand the designer to define and move hard constrained regions on the surface, the proposed modeling process is similar to sculpting.

Geometry brushes allow the user to locally manipulate the metric, enlarge, shrink or rotate parts of the surface and to generate bumps. In a similar way it is possible to modify texture maps, or more generally, arbitrary tensor maps on surfaces. The local changes are globally integrated and visualized in real-time.

The presented mathematical techniques such as discrete differential forms, discrete Hodge decomposition, and integrability issues are fundamental also for a wide range of local and global applications in geometry processing including recent progress in surface parametrization.

[*]Department of Mathematics, FU Braunschweig, Germany. `Konrad.Polthier@fu-berlin.de`

# New Results on Convex Stabbers

Lena Schlipf*

**Abstract**

In this paper, we prove the problem of stabbing a set of disjoint bends by a convex stabber to be NP-hard. We also consider the optimization version of the convex stabber problem and prove this problem to be APX-hard for sets of line segments.

## 1 Introduction

Consider a finite set of geometric objects in the plane. We call this set stabbable if there exists a convex polygon whose boundary intersects every object. The boundary is then called *convex stabber*.

The problem of finding a convex stabber was originally proposed by Tamir in 1987 [5]. Arkin et al. [1] proved this problem to be NP-hard when the geometric objects are line segments or similar copies of a given convex polygon. This paper, in fact, can be considered as a continuation of the paper of Arkin et al. We will show that the problem of finding a convex stabber for a set of disjoint simple polygons is NP-hard. Actually, we even show that it is already hard to stab a set of disjoint bends. Additionally, we study the optimization version: Given a finite set of geometric objects in the plane, compute the maximum number of objects that can be stabbed with the boundary of a convex polygon. We prove this problem to be APX-hard when the objects are line segments.

**Notation.** Two line segments with a common endpoint are called a *bend*. We say that a convex stabber stabs or *traverses* the given objects.

## 2 Convex Stabbers for Disjoint Polygons

We consider the following problem: Given a set of disjoint bends in the plane, is there a convex stabber that intersects every bend of the set?

We prove this problem to be NP-hard. We reduce from planar, monotone 3SAT which was shown to be NP-hard by de Berg and Khosravi [2]. A *monotone* instance of 3SAT is an instance where each clause has either only positive or only negative variables. In the following we call a clause that contains only positive variables a *positive clause* and a clause that contains only negative variables a *negative clause*. Planar, monotone 3SAT remains NP-hard even when a monotone rectilinear representation is given. In a monotone rectilinear representation the variable and clause gadgets are represented as rectangles. All variable rectangles lie on a horizontal line. The edges connecting the clause gadgets to the variable gadgets are vertical line segments and no two edges cross. All positive clauses lie above the variables and all negative clauses lie below the variables.

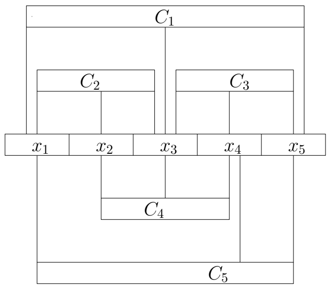


Figure 1: A monotone rectilinear representation of the 3SAT instance $C = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ where $C_1 = x_1 \vee x_3 \vee x_5$, $C_2 = x_1 \vee x_2 \vee x_3$, $C_3 = x_3 \vee x_4 \vee x_5$, $C_4 = \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}$, and $C_5 = \overline{x_1} \vee \overline{x_4} \vee \overline{x_5}$.

Given a monotone rectilinear representation $\phi$ of a 3SAT instance, we construct a set of bends $\mathcal{B}$ such that there exists a convex stabber for $\mathcal{B}$ if and only if $\phi$ is satisfiable. Let $m$ be the number of clauses and $n$ be the number of variables. Let the number of positive clauses and negative clauses be $m_1$ and $m_2$, respectively. (The variable and the clause gadgets are basically constructed in the same way as in [1].)

**Variable gadgets.** A variable gadget consists of a line segment and three points (degenerate bends), see Fig. 2.

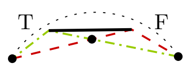


Figure 2: A variable gadget. (The dashed and the dashed-dotted segments and the dotted circular arc are not part of the construction.)

There are two ways to traverse these points and the segment depending on the order in which the middle point is traversed: one corresponds to setting the

*Institute of Computer Science, Freie Universität Berlin, Germany. schlipf@mi.fu-berlin.de. This research was supported by the DFG within the Priority Programme 1307 Algorithm Engineering.

variable to True (the dashed-dotted one), the other to setting the variable to False (the dashed one).

The variable gadgets are fitted into a circular arc by putting the non-middle points on the arc; the middle point and the segment lie inside the circle. Each positive variable gadget is placed into an arc of $1/(16m_1)$ of a unit circle. Each negative variable gadgets is placed into an arc of $1/(16m_2)$ of a unit circle.

**Clause gadgets.** A clause gadget consists of a line segment and two points. Similarly to the variable gadgets the clause gadgets are fit into a circular arc. The points are put on the arc and the segment lies inside the circle. The positive clause gadgets are fit into an arc of $1/(16m_1)$ of a unit circle; the negative variable gadgets are fit into an arc of $1/(16m_2)$ of a unit circle.

**Positive arc.** We place the gadgets representing a positive variable or a positive clause next to each other on an arc of a unit circle. Since any positive variable and any positive clause gadget is fit into an arc of $1/(16m_1)$, they occupy an arc of one quarter of a unit circle. The gadgets have to be placed in a specific order. Consider the monotone rectilinear representation; the clauses and their corresponding variables are connected via edges. We place a gadget for each edge, representing the variable that this edge connects with a clause. These edges are ordered from left to right and the gadgets are ordered in the same way. Thus, a gadget is placed for every occurrence of a variable in a clause. The clause gadgets are placed to the right of the gadget representing their middle variable.

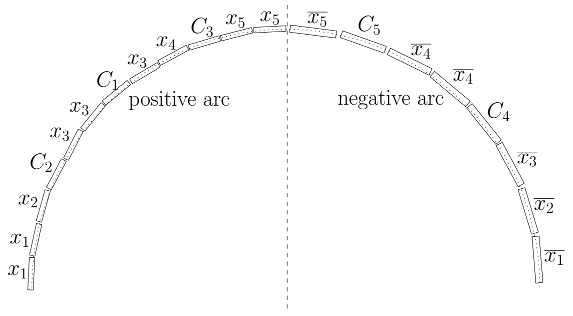


Figure 3: The placement of the gadgets for the instance in Fig. 1 is shown.

**Negative arc.** We place the gadgets representing a negative variable or a negative clause next to each other on an arc of one quarter of a unit circle. The gadgets are ordered in the same way as for the positive arc, but this time from right to left. The negative and the positive arc are placed next to each other (Fig. 3).

**Variable connectors.** To ensure that a stabber traverses all gadgets representing the same variable in the same way, we place $3m$ variable connectors. All gadgets that represent the same variable are connected via segments in a circular manner. The segment touches the True path of one gadget and the False path of the next gadget (Fig. 4). Since on both the positive and the negative arc all variable gadgets representing the same variable lie next to each other, these segments do not intersect anything else. In total, we place one segment for each variable gadget.

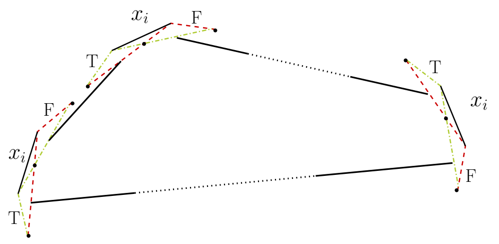


Figure 4: The variable connectors ensure that each variable gadget that represents the same variable is traversed in the same way: either the stabber traverses the True path or the False path.

**Clause connectors.** We place $3m$ more bends in order to connect a clause gadget with its variables. Note that there is a variable gadget for each occurrence of a variable in a clause. The connectors either lie inside the circle or outside. We call them inner connectors and outer connectors, respectively. An inner connector can be a straight line segment whereas an outer connector has to be a bend.

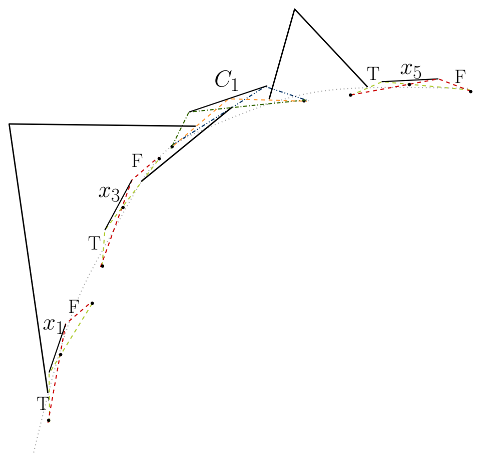


Figure 5: The clause connectors are shown. There are three ways to traverse the clause gadget; each path stabs two out of the three connectors.

The remaining parts of the construction are explained for positive clauses. Negative clauses can be handled similarly. Each clause gadget has two outer connectors and one inner connector. The inner connector connects the clause gadget to the gadget representing its middle variable. Note that this gadget is placed next to the clause gadget in the construction. The other two variables that occur in this clause are connected via outer connectors. One endpoint of each connector lies within the variable gadget as follows: the segment touches the True path through the gadget and does not intersect the False subpath. In every clause gadget, the endpoints of these connectors look the same, see Fig. 5. It can easily be checked that a convex stabber can intersect any two of the three bends but never all three of them. It follows immediately from the monotone rectilinear representation that our construction can be drawn crossing-free; the bends and segments are all pairwise disjoint.

**Correctness.** Assume there exists a satisfying assignment for the 3SAT formula. The convex stabber traverses the variable gadgets according to this assignment. In each clause gadget the stabber can stab two connector bends. Since at least one variable is satisfied in each clause, the stabber can omit the connector connecting the satisfied variable to the clause and stab the other two connectors. Hence, the stabber stabs all bends.

Conversely, assume there is a convex stabber that stabs all bends. Set the variables True or False depending on how the stabber traverse the variable gadget. This is a satisfying assignment for the 3SAT instance: The setting is consistent since the stabber has to omit at least one connector in each clause gadget. These omitted bends have to be stabbed in the variable gadgets; there the stabber can either take the True path or the False path, but not both.

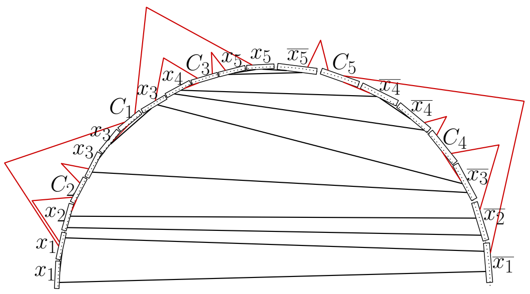


Figure 6: A sketch of the construction for the instance in Fig. 1 is shown.

**Theorem 1** *Let $\mathcal{B}$ a set of disjoint bends in the plane. It is NP-hard to decide whether there exists a convex stabber for $\mathcal{B}$.*

## 3 APX-Hardness

In this section we consider the *maximum convex stabber problem*: Given a set of geometric objects in the plane, compute a convex stabber that stabs the maximum number of these objects.

We prove this problem to be APX-hard if the geometric objects are line segments. We use a reduction from MAX-E3SAT(5) which is a special version of MAX-3SAT where each clause contains exactly 3 literals and every variable occurs in exactly 5 clauses and a variable does not appear in a clause more than once. It is known to be NP-hard to approximate MAX-E3SAT(5) within a factor of $(1-\epsilon)$ for any $\epsilon > 0$ [3]. We first start by reducing MAX-E3SAT(5) to the decision version of the problem (Given a set of line segments and an integer $s$, does there exists a convex stabber that stabs $s$ segments?). The reduction is basically the same as the reduction for the problem of finding a convex stabber for line segments in [1].

We show how to build a set of line segments $\mathcal{L}$, for a 3SAT formula $\phi$, such that there exists a convex stabber for $\mathcal{L}$ if and only if $\phi$ is satisfiable. We use $n, m$ to denote the number of the variables and clauses.

**Variable gadgets.** For each variable we have a gadget that consists of 6 line segments and 18 points. The segments are stacked on top of each other. The 18 points are partitioned into 3 sets of equal size; the points of each set are stacked on top of each other. The stack of segments and the 3 stacks of points are arranged in the same way as in Fig. 2. There are two ways to traverse the gadget (and to stab all segments in the gadget) depending on the order in which the middle stack of points is traversed. One way corresponds to setting the variable to True, the other to setting the variable to False. Thus, a stabber traversing the True or the False path of a gadget stabs at least 6 segments more than any other stabber.

We fit the variable gadget into a circular arc by putting the two non-middle stack of points on the arc. The middle stack of points and the stack of segments lie inside the circle. Each variable gadget is fit into an arc of $1/(4n)$ of a unit circle. The variable gadgets are placed next to each other on an arc of one quarter of a unit circle. We call this arc the *variable arc*.

**Clause gadgets.** Each clause gadget consists of 4 segments and 8 points. The points are partitioned into two sets of equal size. The segments and the points of a set are stacked on top of each other. The stacks are arranged in the same way as in the hardness proof for bends; each clause gadget is fit into an arc by putting the stacks of points on the arc and the stack of segments lies inside the circle. Each clause gadget is fit into an arc of $1/(4m)$ of a unit circle. The clause gadgets are placed next to each other on an arc of one quarter of a unit circle. We call this the *clause arc*. The clause arc is placed next to the variable arc. Both arcs together occupy one half of a unit circle.

**Connector segments.** We now place $3m$ connector segments, connecting a variable gadget to a clause gadget whenever the variable appears in the clause. The placement of the endpoints of the connector segments within the variable gadgets is as follows: Suppose the variable appears unnegated in the clause. Then the connector segment touches the True path of the gadget and it does not intersect the False path. If the variable appears negated in the clause the segment touches the False path and not the True path. The placement of the endpoints of the connector segments within the clause gadgets is as follows: We have to ensure that a convex stabber can stab any two of these connector segments in each clause gadget, but not all three. Fig. 7 shows the placement of the segments endpoints and the three possible ways to traverse the gadget. It can be easily checked that any two of these segments are stabbed by one of the three possible ways. All other possibilities to traverse the gadget stab less than two of the connector segments. Hence there is no way to stab all three connector segments within the clause gadget.

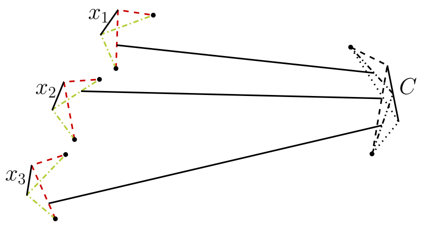


Figure 7: Example for a clause $C = \overline{x_1} \vee x_2 \vee \overline{x_3}$.

Observe that there exists a convex stabber that stabs all segments of the variable and clause gadgets and at least 2 out of the 3 connector segments of each clause gadget. Thus, there always exists a stabber that stabs at least $24n + 14m$ segments.

**Lemma 2** *There is a convex stabber stabbing $24n + 14m + k$ segments if and only if there is an assignment satisfying $k$ clauses.*

**Proof.** If there is an assignment that satisfies $k$ clauses, the stabber traverses the variable gadgets according to the assignment. In each satisfied clause at least one of the connectors is already stabbed, hence the stabber stabs the other two connector segments. Thus, it stabs 3 connector segments for each of the $k$ satisfied clauses and 2 connector segments for each of the $(m - k)$ not satisfied clauses and in total $24n + 12m + 3k + 2(m - k) = 24n + 14m + k$ segments.

On the other hand, if there is a stabber stabbing $24n + 14m + k$ segments, the stabber is forced to traverse the gadgets in *the right order* meaning it either takes the True or False path at each variable gadget and stabs at least two connector segments at each clause gadget. Any stabber traversing the gadgets in any other way stabs less than $24n + 14m$ segments. Thus, a stabber stabs already $24n + 14m$ segments by traversing the gadgets in the right order and the $k$ additional segments have to be stabbed in the variable gadgets. Hence, we set the variables according to the stabber traversing these gadgets and so the formula has $k$ satisfied clauses. □

**Theorem 3** *Let $\mathcal{L}$ be a set of line segments in the plane. It is APX-hard to compute a convex stabber that stabs the maximum number of segments of $\mathcal{L}$.*

**Proof.** We use a PTAS-reduction from MAX-E3SAT(5). Let $n$ be the number of variables and $m$ be the number of clauses. We reduce the problem to the convex stabber problem as explained before. Let $k$ be the maximum number of satisfied clauses. Since there always exists an assignment satisfying at least 7/8 of the clauses, we conclude that $k \geq 7/8m$. Assume there exists a polynomial-time algorithm for the maximum convex stabber problem that returns a solution that is at least $(1 - \epsilon)$ times the value of the optimal solution. Then we can approximate MAX-E3SAT(5) by subtracting $24n + 14m$ (note that $3m = 5n$):

$$\begin{aligned} & (1-\epsilon)(24n + 14m + k) - 24n - 14m \\ = \;& k - \epsilon k - 142/5m\epsilon \leq k - \epsilon k - 1136/35\epsilon k \\ \leq \;& (1 - \epsilon')k \end{aligned}$$

□

Since there exists a PTAS for *planar* MAX-SAT [4], we cannot use these ideas to show APX-hardness for a set of disjoint bends.

## Acknowledgment

I thank Esther M. Arkin and Joseph S. B. Mitchell for pointing out the convex stabber problem for disjoint objects and Wolfgang Mulzer for mentioning the optimization problem.

## References

[1] E. M. Arkin, C. Dieckmann, C. Knauer, J. S. B. Mitchell, V. Polishchuk, L. Schlipf, and S. Yang. Convex transversals. In *WADS*, pages 49–60, 2011.

[2] M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *COCOON*, pages 216–225, 2010.

[3] U. Feige. A threshold of ln $n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[4] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *STOC*, pages 329–337, 1996.

[5] A. Tamir. Problem 4-2 (New York University, Dept. of Statistics and Operations Research), Problems Presented at the Fourth NYU Computational Geometry Day (3/13/87).

# Unions of Onions

Maarten Löffler[*] Wolfgang Mulzer[†]

## Abstract

Let $\mathcal{D}$ be a set of $n$ pairwise disjoint unit disks in the plane. We describe how to build a data structure for $\mathcal{D}$ so that for any point set $P$ containing exactly one point from each disk, we can quickly find the onion decomposition (convex layers) of $P$.

Our data structure can be built in $O(n \log n)$ expected time and has linear size. Given $P$, we can find its onion decomposition in $O(n \log k)$ time, where $k$ is the number of layers. We also provide a lower bound showing that the running time must depend on $k$.

Our solution is based on a recursive space decomposition, combined with a fast algorithm to compute the union of two disjoint onion decompositions.

## 1 Introduction

Let $P$ be a planar $n$-point set. Take the convex hull of $P$ and remove it; repeat until $P$ becomes empty. This process is called *onion peeling*, and the resulting decomposition of $P$ into convex polygons is the *onion decomposition*, or *onion* for short, of $P$. It can be computed in $O(n \log n)$ time [4]. Onions provide a natural, more robust, generalization of the convex hull, and they have applications in pattern recognition, statistics, and planar halfspace range searching [5, 9].

Recently, a new paradigm has emerged for modeling data imprecision. Suppose we need to compute some interesting property of a planar point set. Suppose further that we have some advance knowledge about the possible locations of the points, e.g., from an imprecise sensor measurement. We would like to preprocess this information, so that once the precise inputs are available, we can obtain our structure faster. Many problems have been considered in this model, e.g., point set triangulation, Voronoi diagrams, and convex hulls [2, 6–8, 11, 12]. We will study the complexity of computing onions in this framework.

### 1.1 Results

We begin by showing that the union of two disjoint onions can be computed in $O(n + k^2 \log n)$ time, where

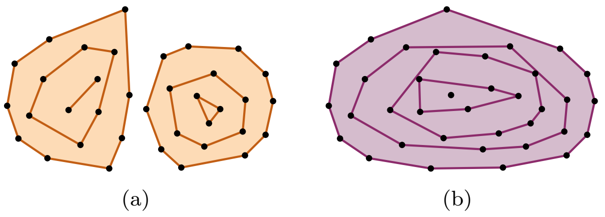


Figure 1: (a) Two disjoint onions. (b) Their union.

$k$ is the number of layers in the resulting onion.

We apply this algorithm to obtain an efficient solution to the onion preprocessing problem mentioned in the introduction. Given $n$ pairwise disjoint unit disks that model an imprecise point set, we build a data structure of size $O(n)$ such that the onion decomposition of an instance can be retrieved in $O(n \log k)$ time, where $k$ is the number of layers in the resulting onion. The expected preprocessing time is $O(n \log n)$.

We also show that without paramerising by $k$, it is not possible to speed up the computation of the onion decomposition: in the worst case, any algorithm can be forced to take $\Omega(n \log n)$ time on some instances.

## 2 Preliminaries and Definitions

Let $P$ be a set of $n$ points in $\mathbb{R}^2$. The *onion decomposition*, or *onion*, of $P$, is the sequence 🧅$(P)$ of pairwise disjoint convex polygons with vertices from $P$, constructed recursively as follows: if $P \neq \emptyset$, we set 🧅$(P) := \{\mathrm{ch}(P)\} \cup$ 🧅$(P \setminus \mathrm{ch}(P))$, where $\mathrm{ch}(P)$ is the convex hull of $P$; if $P = \emptyset$, then 🧅$(P) := \emptyset$ [4]. An element of 🧅$(P)$ is called a *layer* of $P$.

Let $\mathcal{D}$ be a set of disjoint unit disks in $\mathbb{R}^2$. We say a point set $P$ is a *sample* from $\mathcal{D}$ if every disk in $\mathcal{D}$ contains exactly one point from $P$. We write $\log$ for the logarithm with base $2$.

## 3 The Algorithm

In the following sections, we will describe the individual pieces required for our result.

### 3.1 Unions of Onions

Suppose we have two point sets $P$ and $Q$, together with their onions. We show how to find 🧅$(P \cup Q)$

[*]Universiteit Utrecht, `m.loffler@uu.nl`. Funded by Netherlands Organisation for Scientific Research (NWO) grant 639.021.123.

[†]Freie Universität Berlin, `mulzer@inf.fu-berlin.de`. Supported in part by DFG project MU/3501/1.

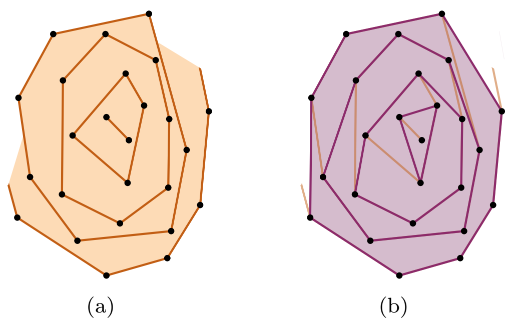


Figure 2: (a) A half-eaten onion; (b) the restored onion.

quickly, given that 🧅$(P)$ and 🧅$(Q)$ are disjoint. Deleting points can only decrease the number of layers, so:

**Observation 1** *Let $P, Q \subseteq \mathbb{R}^2$. Then* 🧅$(P)$ *and* 🧅$(Q)$ *cannot have more layers than* 🧅$(P \cup Q)$. □

The following lemma constitutes the main ingredient of our onion-union algorithm. By a *convex chain*, we mean any connected subset of a convex closed curve.

**Lemma 1** *Let $A$ and $B$ be two non-crossing convex chains. We can find $\text{ch}(A \cup B)$ in $O(\log n)$ time.*

**Proof.** Since $A$ and $B$ do not cross, the pieces of $A$ and $B$ that appear on $\text{ch}(A \cup B)$ are both connected: otherwise, $\text{ch}(A \cup B)$ would contain four points belonging to $A$, $B$, $A$, and $B$, in that order. However, the points on $A$ must be connected inside $\text{ch}(A \cup B)$; as do the points on $B$. Thus, the chains $A$ and $B$ cross, which is impossible. Since $A$ and $B$ are convex chains, we can compute $\text{ch}(A), \text{ch}(B)$ in $O(\log n)$ time. Furthermore, since $A$ and $B$ are disjoint, we can also, in $O(\log n)$ time, make sure that $\text{ch}(A) \cap \text{ch}(B) = \emptyset$, by removing parts from $A$ or $B$, if necessary. Now we can find the bitangents of $\text{ch}(A)$ and $\text{ch}(B)$ in logarithmic time [10]. □

**Lemma 2** *Suppose* 🧅$(P)$ *has $k$ layers. Let $A$ be the outer layer of* 🧅$(P)$*, and $p, q$ be two vertices of $A$. Let $A_1$ be the points on $A$ between $p$ and $q$, going counterclockwise. We can find* 🧅$(P \setminus A_1)$ *in $O(k \log n)$ time.*

**Proof.** The points $p$ and $q$ partition $A$ into two pieces, $A_1$ and $A_2$. Let $B$ be the second layer of 🧅$(P)$. The outer layer of 🧅$(P \setminus A_1)$ is the convex hull of $P \setminus A_1$, i.e., the convex hull of $A_2$ and $B$. By Lemma 1, we can find it in $O(\log n)$ time. Let $p', q' \in P$ be the points on $B$ where the outer layer of 🧅$(P \setminus A_1)$ connects. We remove the part between $p'$ and $q'$from $B$, and use recursion to compute the remaining layers of 🧅$(P \setminus A_1)$ in $O((k-1)\log n)$ time; see Figure 2. □

We conclude with the main theorem of this section:

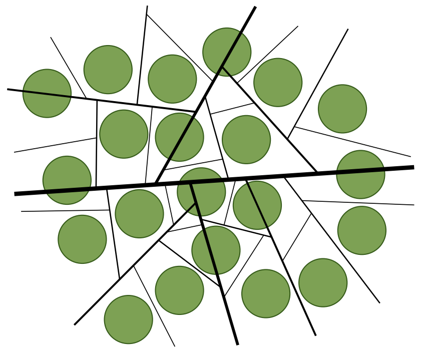


Figure 3: A space decomposition tree for 21 unit disks.

**Theorem 3** *Let $P$ and $Q$ be two planar point sets of total size $n$. Suppose that* 🧅$(P)$ *and* 🧅$(Q)$ *are disjoint. We can find the onion* 🧅$(P \cup Q)$ *in $O(k^2 \log n)$ time, where $k$ is the resulting number of layers.*

**Proof.** By Observation 1, 🧅$(P)$ and 🧅$(Q)$ each have at most $k$ layers. We use Lemma 1 to find $\text{ch}(P \cup Q)$ in $O(\log n)$ time. By Lemma 2, the remainders of 🧅$(P)$ and 🧅$(Q)$ can be restored to proper onions in $O(k \log n)$ time. The result follows by induction. □

## 3.2 Space Decomposition Trees

We now describe how to preprocess the disks in $\mathcal{D}$ for fast divide-and-conquer. A *space decomposition tree* $T$ is a rooted binary tree where each node $v$ is associated with a planar region $R_v$. The root corresponds to all of $\mathbb{R}^2$; for each leaf $v$ of $T$, the region $R_v$ intersects at most one disk in $\mathcal{D}$. Furthermore, each inner node $v$ in $T$ is associated with a directed line $\ell_v$, so that if $u$ is the left child and $w$ the right child of $v$, then $R_u := R_v \cap \ell_v^+$ and $R_w := R_v \cap \ell_v^-$. Here, $\ell_v^+$ is the halfplane to the left of $\ell_v$ and $\ell_v^-$ the halfplane to the right of $\ell_v$.

We would like to construct a space decomposition tree for $\mathcal{D}$ whose height is as low as possible. For this, we use the following lemma, which is a constructive version of a result by Alon *et al.* [1, Theorem 1.2].

**Lemma 4** *There exists a constant $c \geq 0$, so that for any set $\mathcal{D}$ of $m$ congruent nonoverlapping disks in the plane, there is a line $\ell$ with at least $m/2 - c\sqrt{m \log m}$ disks completely to each side of it. We can find $\ell$ in $O(m)$ expected time.*

**Proof.** Our proof closely follows Alon *et al.* [1, Section 2]. Set $r := \lfloor \sqrt{m/\log m} \rfloor$. Pick a random integer $z$ between $1$ and $r/2$. Find a line $\ell$ whose angle with the $x$-axis is $(z/r)\pi$ and that has $\lfloor m/2 \rfloor$ centers of disks in $\mathcal{D}$ on each side. Given $z$, we can find $\ell$ in $O(m)$ time by a median computation. The proof by Alon *et al.* shows that with probability at least $1/2$ over the choice of $z$, the line $\ell$ intersects at most

$c\sqrt{m \log m}$ disks in $\mathcal{D}$, for some constant $c \geq 0$. Thus, we need two tries in expectation to find a good line $\ell$. The expected total running time is $O(m)$. □

To obtain our space decomposition tree $T$, we apply Lemma 4 recursively until the region for each node intersects at most one disk in $\mathcal{D}$. The next lemma helps us analyze the complexity of $T$.

**Lemma 5** *Let $T$ be a space decomposition tree obtained by recursive application of Lemma 4. For $k = 0, \dots, \log n$, let $d_k$ be the maximum number of disks that intersect the region for a node in $T$ of depth $k$. Then $\log d_k \leq \log n - k + O(1)$. Furthermore, the tree $T$ has $O(n)$ nodes and height $\log n + O(1)$.*

**Proof.** We have $d_0 = n$. Furthermore, by Lemma 4,

$$d_k/2 \leq d_{k+1} \leq d_k/2 + c\sqrt{d_k \log d_k}.$$

Define

$$\beta_k := \sum_{i=0}^{k-1} \left(\frac{2^i}{n}\right)^{1/3}.$$

Note that $\beta_k \leq 4$, for $k = 0, \dots, \log n$. We assume that $n \geq 2^{12}(3c+2)^9$, and we prove by induction that $\log d_k \leq \log n - k + \beta_k$, for $k = 0, \dots, \log n - 12 - 9\log(3c+2)$. For $k = 0$, this is clear, since $\beta_0 = 0$. Now note that for every $k \geq 0$, we have $d_k \leq n$ and $d_k \geq n/2^k$. Thus,

$$\begin{aligned} &\log d_{k+1} \leq \log(d_k/2 + c\sqrt{d_k \log d_k}) \\ &= \log(d_k/2) + \log\left(1 + 2c\sqrt{\frac{\log d_k}{d_k}}\right) \\ &\leq \log d_k - 1 + \log\left(1 + 2c\sqrt{\frac{2^k \log d_k}{n}}\right) \\ &\leq \log n - (k+1) + \beta_k + \frac{2c}{\ln 2}\sqrt{\frac{2^k(\log n - k + 4)}{n}} \\ &= \log n - (k+1) + \beta_{k+1}, \end{aligned}$$

since for $k \leq \log n - 12 - 9\log(3c+2)$, we have

$$\frac{2c}{\ln 2}\sqrt{\frac{2^k(\log n - k + 4)}{n}} \leq \left(\frac{2^k}{n}\right)^{1/3}.$$

It follows that after $k^* := \log n - 12 - 9\log(3c+2)$ iterations, there are at most $2^{12+9\log(3c+2)+\beta_{k^*}} = O(1)$ disks per region left. Hence, $T$ has height $\log n + O(1)$ and $O(n)$ nodes. □

By Lemma 5, $T$ induces a planar subdivision $G_T$ with $O(n)$ faces. We add a large enough bounding box to $G_T$ and triangulate the resulting graph. Since $G_T$ is planar, the triangulation has complexity $O(n)$ and can be computed in the same time (no need for heavy machinery—all faces of $G_T$ are convex). With each disk in $\mathcal{D}$, we store the list of triangles that intersect it (recall that each triangle intersects at most one disk). This again takes $O(n)$ time and space. We conclude with the main theorem of this section:

**Theorem 6** *Let $\mathcal{D}$ be a set of $n$ disjoint unit disks in $\mathbb{R}^2$. In $O(n \log n)$ expected time, we can construct a space partition tree $T$ for $\mathcal{D}$ with $O(n)$ nodes and height $\log n + O(1)$. Furthermore, for each disk $D \in \mathcal{D}$, we have a list of triangles $T_D$ that cover the leaf regions of $T$ that intersect $D$.* □

## 3.3 Processing a Precise Input

Let $P$ be a sample from $\mathcal{D}$. Suppose first that we know $k$, the number of layers in $◎(P)$. For each input point $p_i$, let $D_i \in \mathcal{D}$ be the corresponding disk. We check all triangles in $T_{D_i}$, until we find the one that contains $p_i$. Since there are $O(n)$ triangles, this takes $O(n)$ time. Afterwards, we know for each point in $P$ the leaf of $T$ that contains it.

The *upper tree* $T_u$ of $T$ consists of all nodes with depth at most $\log n - 2\log k$. Each leaf of $T_u$ is the root of a subtree of $T$ of height at most $2\log k + O(1)$. Hence it corresponds to a subset of $P$ with $O(k^2)$ points. For each such subset, we use Chazelle's algorithm [4] to find its onion decomposition in $O(k^2 \log k)$ time. This takes $O(n \log k)$ total time. Now, in order to obtain $◎(P)$, we perform a postorder traversal of $T_u$, using Theorem 3 in each node to unite the onions of its children.

For a node of depth $i$, this takes time $O(k^2 \log d_i) = O(k^2(\log n - i + 1))$, by Lemma 5. Thus, the total work is proportional to

$$\begin{aligned} &\sum_{i=0}^{\log n - 2\log k} 2^i k^2(\log n - i + 1) \\ &= k^2 \frac{n}{k^2} \sum_{i=0}^{\log \frac{n}{k^2}} \frac{2\log k + i + 1}{2^i} \\ &= O(n \log k). \end{aligned}$$

So far, we have assumed that $k$ is given. Using standard exponential search, this requirement can be removed. More precisely, let $k_i = 2^{2^i}$, for $i = 1, \dots, \log\log n$. Run the algorithm for $k_0, k_1, \dots$. If the algorithm succeeds, report the result. If not, abort as soon as it turns out that an intermediate onion has more than $k_i$ layers and try $k_{i+1}$. The total time is

$$\sum_{i=0}^{\log\log k} O(n2^i) = O(n \log k),$$

as desired. This finally proves our main result.

**Theorem 7** *Let $\mathcal{D}$ be a set of $n$ disjoint unit disks in $\mathbb{R}^2$. We can build a data structure that stores $\mathcal{D}$, of size $O(n)$, in $O(n \log n)$ expected time, such that given a sample $P$ of $\mathcal{D}$, we can compute $◎(P)$ in $O(n \log k)$ time, where $k$ is the number of layers in $◎(P)$.* □

**Remark.** Using the same approach, without the exponential search, we can also compute the outermost

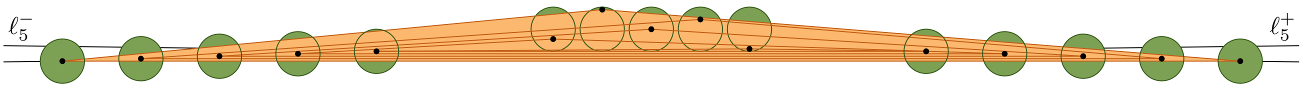


Figure 4: The lower bound construction consists of $n/3$ unit disks centered on a horizontal line ($5$ in the figure), and two groups of $n/3$ points sufficiently far to the left and to the right of the disks. Distances not to scale.

$k$ layers of an onion with arbitrarily many layers in $O(n \log k)$ time, for any $k$. In order to achieve this, we simply abort the union algorithm whenever $k$ layers have been found, and note that the points in $P$ not on the outermost $k$ layers of $◎(P)$ will never be part of the outermost $k$ layers of $◎(Q)$ for any $Q \supset P$.

## 4 Lower Bounds

We now show that the query time must depend on $k$ (i.e. for onions with many layers, we cannot hope to speed up the computation).

Let $n$ be a multiple of $3$, and consider the lines

$$\ell_n^- : y = -1/2 - 6/n - x/n^2;$$
$$\ell_n^+ : y = -1/2 - 6/n + x/n^2.$$

Let $\mathcal{D}_n$ consist of $n/3$ disks centered on the $x$-axis at $x$-coordinates between $-n/6$ and $n/6$; a group of $n/3$ disks centered on $\ell_n^-$ at $x$-coordinates between $n^2$ and $n^2+n/3$; and a symmetric group of $n/3$ disks centered on $\ell_n^+$ at $x$-coordinates between $-n^2 - n/3$ and $-n^2$. Figure 4 shows $\mathcal{D}_{15}$.

**Lemma 8** *Let $\pi$ be a permutation on $n/3$ elements. There is a sample $P$ of $\mathcal{D}_n$ such that $p_i$ (the point for the $i$th disk from the left in the main group) lies on layer $\pi(i)$ of $◎(P)$.*

**Proof.** Take $P$ as the $n/3$ centers of the disks in $\mathcal{D}$ on $\ell_n^-$, the $n/3$ centers of the disks in $\mathcal{D}$ on $\ell_n^+$, and for each disk $D_i \in \mathcal{D}$ on the $x$-axis the point $p_i = (i - n/6, \pi(i) \cdot 3/n - 1/2)$. By construction, the outermost layer of $◎(P)$ contains at least the leftmost point on $\ell_n^+$, the rightmost point on $\ell_n^-$, and the highest point (with $y$-coordinate $1/2$). However, it does not contain any more points: the line segments connecting these three points have slope at most $2/n^2$. The second highest point lies $3/n$ lower, and at most $n/3$ further to the left or the right. The lemma follows by induction. ◻

There are $(n/3)! = 2^{\Theta(n \log n)}$ permutations $\pi$; so any corresponding decision tree has height $\Omega(n \log n)$.

**Theorem 9** *For any $n$, there is a set $\mathcal{D}$ of $n$ disjoint unit disks in $\mathbb{R}^2$, such that any decision-based algorithm to compute $◎(P)$ for $P$ a sample of $\mathcal{D}$, based only on prior knowledge of $\mathcal{D}$, takes $\Omega(n \log n)$ time in the worst case.*

We expect that our lower bound can be strengthened to $\Omega(n \log k)$ and that it also applies to randomized algorithms. Details will follow in the full version.

## 5 Conclusion and Further Work

It would be interesting how much the parameter $k$ can vary for a set of imprecise bounds and how to estimate $k$ efficiently. Further work includes considering more general regions, such as overlapping disks, disks of different sizes, or fat regions. Furthermore, three-dimensional onions are not well understood. The best general algorithm needs $O(n \log^6 n)$ expected time [3], giving more room for improvement in our setting.

## References

[1] N. Alon, M. Katchalski, and W. Pulleyblank. Cutting disjoint disks by straight lines. *DCG*, 4:239–243, 1989.

[2] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for Delaunay triangulation: simplified and extended. *Algorithmica*, 61(3):675–693, 2011.

[3] T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):Art. 16, 15 pp., 2010.

[4] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, 31(4):509–517, 1985.

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

[6] O. Devillers. Delaunay triangulation of imprecise points: preprocess and actually get a fast query time. *J. Comput. Geom.*, 2(1):30–45, 2011.

[7] E. Ezra and W. Mulzer. Convex hull of points lying on lines in $o(n \log n)$ time after preprocessing. *Comput. Geom.*, 46(4):417–434, 2013.

[8] M. Held and J. S. B. Mitchell. Triangulating input-constrained planar point sets. *IPL*, 109:54–56, 2008.

[9] P. J. Huber. Robust statistics: A review. *Ann. Math. Statist.*, 43:1041–1067, 1972.

[10] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *Proc. 4th WADS*, pages 183–193, 1995.

[11] M. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM J. Comput.*, 39(7):2990–3000, 2010.

[12] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom.*, 43(3):234–242, 2010.

# Finding a largest empty convex subset in space is W[1]-hard

Panos Giannopoulos*[†] Christian Knauer*

**Abstract**

We consider the following problem: Given a point set in space find a largest subset that is in convex position and whose convex hull is empty. We show that the (decision version of the) problem is W[1]-hard.

## 1 Introduction

**Problem definition.** Let $P$ be a set of $n$ points in $\mathbb{R}^3$ and $k \in \mathbb{N}$. In the LARGEST-EMPTY-CONVEX-SUBSET problem we want to decide whether there is a set $Q \subset P$ of $k$ points in convex position whose convex hull does not contain any other point of $P$.

### 1.1 Results

We show that LARGEST-EMPTY-CONVEX-SUBSET is W[1]-hard with respect to the solution size $k$, under the extra condition that the solution set is *strictly* convex, i.e., the interior of the convex hull of any of its subsets is empty. This means that (under standard complexity-theoretic assumptions) the problem is not fixed-parameter tractable with respect to $k$, i.e., it does not admit an $O(f(k) \cdot n^c)$-time algorithm for any computable function $f$ and any constant $c$. See [4] for basic notions of parameterized complexity theory.

### 1.2 Related work

LARGEST-EMPTY-CONVEX-SUBSET has been shown to be NP-hard in last year's EuroCG [6]. (In that paper, NP-hardness has been shown also for the more general version where the emptiness condition is dropped.) Several interesting questions were also raised such as whether the problem is fixed-parameter tractable with respect to the solution size and whether it admits a polynomial $o((\log n)/n)$-approximation algorithm. Here, we give a negative answer to the first question. Note that in the plane, the problem is solvable in polynomial time; see, for example, [1], [2].

From a combinatorial point of view, there is a long history of results starting with the famous Erdös-Szekeres theorem [3], which states that for every $k$ there is a number $n_k$ such that every planar set of $n_k$ points in general position contains $k$ points in convex position. Horton [5] showed that this is not true when the emptiness condition is imposed: There are arbitrarily large sets that do not contain empty 7-gons. Results of this type exist also for higher dimensions; see [7].

*Institut für Informatik, Universität Bayreuth, Universitätsstraße, 30, D-95447 Bayreuth, Germany, {christian.knauer, panos.giannopoulos}@uni-bayreuth.de

[†]Research supported by the German Science Foundation (DFG) under grant Kn 591/3-1.

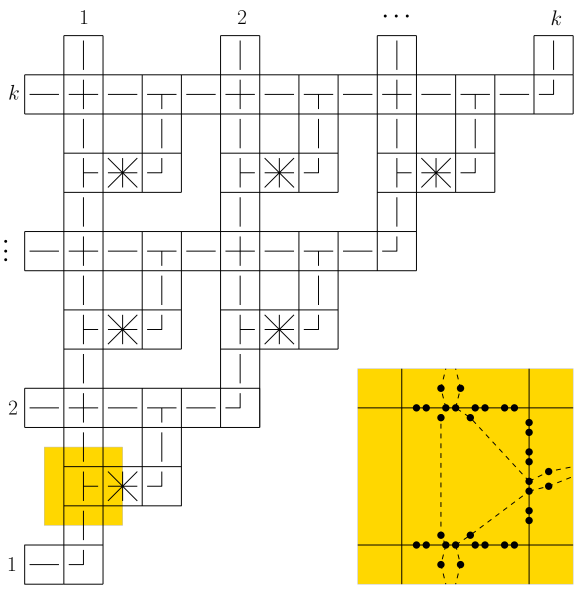


Figure 1: High level schematic of the construction. The zoomed-in area shows points shared among gadgets on their common boundaries and some pairs of points inside each gadget that take part in a choice of an empty convex set, (partially) marked with dashed segments.

## 2 Reduction

We show that LARGEST-EMPTY-CONVEX-SUBSET is W[1]-hard by an *fpt*-reduction from the W[1]-hard $k$-CLIQUE problem [4]: Given a graph $G([n], E)$ and $k \in \mathbb{N}$, decide whether $G$ contains a clique of size $k$.

### 2.1 High level description

We begin with a high-level description of the construction, see Fig. 1. Initially, the construction will lie on the plane; later on, it will be lifted to the elliptic paraboloid with a (more or less) standard transform. The construction is organized as the upper diagonal part of a grid with $k$ rows and $k$ columns. The $i$th row and column represent a choice for the $i$th vertex of a clique in $G$ and are made of $4i-2$ and $4(k-i+1)-2$ gadgets respectively. There are $n$ choices and each choice is represented by a collection of empty convex subsets of points – one subset with a constant number of points from each gadget.

Each gadget consists of $\Theta(n)$ points within a rectangular region, which are organized in sets (of pairs) of collinear points. There is a constant number of such sets and, since we are looking for strictly convex subsets, only one pair of consecutive points per set can be chosen at any time. Certain choices are rendered invalid by additional points. Neighboring gadgets share the points on their common rectangle edge, see the zoomed-in area in Fig 1. Through these common points, the choice of subsets is made consistent among the gadgets. In particular, the choice in the $i$th row is made consistent with the choice in the $i$th column via the 'diagonal', ◰ gadget in their intersection corner; consistency here means that they both correspond to the same choice of a vertex of $G$. On the other hand, in the intersection of the $i$th column with the $j$th row, for every $j \geq i+1$, there is a 'cross', ⊞ gadget, which ensures that the choice in the column is propagated independently of the choice in the row and vice versa. Finally, the $j$th column is 'connected' to the $i$th row, for every $j+1 \leq i \leq k$, by three additional gadgets. One of them, the 'star', ⊛ gadget, encodes graph $G$, i.e., it allows only for combinations of choices (in the column and the row) that are consistent with the edges in $G$.

Locally, every valid subset from a gadget consists of points that are in strictly convex position and whose convex hull is empty. By lifting the whole construction to the paraboloid appropriately, we make sure that this property is true globally, i.e., for any set constructed from the local choices in a consistant manner.

In total, the construction consists of a set $P$ of $\Theta(k^2n^2)$ points in $\mathbb{R}^3$, such that there exists a set $Q \subset P$ with the desired property and $|Q| = f(k)$, for some function $f(k) \in \Theta(k^2)$, if and only if $G$ has a clique of size $k$.

## 2.2 Gadgets

There are five different types of gadgets, and each type has a specific function, which is explained below.

**⊟ gadget.** This gadget propagates a choice of pairs of points horizontally, see Fig. 2. It has $n$ pairs of points $L_i$, $R_i$ on the left and right side of its rectangle respectively, which correspond to the vertices of $G$; pairs corresponding to the same vertex are aligned horizontally. It also has $n(n-1)+2$ points inside the rectangle on a vertical line $\ell$ as follows. For every two pairs $L_i$ and $R_j$, with $i \neq j$, a point is placed such that it is inside the parallelogram $L_iR_j$ formed by the pairs but outside the parallelogram formed by

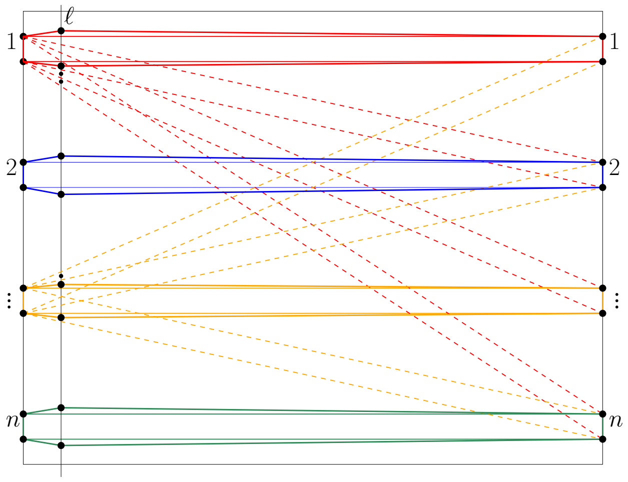


Figure 2: The ⊟ gadget. Dashed parallelograms represent choices cancelled by points on $\ell$. Parallelograms in full are not cancelled. There are $n$ choices of convex empty 6-gons.

any other two pairs. Effectively, this point cancels a choice of pairs that correspond to different vertices of $G$. One point is placed above $L_1R_1$ on $\ell$ and close to its boundary; similarly one point is placed below $L_nR_n$. Due to the strict convexity condition, at most one pair per rectangle side and at most one pair on $\ell$ can be chosen. Thus, there are $n$ maximum size empty convex subsets. Each subset contains six points and is formed by three pairs: $L_i$, $R_i$, for some $i$, and the pair of points on $\ell$ that are closest to the parallelogram $L_iR_i$; this latter pair is formed by the point that cancels $L_iR_{i-1}$ and the point that cancels $L_iR_{i+1}$. The ◫ gadget is just a 90°-rotated copy of the ⊟ gadget.

**◰ gadget.** This gadget has basically the same structure as the ⊟ gadget but propagates information diagonally. For completeness, it is shown in Fig 3.

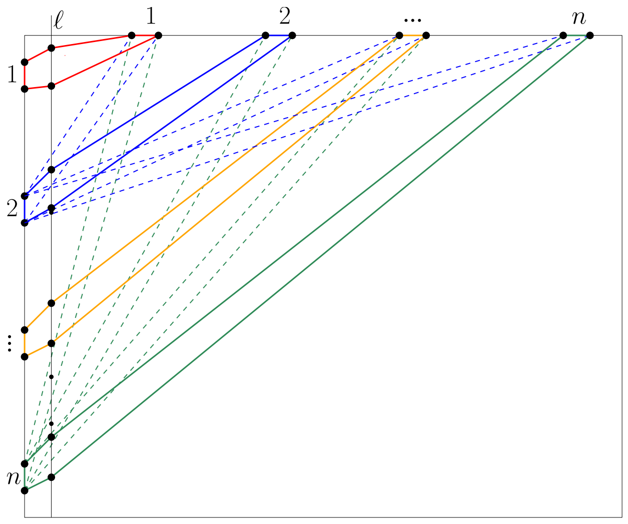


Figure 3: The ◰ gadget.

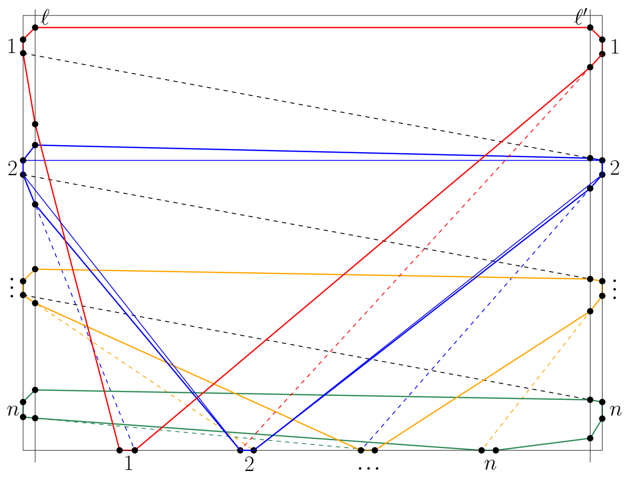


Figure 4: The ⍊ gadget. There are $n$ choices of convex empty 10-gons. An empty convex 6-gon, as described in the text, is shown only for $i = 2$.

**⍊ gadget.** This gadget propagates information both horizontally and diagonally, see Fig. 4. It has $n$ pairs of points $L_i$, $R_i$, and $B_i$, on the left, right, and bottom side of its rectangle respectively. As before, an $i$th pair corresponds to the $i$th vertex of $G$. There will be only $n$ valid choices and three pairs per choice, namely, the $i$th pair from each side. This is enforced by the strict convexity condition and by placing, for every $i$, four additional points on two vertical lines $\ell$ and $\ell'$ inside the rectangle. These points are placed *outside* the convex 6-gon $L_iR_iB_i$ that is formed by the points in the corresponding pairs. (The points are also outside every other 6-gon for $j \neq i$.) See the example for $i = 2$ in Fig. 4. More specifically, looking at the gadget from top to bottom and from left to right, a point is placed on the intersection of $\ell$ and the line through the second point of $L_i$ and the second point of $B_{i-1}$; for the case of $i = 1$, the point is placed just below the 6-gon. A second point is placed on $\ell$ just above the 6-gon. At the right side, two points are placed on $\ell'$ as follows. One point is placed on the intersection of $\ell'$ and the line through the second point of $L_i$ and the first point of $R_{i+1}$; for $i = 1$, the point is placed just above the 6-gon. A second point is placed on the intersection of $\ell'$ and the line through the second point of $R_i$ and the first point of $B_{i+1}$; for $i = n$, the point is placed just below the 6-gon.

There are $n$ maximum size empty convex subsets with 10 points each. A subset is formed by the points in the pairs $L_i$, $R_i$, $B_i$, and the four points on $\ell$ and $\ell'$ that are closest to the corresponding 6-gon.

**⊞ gadget.** This gadget consists of eight subgadgets, which are very similar to the ones we have already described above. See Fig. 5. It can be thought of as having two inputs (at the upper and lower left corner) and two outputs (at the upper and lower right corner).

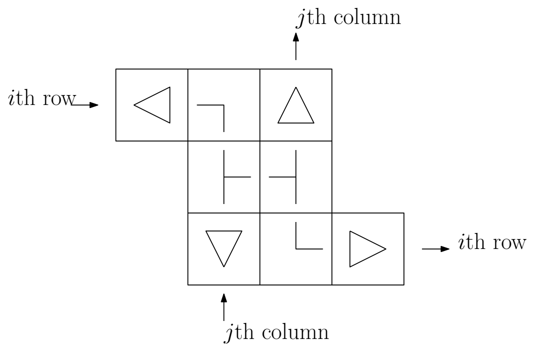


Figure 5: The ⊞ gadget: a high level schematic.

It propagates the inputs (choices) independently from each other, one horizontally and one vertically. The input subgadgets ◁ and ▽ are respectively connected (through their left and bottom rectangle sides) to the ⊟ and ◫ gadgets of the $i$th row and $j$th column of the global construction (Fig. 1). The output subgadgets ▷ and △ are similarly connected to a ⊟ and ◫ gadget. Note that the subgadgets ⊢ and ⊣ in the middle of the ⊞ gadget (Fig. 5) as well as the subgadgets ┐ and └ are *not* connected directly to any row or column of the global construction. Roughly speaking, the ⊞ gadget has the following function: it multiplexes the two inputs, then it mirrors them (vertically and horizontally), and then demultiplexes them.

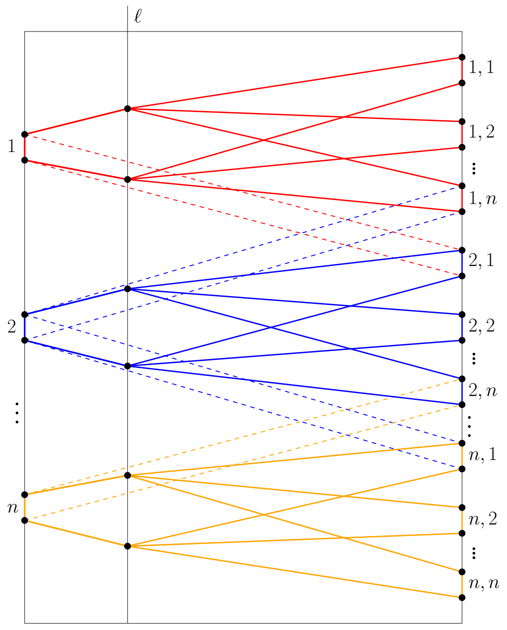


Figure 6: The ◁ subgadget.

The ◁ subgadget is shown in Fig. 6. It is similar to the previously described ⊟ gadget in Fig. 2. It has again $n$ pairs of points $L_i$ on the left side. The difference now is that there are $n^2$ pairs of points $R_{i,j}$, $1 \leq i, j \leq n$, on the right side of the gadget. The second index $j$ basically encodes the choice coming from the input subgadget ⍔ at the lower left corner, which is communicated through the ⊞ and ◱ subgadgets inbetween. Only the $n^2$ pairs $L_i$ and $R_{i,j}$ constitute valid choices. The rest are cancelled by additional points as usually. Together with pairs of canceling points (which are also chosen as before) there are exactly $n^2$ empty convex 6-gons, and these are of maximum size.

The input ⍔ subgadget propagates information vertically and is defined similarly to the ⍃ subgadget. It has $n$ pairs of points $B_j$ on the bottom side and $n^2$ pairs $T_{ij}$ on the top side. The difference now is that only the $n^2$ pairs $B_j$ and $T_{i,j}$ are valid.

The output subgadgets ⍄ and ⍍ are just mirrored images of their input counterparts. The ◱ and ◳ subgadgets are constructed in the same way as the ◰ gadget in Fig. 3 but have $n^2$ valid 6-gons, while the ⊞ and ⊞ subgadgets are constructed in the same way as the ⊞ gadget in Fig. 4 and have $n^2$ valid 10-gons.

**⧆ gadget.** This gadget encodes the edges of the input graph $G$. See Fig. 7. It is similar to gadget ⊟ (Fig. 2) and allows only combinations of pairs that correspond to edges of the graph: for every *non*-edge $ij$ of $G$, a point is placed inside the parallelogram $L_iR_j$.

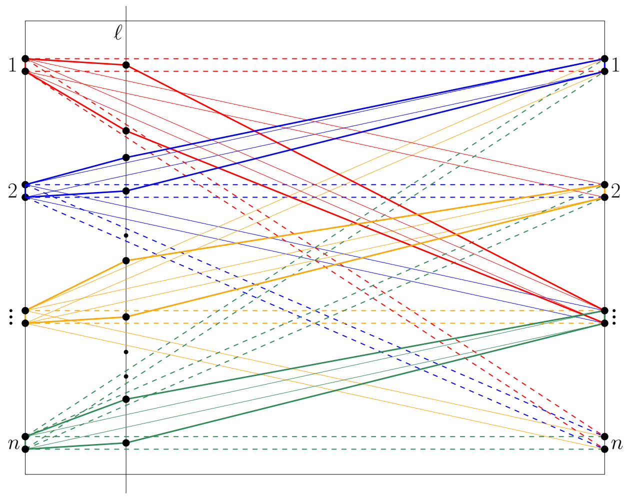


Figure 7: The ⧆ gadget. Several examples of cancelled choices (in dashed) and of empty convex 6-gons (in bold) are shown.

## 2.3 Lifting to $\mathbb{R}^3$

Every corner of a gadget rectangle is lifted to the paraboloid with the map $(x, y) \mapsto (x, y, x^2 + y^2)$. The images of the corners of each rectangle lie on one distinct plane (since the corners lie on a circle). The points in a gadget are projected orthogonally on the corresponding plane. This is an affine map and thus colinearity and convexity within a gadget is preserved. Each gadget now lies on a distinct facet (a parallelogram) of a convex polyhedron.

## 2.4 Correctness

The total number of (sub)gadgets of each type together with the size of a largest valid (i.e., empty and convex) subset in a gadget of the type is

| | |
|---|---|
| ⊟ | $k^2$, 6; |
| ◫ | $k(3k-1)/2$, 6; |
| ◰, ◱, ◳ | $k(3k-1)/2$, 6; |
| ⊞, ⊞, ⊞ | $2k(k-1)$, 10; |
| ⍃, ⍄, ⍍, ⍔ | $2k(k-1)$, 6; |
| ⧆ | $k(k-1)/2$, 6. |

A global valid subset is formed by locally choosing one valid subset from every gadget in a consistent manner. When a largest locally possible subset (as given above) can be chosen, the global subset has size $k(35k - 23)$. Such a global subset corresponds to a $k$-size clique of $G$.

For suppose there exists a global valid subset of size $k(35k - 23)$. Then, a largest locally possible valid subset must be chosen from every gadget. Consider such a choice of subsets and let $v_i$ be the vertex of $G$ corresponding to the choice from the leftmost gadget of the $i$th grid row. By construction, the subset corresponding to the same vertex $v_i$ must be chosen from every other gadget in this row as well as every gadget in the $i$th column. Consider the $j$th row, for some $j \neq i$. Through the ⧆ gadget that connects the $i$th column to the $j$th row, when $i + 1 \leq j$, or the $j$th column to the $i$th row, when $j < i$, the subset chosen from the $j$th row must correspond to a vertex $v_j$ such that $v_iv_j$ is an edge of $G$. Hence $\{v_1, \ldots v_k\}$ is a clique in $G$. The converse is obvious.

**Theorem 1** Largest-Empty-Convex-Subset *is W[1]-hard, under the condition of strict convexity.*

## References

[1] D. Avis and D. Rappaport. Computing the largest empty convex subset of a set of points. In *Proc. of SCG '85*, 1985. ACM.

[2] D. Dobkin, H. Edelsbrunner, and M. Overmars. Searching for empty convex polygons. *Algorithmica 5*, 1990.

[3] P. Erdös and G. Szekeres. A combinatorial problem in geometry. *Compositio Math. 2*, 1935.

[4] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006.

[5] J. D. Horton. Sets with no empty convex 7-gons. *C. Math. Bull. 26*, 1983.

[6] C. Knauer and D. Werner. Erdös-szekeres is NP-hard in 3 dimensions - and what now? In *Abstracts of 28th EuroCG*, pages 61–64, 2012.

[7] W. Morris and V. Soltan. The Erdös-Szekeres problem on points in convex position – a survey. *Bull. Amer. Math. Soc. 37*, 2000.

# The Degree of Convexity

Günter Rote[*]

## Abstract

We measure the degree of convexity of a planar region by the probability that two randomly chosen points see each other inside the polygon. We show that, for a polygonal region with $n$ edges, this measure can be evaluated in polynomial time as a sum of $O(n^9)$ closed-form expressions.

## 1 Introduction

A set $P$ is convex if, for every two points $u, w \in P$, the whole segment $uw$ belongs to $P$. If $P$ is not convex, this conclusion will not always be true, and we can get a quantity for the "degree" or "measure" of convexity if we take the probability with which it is fulfilled, for two points $u, w$ selected uniformly at random from $P$.

More formally, let $|P|$ denote the area of $P$. Then the *degree of convexity* $C(P)$ is defined as the normalized double area integral

$$C(P) := \frac{1}{|P|^2} \int_{u\in P} \int_{w\in P} [uw \subset P]\, dwdu, \qquad (1)$$

using the bracket notation for the characteristic function of a logical expression: $[uw \subset P]$ equals 1 if the condition $uw \subset P$ holds and 0 otherwise.

## 2 Related Work

Stern [3] has been the first to consider the measure (1), as a simple alternative to another measure he proposed, the so-called "polygonal entropy". He evaluated $C(P)$ by Monte Carlo estimation. Stern observed that $C(P)$ can equally be expressed as the normalized average visible area, i.e., the expected area of the visibility polygon of a random point, divided by $|P|$.

Various other definitions have been proposed to measure "convexity", primarily in the pattern recognition community, in addition to measuring "circularity", "squareness", "rectangularity" "elongation" etc.

In principle, one can take any quantity that is bounded by 1, for which equality holds (among compact sets) for convex sets only. Some very primitive measures that count the reflex angles of a polygon, (or sum them up) fulfill this requirement but they are not very sensitive to the shape of $P$.

Žunić and Rosin [4] mention, besides $C(P)$, the area of $P$ divided by the area of the convex hull. The complement of this is called the *deficit of convexity* in the textbook [2, p. 35] (p. 23 in the 2008 version). Instead of the area, one can also look at the perimeter.

Boxer [1] considered the *index of non-convexity*, the maximum distance of a pocket from the corresponding convex hull edge, and related measures.

## 3 Properties of the Degree of Convexity

Most of the following basic properties were already observed by Stern [3].

Clearly, $C(P)$ is between 0 and 1, and $C(P) = 1$ if $P$ is convex. For a compact set $P$ which is the closure of its interior, $C(P) = 1$ if and only if $P$ is convex.

$C(P)$ is invariant under affine transformations.

## 4 Partitioning the Region

Let $P$ be a polygonal region with $n$ boundary edges. Throughout the paper we will assume general position, to keep the discussion simple.

An *internal bitangent* of $P$ with tangency vertices $x$ and $y$ $(x \neq y)$ is a line segment $ab \subset P$ whose endpoints $a$ and $b$ lie in the interior of $P$ and which has the two distinct points $x$ and $y$ in common with the boundary of $P$, see Fig. 1a. Then $x$ and $y$ are necessarily reflex vertices.

We extend the rays from $x$ and $y$ towards the endpoints until they hit the boundary of $P$, see Fig. 1b–c. We use these extension rays to partition $P$, as in Fig. 1e. We don't include the segment $xy$.

We also extend the edges of every reflex vertex, in order to ensure that all cells of resulting partition $Z$ are convex as shown in Fig. 1f.

Finally, we will compute the integral in (1) separately for each pair of cells $A, B$ of $Z$, and add up the results

$$I(A, B) := \int_{u\in A} \int_{w\in B} [uw \subset P]\, dwdu \qquad (2)$$

Remark: If we would extend the segments between any two vertices, like in Fig. 1d, we would get the well-known *visibility cell decomposition*: a refinement of $Z$ that corresponds to the combinatorially different visibility polygons of all points in $P$.

[*]Freie Universität Berlin, Institut für Informatik, Takustraße 9, 14195 Berlin, Germany. rote@inf.fu-berlin.de.
Supported by the ESF programme EuroGIGA-VORONOI, Deutsche Forschungsgemeinschaft (DFG): RO 2338/5-1.

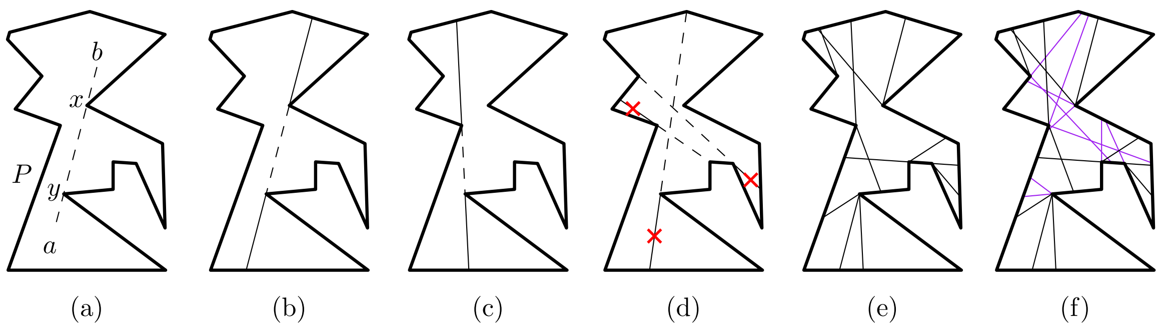


Figure 1: Partitioning $P$ by extensions of bitangents. (a) A bitangent. (b–c) Some extensions of bitangents, and (e) the arrangement of all extensions. (d) Not all extensions of visibility edges are used (only bitangents). (f) The final partition $Z$.

**Definition 1** *Consider to disjoint open convex regions $A, B \subset P$. Let $V_1, V_2, \ldots, V_k$ be a set of open segments (i.e., not containing their starting point), disjoint from $A$ and $B$ (see Fig. 2).*

- *We say that* visibility between $A$ and $B$ is determined by the segments $V_1, V_2, \ldots$ *if the following holds: two points $u \in A$ and $w \in B$ can see each other iff the segment $uw$ doesn't intersect any of the segments $V_1, V_2, \ldots$. We refer to these segments as* blocking segments.

- *We say that a set of blocking segments is* mutually exclusive *if no segment $uw$ for $u \in A$ and $w \in B$ can intersect more than one of the blocking segments.*

If we have mutually exclusive blocking segments, we can consider them independently from each other and evaluate the integral (2) as follows

$$I(A,B) = \int_{u\in A}\int_{w\in B}\big(1 - [uw \cap V_1 \neq \emptyset] - \\ - [uw \cap V_2 \neq \emptyset] - \cdots\big)\, dw du$$

$$= |A|\cdot|B| - \sum_{i=1}^{k}\int_{u\in A}\int_{w\in B}[uw \cap V_i \neq \emptyset]\, dw du$$

**Lemma 1** *Let $P$ be a polygonal region, possibly with holes. Consider two cells $A, B$ of the partition $Z$, considered as open sets. Then there are three possibilities.*

1. *All points of $A$ see all points of $B$.*
2. *No point from the interior of $A$ sees a point from the interior of $B$.*
3. *Visibility between $A$ and $B$ is determined by some mutually exclusive blocking segments $V_1, \ldots, V_k$.*

For the case of a simple polygon, we can have at most two blocking segments (one blocking from the left and one from the right, when looking from $A$ towards $B$). If $P$ has $h$ holes, there can be at most $h$ additional blocking segments.

**Proof.** Suppose that $A$ and $B$ are separated by a vertical line, as in Fig. 2, so that, for a segment $uw$ from $A$ to $B$, it makes sense to speak of "above $uw$" or "below $uw$".

If $u \in A$ and $w \in B$ move, the segment $uw$ will sometimes be contained in $P$ (we call it a *free segment* in this case), and sometimes it will intersect the boundary of $P$. (Otherwise, we are in Case 1 or 2 and we are done.) When a free segment hits the boundary of $P$, it will do so at a reflex vertex $r$ of $P$. We call such a vertex $r$ a *blocking corner*, and it is a *top blocking corner* or a *bottom blocking corner*, when the direction into which $uw$ can freely move is above $r$ or below $r$. Fig. 2 shows a top blocking corner $r$. We can rotate $uw$ around $r$ while maintaining $u \in A$ and $w \in B$. The extremes of this rotation are segments where $uw$ becomes tangent to $A$ or $B$. If, during this rotation, $uw$ would hit another obstacle vertex $r'$ before hitting the extreme directions we would have an extension ray for the bitangent $rr'$ that would cut through $A$ and $B$, a contradiction to the assumption that $A$ and $B$ are cells of $Z$. Thus, we can state: *No segment from $A$ to $B$ goes through two blocking corners.*

As a consequence, the blocking corners can be linearly ordered from bottom to top: Choose an arbitrary free segment through each blocking corner. For two corners $r$ and $r'$ with respective segments $uw$ and $u'w'$ through them, $r$ lies above $u'w'$ iff $r'$ lies below $uw$. Thus, we get a (consistent) linear order (cf. Fig. 3a). In this order, top corners and bottom corners must alternate We create the blocking segments $V_i$ by matching each top blocking corner with the bottom blocking corner immediately below it.

If a bottom corner $r$ at the very top remains unmatched, we attach a sufficiently long upward vertical segment (or effectively, an infinite ray) to $r$, as shown for segment $V_1$ in Fig. 2. Similarly, at the bottom,

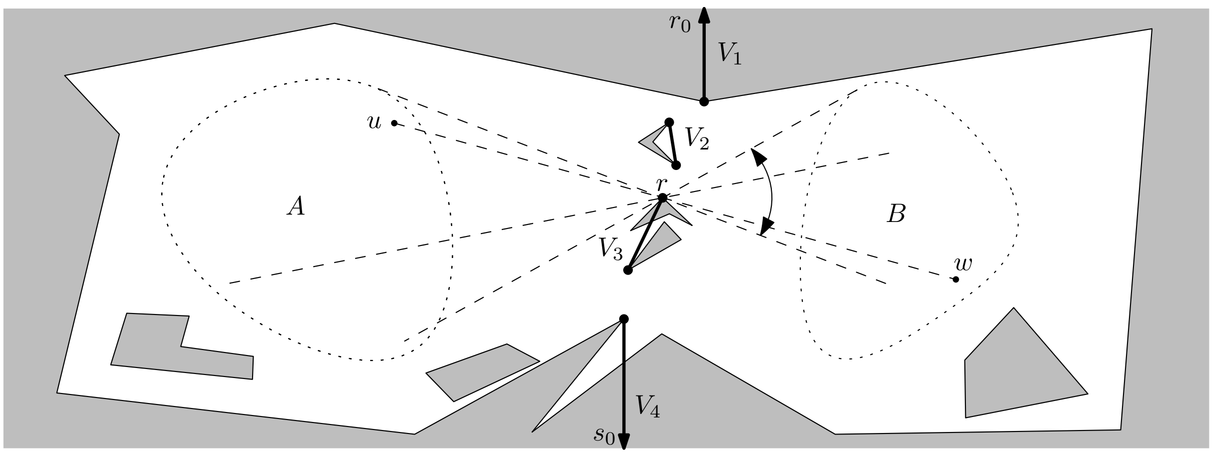


Figure 2: Visibility between two (hypothetical) regions $A$ and $B$ in a polygon with six holes is determined by the mutually exclusive blocking segments $V_1, V_2, V_3, V_4$.

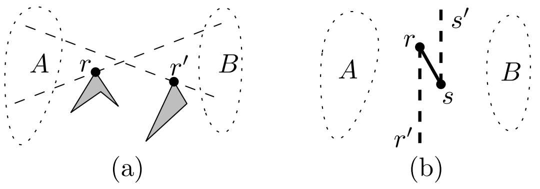


Figure 3: (a) An inconsistent order between blocking vertices cannot happen. (b) Replacing a blocking segment $rs$ by two unbounded blocking rays.

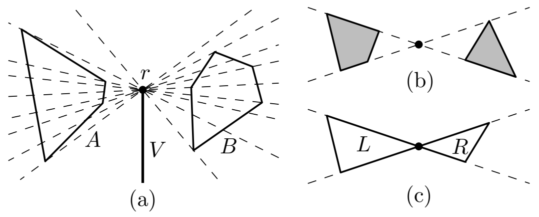


Figure 4: Two cells $A$ and $B$ with a single blocking segment $V$.

we attach a downward ray to an unmatched bottom corner (segment $V_4$ in Fig. 2).

The resulting segments determine the visibility between $A$ and $B$. □

As mentioned above, in Case 3 the calculation can be reduced to considering the blocking probability of single blocking segments: For such a single blocking segment $V = rs$, we have to integrate over the point pairs $(u, w) \in A \times B$ for which $uw \cap V \neq \emptyset$. This can be further reduced to two integrations over unbounded vertical blocking rays $rr'$ and $ss'$, as shown in Fig. 3b. For the integrand, we have

$$[uw \cap V \neq \emptyset] = [uw \cap rr' \neq \emptyset] + [uw \cap ss' \neq \emptyset] - 1$$

For an integral of the form

$$\int_{u \in A} \int_{w \in B} [uw \cap rr' \neq \emptyset] \, dw du$$

we simply have to test whether $uw$ passes above or below the point $r$.

## 5 A Single Blocking Ray

Let us consider the integral

$$\int_{u \in A} \int_{w \in B} [uw \cap V \neq \emptyset] \, dw du \qquad (3)$$

for a single blocking segment $V$ extending from $r$ downward to infinity, see Fig. 4a. We draw a line from each vertex through $r$. These lines decompose the problem into double wedges between adjacent lines. Sectors of $A$ and $B$ which are not in the same double-wedge are blocked by $V$ either completely or not at all, and their contribution to (3) is easy to compute. We are left with the situation of regions in two wedges like in Fig. 4b, where each boundary edge extends between the two rays of the wedge. On each side, we can express the region as a difference of two triangles that involve the apex $r$, as in Fig. 4c. Doing this on each side, the evaluation of the integral for a single double-wedge is reduced to four integrals over triangular regions $L$ and $R$ as in Fig. 4c.

## 6 The Basic Integral

It is now convenient to revert to a probabilistic interpretation of these integrals. The integral (3), for $A = L$ and $B = R$, equals $|L| \cdot |R|$ times the probability that a random point $u \in L$ and a random point $w \in R$ form a counterclockwise triangle with $r$. To evaluate this probability, we make our life easier by transforming the situation to the standard situation shown in Fig. 5. First of all, the problem is invariant under affine transformations. So we assume that the

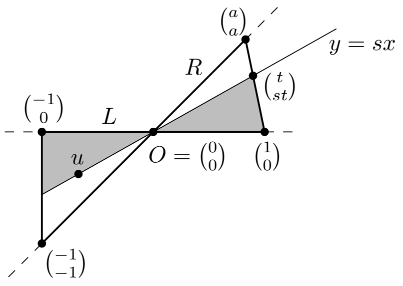


Figure 5: Transforming the problem to the standard situation, for which we evaluate the integral.

apex $r$ is at the origin $O$ and the left triangle $L$ is bounded by the lines of slope 0 and 1 and the line $x = -1$. Scaling the right triangle $R$ from the origin does not change the probability of a positive orientation for the random triangle $uwO$. Thus we can scale $R$ so that the lower corner lies at $\binom{1}{0}$. Then the upper corner lies on the line $y = x$ of slope 1 at some point $\binom{a}{a}$, for $a > 0$. There is only one free parameter, $a$.

The area of $L$ is $1/2$ and the area of $R$ is $a/2$. If we take a random point $u \in L$, the slope $s$ of the line $uO$ is uniformly distributed in $[0, 1]$. (This follows from the fact that the probability that the slope is smaller than $s$ is the area of the shaded region on the left, divided by $1/2$, which equals $s$.) Thus, the probability of blocking is the expected value of the region in the triangle on the right side below the line $y = sx$ of slope $s$, when $s$ chosen uniformly at random, divided by the area $a/2$ of the whole triangle $R$. This region, which is shaded in Fig. 5, is a triangle with vertices $O$, $\binom{1}{0}$, and $\binom{t}{st}$, where $t$ can be evaluated as $t = \frac{a}{a+s-as} > 0$. The area of this triangle is $\frac{st}{2} = \frac{1}{2} \cdot \frac{as}{a+s-as}$, and hence the desired probability is

$$Q(a) := \frac{1}{a} \int_{s=0}^{1} \frac{as}{a+s-as}\, ds = \int_{s=0}^{1} \frac{s}{a+s-as}\, ds$$
$$= \frac{1 - a + a \ln a}{(1-a)^2} \tag{4}$$

This derivation assumes $a \neq 1$. For the symmetric situation ($a = 1$) we get $Q(1) = \frac{1}{2}$. The formula (4) is numerically unstable near $a = 1$. Near $a = 1$ we might therefore resort to the power series expansion

$$Q(a) = \sum_{i=0}^{\infty} \frac{(1-a)^i}{(i+1)(i+2)}$$
$$= \tfrac{1}{2} + \tfrac{1-a}{6} + \tfrac{(1-a)^2}{12} + \tfrac{(1-a)^3}{20} + \tfrac{(1-a)^4}{30} + \cdots$$
$$= \tfrac{1}{2} - \tfrac{\ln a}{6} + \tfrac{(\ln a)^3}{180} - \tfrac{(\ln a)^5}{5040} + \tfrac{(\ln a)^7}{151200} - \cdots$$

## 7 Complexity and Runtime Analysis

Let $P$ have $h$ holes and $n$ boundary edges, and therefore at most $n$ vertices. Suppose $P$ has $n_R$ reflex vertices and $n_B = O(n_R^2)$ interior bitangents. The partition $Z$ is generated by $2n_B + 2n_R$ segments and hence has complexity $n_Z = O((n_B + n_R)^2 + n) = O(n_R^4 + n) = O(n^4)$. For each of the $n_Z^2 = O(n^8)$ pairs of regions $A, B$ we must evaluate the integral $I(A, B)$ and sum up the results. There might be $2+h$ blocking segments, which are reduced to $2+2h$ blocking rays, accounting for a factor of $O(1 + h)$. The decomposition in Fig. 4a incurs an overhead proportional to the complexity of $A$ plus $B$; in total, this is still $O(n_Z^2) = O(n^8)$. Thus, the total number of integrals (3) that we have to compute is $O(n_Z^2(1 + h)) = O((n_B + n_R)^4 + n^2)(1 + h) = O(n_R^8 + n^2)(1 + h) = O(n^8(1 + h)) = O(n^9)$.

Algorithmically, we can compute the $n_B$ bitangents and extension rays in $O(n_R n \log n)$ time by a circular sweep arount each reflex vertex. We can compute the partition $Z$ in $O(n_Z \log n)$ time by a plane sweep.

We pick one of the $O(n_Z)$ regions $A$ and compute the visible region from some arbitrary point $u \in A$, in $O(n \log n)$ time. This yields all the potential lower and upper blocking points and their sorted order around $u$. For every region $B$, we can select the blocking points that lie between $A$ and $B$ (in the convex hull of $A \cup B$). By processing them in sorted order, we can identify the blocking segments. This takes $O(n)$ time per pair $A, B$.

As mentioned above, the overhead from the decomposition into wedges (Fig. 4a) does not add up to more than $O(n_Z^2) = O(n^8)$. Thus, for the overall running time, we get $O((n_R n + n_Z) \log n + n_Z n \log n + n_Z^2 n) = O(n((n_B + n_R)^4 + n^2)) = O(n(n_R^8 + n^2)) = O(n^9)$.

## 8 Other Convexity Measures

Another interesting measure is the average "detour" of the geodesic path between $u$ and $w$ within $P$ (either the *quotient* over the Euclidean distance, or the *difference*, suitably normalized). With quotients, this looks very hard, but computing the excess might be within reach, although I don't even know whether the average Euclidean distance in a polygon is computable in closed form. Taking *squared* distances might be a way out: the average squared distance is just the variance.

The definition (1) extends to higher dimensions, but the computation of this integral is much harder.

## References

[1] L. Boxer. Computing deviations from convexity in polygons. *Pattern Recognition Lett.*, 14:163–167, 1993.

[2] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* Chapman & Hall, 1993.

[3] H. I. Stern. Polygonal entropy: a convexity measure. *Pattern Recognition Letters*, 10:229–235, 1989.

[4] J. Žunić and P. L. Rosin. A new convexity measure for polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):923–934, 2004.

# Algorithms for distance problems in planar complexes of global nonpositive curvature

Daniela Maftuleac

## Abstract

CAT(0) metric spaces and hyperbolic spaces play an important role in combinatorial and geometric group theory. In this paper, we present efficient algorithms for distance problems in CAT(0) planar complexes. First of all, we present an algorithm for answering single-point distance queries in a CAT(0) planar complex. Namely, we show that for a CAT(0) planar complex $\mathcal{K}$ with $n$ vertices, one can construct a data structure $\mathcal{D}$ of size $O(n^2)$ so that, given a point $x \in \mathcal{K}$, the shortest path $\gamma(x, y)$ between $x$ and the query point $y$ can be computed in $O(n^2)$ time. Our second algorithm computes the convex hull of a finite set of points in a CAT(0) planar complex. This algorithm is based on Toussaint's algorithm for computing the convex hull of a finite set of points in a simple polygon and it constructs the convex hull of a set of $k$ points in $O(n^2 + nk \log k)$ time, using a data structure of size $O(n^2 + k)$.

## 1 Introduction

Introduced by Gromov in 1987, CAT(0) metric spaces (or spaces of nonpositive curvature) constitute a far-reaching common generalization of Euclidean and hyperbolic spaces and simple polygons. Most of the known results on CAT(0) metric spaces are mathematical. To the best of our knowledge, from the algorithmic point of view, these spaces remain relatively unexplored. Still there are some algorithmic results in some particular CAT(0) spaces. For example, Billera, Holmes and Vogtmann [2] showed that the space of all phylogenetic trees with the same set of leaves, can be seen as a CAT(0) cube complex. In her doctoral thesis [8], Owen proposed exponential time algorithms for computing the shortest path in the space of phylogenetic trees.
Subsequently, the question of whether the distance and the shortest path between two trees in this CAT(0) space can be computed in polynomial (with respect to the number of leaves) time was raised. Recently, Owen and Provan [9] solved this question in the affirmative; the paper [4] reports on the implementation of the algorithm of [9]. Using the result of [9], Ardila, Owen, and Sullivant [1] described a finite algorithm that computes the shortest path between two points in general CAT(0) cubical complexes. This algorithm is not a priori polynomial and finding such an algorithm that computes the shortest path in a CAT(0) complex of general dimension, remains an open question.
In our paper [5], we proposed a polynomial time algorithm for two-points shortest path queries in 2-dimensional CAT(0) cubical complex and some of its subclasses. In this paper, we present efficient algorithms for single-point distance queries and convex hulls in CAT(0) planar complexes. A detailed description of these results is given in my doctoral thesis [10].

## 2 CAT(0) planar complexes

Let $(X, d)$ be a metric space. A *geodesic* joining two points $x$ and $y$ from $X$ is the image of a (continuous) map $\gamma$ from a line segment $[0, l] \subset \mathbb{R}$ to $X$ such that $\gamma(0) = x, \gamma(l) = y$ and $d(\gamma(t), \gamma(t')) = |t - t'|$ for all $t, t' \in [0, l]$. The space $(X, d)$ is said to be *geodesic* if every pair of points $x, y \in X$ is joined by a geodesic [3]. A *geodesic triangle* $\Delta(x_1, x_2, x_3)$ in a geodesic metric space $(X, d)$ consists of three distinct points in $X$ (the vertices of $\Delta$) and a geodesic between each pair of vertices (the sides of $\Delta$). A *comparison triangle* for $\Delta(x_1, x_2, x_3)$ is a triangle $\Delta(x'_1, x'_2, x'_3)$ in the Euclidean plane $\mathbb{E}^2$ such that $d_{\mathbb{E}^2}(x'_i, x'_j) = d(x_i, x_j)$ for $i, j \in \{1, 2, 3\}$. A geodesic metric space $(X, d)$ is a *CAT(0) space* [6] if all geodesic triangles $\Delta(x_1, x_2, x_3)$ of $X$ satisfy the comparison axiom of Cartan–Alexandrov–Toponogov:
*If $y$ is a point on the side of $\Delta(x_1, x_2, x_3)$ with vertices $x_1$ and $x_2$ and $y'$ is the unique point on the line segment $[x'_1, x'_2]$ of the comparison triangle $\Delta(x'_1, x'_2, x'_3)$ such that $d_{\mathbb{E}^2}(x'_i, y') = d(x_i, y)$ for $i = 1, 2$, then $d(x_3, y) \leq d_{\mathbb{E}^2}(x'_3, y')$.*

A fundamental property of CAT(0) spaces is that any pair of points are connected by a unique geodesic.

A *planar complex* is a 2-dimensional piecewise Euclidean cell complex whose 1-skeleton has a planar drawing in such a way that the 2-cells of the complex are exactly the inner faces of the 1-skeleton in this drawing.
A planar complex $\mathcal{K}$ can be endowed with the intrinsic $l_2$-metric in the following way. Suppose that inside every 2-cell of $\mathcal{K}$ the distance is measured according to an $l_2-$metric. A *path* between $x, y \in \mathcal{K}$ is a sequence of points $p = (x_0 = x, x_1, \ldots, x_{k-1}, x_k = y$ such that any two consecutive points $x_i, x_{i+1}$ belong

to a common cell. The length of a path is the sum of distances between all pairs of consecutive points of this path. Then the intrinsic $l_2$−metric between two points of $\mathcal{K}$ equals to the infimum of the lengths of the finite paths joining them. A planar complex $\mathcal{K}$ is a planar CAT(0) complex if with respect to its intrinsic metric, $\mathcal{K}$ is a CAT(0) space. Equivalently, a planar complex is CAT(0) planar (see Fig. 2) if for all inner 0-cells the sum of angles in one of these points is at least equal to $2\pi$. By triangulating each 2-cell of a CAT(0) planar complex, we can assume without loss of generality and without changing the size of the input data, that all 2-cells (faces) of $\mathcal{K}$ are isometric to arbitrary triangles of the Euclidean plane.

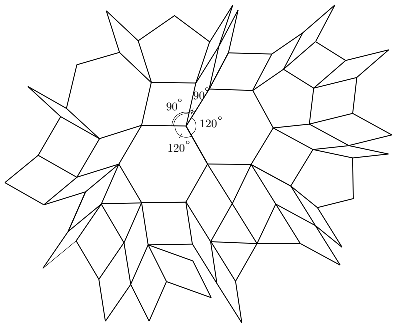


Figure 1: A CAT(0) planar complex

## 3 Shortest path map

The notion of shortest path map was introduced by Hershberger and Suri [7] as a preprocessing step for *continuous Dijkstra algorithm*. This algorithm computes the shortest distance between a given point and any other point in a polygon with holes or in the plane in the presence of obstacles. This method is a conceptual algorithm to compute shortest paths from a given source $s$ to all other points, by simulating the propagation of a sweeping line from a point to all points using scanning level-lines. The output of the continuous Dijkstra method is called the *shortest path map* SPM($s$) of a given point $s$. The shortest path map SPM($s$) is a subdivision of the polygon (free-space) into cells, such that each cell is a set of points whose shortest paths to $s$ are equivalent from a combinatorial point of view.

The shortest path map SPM($x$) in a CAT(0) planar complex $\mathcal{K}$ is a partition of the complex in convex cones. This partition has a shortest path tree structure with the common root $x \in \mathcal{K}$. The construction of SPM($x$) is used in our algorithms for computing the shortest path and the convex hull of a finite set of points in $\mathcal{K}$. Thus we will give a more detailed definition of SPM($x$) in $\mathcal{K}$.

A *cone* in a CAT(0) planar complex $\mathcal{K}$ is the set of points of the complex located between the geodesics

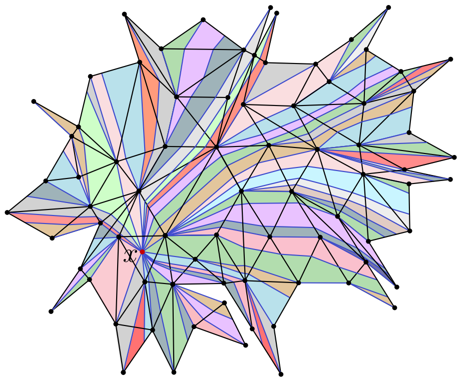


Figure 2: A shortest path map SPM($x$) in a CAT(0) planar complex

$\gamma(u,v), \gamma(u,w)$ in the region containing the angle $\alpha < \pi$, where $v$ and $w$ belong to the boundary of $\mathcal{K}$; we denote by $\mathcal{C}(u;v,w)$ the cone of origin (apex) $u$, where $\gamma(u,v)$ and $\gamma(u,w)$ are the *sides* of this cone. By the *interior* of the cone $\mathcal{C}(u;v,w)$ we mean the set of points $\text{int}(\mathcal{C}(u;v,w)) = \mathcal{C}(u;v,w) \setminus (\gamma(u,v) \cup \gamma(u,w))$.

A formal definition of SPM($x$) is as follows: given a point $x \in \mathcal{K}$, the *shortest path map SPM(x)* is a partition of $\mathcal{K}$ into cones $\mathcal{C}(z_i;p_i,q_i)$, $i \in I$, where $I = \{i_1, i_2, \ldots, i_m\} \subset \mathbb{N}$, such that

(1) for every vertex $y$ of $\mathcal{K}$ there exists a geodesic $\gamma(x,p)$, with $p$ on the boundary of $\mathcal{K}$ which passes via $y$ and:
  (a) if $\alpha(y) = 2\pi$, then $y$ belongs to a common side $\gamma(z,p)$ of the two cones $\mathcal{C}(z;p,q)$, $\mathcal{C}(z';p,r)$ of SPM($x$);
  (b) if $y$ is a vertex of negative curvature, then $y$ is the apex of at least one cone $\mathcal{C}(y;p,q)$ of SPM($x$);

(2) let $\mathcal{C}(z;p,q)$ be a cone of SPM($x$), then its apex $z$ belongs to the geodesics $\gamma(x,p)$ and $\gamma(x,q)$.

We continue with a list of simple but essential properties of the shortest path map SPM($x$) in a CAT(0) planar complex $\mathcal{K}$.

**Proposition 1** *Let $\mathcal{K}$ be a CAT(0) planar complex, $x$ a point of $\mathcal{K}$ and SPM($x$) the shortest path map of $\mathcal{K}$, then the following conditions are satisfied:*

*(i) The shortest path map SPM($x$) has a geodesic tree structure;*

*(ii) Each cone $\mathcal{C}(z;p,q)$ of SPM($x$) is a convex subset of $\mathcal{K}$;*

*(iii) If $\mathcal{C}(z;p,q)$ is a cone of SPM($x$), then int($\mathcal{C}(z;p,q)$) contains no vertices of $\mathcal{K}$;*

*(iv) The angle $\angle_z(p,q)$ formed by the sides $\gamma(z,p)$ and $\gamma(z,q)$ of a cone $\mathcal{C}(z;p,q)$ of SPM($x$) is less or equal to $\pi$;*

*(v) An inner point of $\mathcal{K}$ belongs either to a single cone, or to a common side of two cones of SPM($x$);*

*(vi) For any point $u$ belonging to one side $\gamma(z,p)$ of a cone $\mathcal{C}(z;p,q)$ of SPM$(x)$, the angle $\angle_u(z,p)$ is at least $\pi$ inside $\mathcal{C}(z;p,q)$.*

The next two lemmas present some properties of the shortest path map in a CAT(0) planar complex $\mathcal{K}$, which will allow us to describe an efficient algorithm for constructing SPM$(x)$.

**Lemma 2** *For any point $y$ of $\mathcal{K}$, the shortest path between $x$ and $y$ passes through the apex of the cone of SPM$(x)$ containing $y$.*

**Lemma 3** *All cones of SPM$(x)$ can be embedded in the plane as acute triangles.*

We propose a sweeping-line algorithm for constructing the shortest path map in a CAT(0) planar complex $\mathcal{K}$. The algorithm visits the faces of $\mathcal{K}$ using a *sweeping(or level)-line*, denoted by $C(r)$. We call *events* of the sweep, all the crossing points of $C(r)$ from one face to another. The sweeping line $C(r)$ at the $r$ instance consists of a sequence of arcs of circles of radius $r$ and a fixed center $x$, concatenated by *break points*. These arcs appear when $C(r)$ crosses the events of the sweep.
When $C(r)$ passes through an event $h$, we construct the geodesic $\gamma(x,h)$ between $h$ to the root $x$ in $\mathcal{K}$. All the geodesics constructed this way form a partition of the complex $\mathcal{K}$ into convex regions called *cones* of SPM$(x)$.

There are two types of events encountered by the sweeping line of $\mathcal{K}$: edge-events and vertex-events.

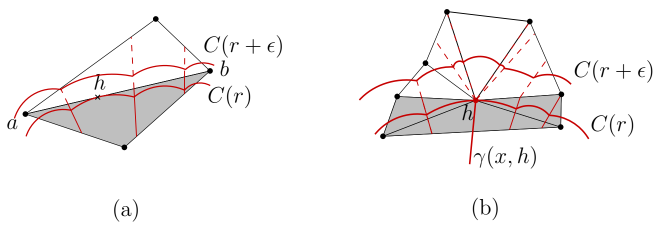


Figure 3: (a) Edge-event, (b) Vertex-event.

The *vertex-events* are all vertices of $\mathcal{K}$. The passage of the sweeping line $C(r)$ through an event $h$ of this type creates new arcs on $C(r)$ (see Fig. 3 (b)). When $C(r)$ passes via $h$, we construct the geodesic $\gamma(x,h)$ connecting the points $x$ and $h$. The information for the sweep of $\mathcal{K}$ is then transmitted from the face already covered by $C(r)$ which contains $h$ to all other faces incident to $h$.

The *edge-events* are inner points of edges of $\mathcal{K}$. For some edge-event $h$, there exists a face $F = \Delta(a,b,c)$ of $\mathcal{K}$, and a cone $\mathcal{C}(z;p,q)$ of SPM$(x)$ such that $h$ is the closest point of $F$ to $x$ in $\mathcal{C}(z;p,q)$. An edge $e$ of $\mathcal{K}$ can contain multiple edge-events (see Fig. 3 (a)). The passage of the sweeping line $C(r)$ through some edge-event $h$ does not change the shape of $C(r)$. These events are only used to transmit the information for the sweep from one face to another.

The data structure used by our algorithm, consists of two substructures: a *static* substructure $D_s$, which does not change during the steps of the algorithm, and a *dynamic* substructure $D_d$, which is initialized at step one of the algorithm and changes during the sweep of $\mathcal{K}$.
The static substructure contains the planar map of the complex $\mathcal{K}$ and the circular lists of angles of every vertex of $\mathcal{K}$. At the instance $r$ of the sweep, the dynamic substructure contains a priority queue $\mathcal{Q}$ of events crossed by $C(r)$ and a list $\mathcal{C}$ of cones constructed up to $C(r)$. Note that the intersection of a triangular face of $\mathcal{K}$ with a cone of SPM$(x)$ is at most a quadrilateral. Thus, the dynamic substructure $D_d$ contains, for any face $F$ of $\mathcal{K}$ the list of quadrilaterals issued from the intersection of $F$ with cones of $\mathcal{C}$.
We use the representation of the complex as a planar map which is a doubly-connected edge list. This representation allows us to use a data space of linear size with respect to the number of vertices of $\mathcal{K}$.

Given a CAT(0) planar complex $\mathcal{K}$ and a point $x \in \mathcal{K}$ we construct the shortest path map SPM$(x)$ as a geodesic tree structure.

**Theorem 4** *Given a CAT(0) planar complex $\mathcal{K}$ with $n$ vertices and a point $x$ of $\mathcal{K}$, it is possible to construct the shortest path map SPM$(x)$ of $\mathcal{K}$ in $O(n^2)$ time, using $O(n^2)$ space.*

## 4 Shortest path problem in CAT(0) planar complexes

We consider the following problem: given a source-point $x$ in a CAT(0) planar complex $\mathcal{K}$, preprocess $\mathcal{K}$ so that for each query point $y$ in $\mathcal{K}$ one can compute the unique shortest path $\gamma(x,y)$ between $x$ and $y$.

In order to solve this problem, we propose an algorithm based on the *continuous Dijkstra method* for computing the shortest path between a given point and any query point inside a polygon with holes.
Given a point $x$ in a CAT(0) planar complex $\mathcal{K}$, we construct the shortest path map SPM$(x)$ in $\mathcal{K}$. Moreover, we show that all cones of SPM$(x)$ can be unfolded in $\mathbb{R}^2$, such that shortest path between $x$ and every query point $y$ is the same inside the cone containing $y$ as in the unfolding of the cone in $\mathbb{R}^2$ between the images of $x$ and $y$.
First we compute the shortest path map SPM$(x)$ in $O(n^2)$ time as described in previous section. Then for

any query point $y$ in $\mathcal{K}$, our algorithm determines the cone containing $y$, unfolds the cone in the plane as an acute triangle (Lemma 3) and computes the shortest path $\gamma(x, y)$ between $x$ and $y$ inside $\mathcal{K}$. The geodesic $\gamma(x, y)$ is calculated in $O(n)$ time as the isometric image of the shortest path between the images of $x$ and $y$ in the plane.

The following theorem gives the main result regarding the one-point query version of the shortest path problem in a CAT(0) planar complex.

**Theorem 5** *Given a CAT(0) planar complex $\mathcal{K}$ with $n$ vertices, a point $x \in \mathcal{K}$ and the shortest path map SPM($x$), it is possible to construct a data structure of size $O(n^2)$ such that, for a query $y \in \mathcal{K}$ the algorithm computes the shortest path $\gamma(x, y)$ between $x$ and $y$ in $\mathcal{K}$ in linear time $O(n)$.*

## 5 Computing convex hulls in a CAT(0) planar complex

Let $S$ be a finite set of points in a planar CAT(0) complex $\mathcal{K}$. We consider the problem of computing the convex hull conv($S$) in $\mathcal{K}$.
In order to solve this problem, we propose an algorithm based on Toussaint's algorithm [11], which constructs the convex hull of a finite point-set inside a triangulated simple polygon in $O(n \log n)$ time, using a data structure of size $O(n)$.

As the pre-processing step, our algorithm builds the shortest path map SPM($x$), where $x$ is an arbitrary point of the boundary of $\mathcal{K}$. Note that this partition of $\mathcal{K}$ in cones will play the role of the triangulation of a simple polygon. Then the algorithm unfolds each cone SPM($x$) in the plane, and computes the convex hull of the subset of $S$ located in this cone. Thus we obtain a finite number of convex hulls in $\mathcal{K}$, each contained in a cone $\mathcal{C}(z_i; p_i, q_i)$, $i \in I' \subseteq I$, so that $\mathcal{C}(z_i; p_i, q_i) \cap S \neq \emptyset$. As $x$ is chosen on the boundary of $\mathcal{K}$, SPM($x$) can be seen as a canonical counterclockwise non-cyclic sequence of cones. We refer to two consecutive convex hulls constructed earlier, as the convex hulls contained in two cones of SPM($x$), so that any other cone of the subsequence defined by these two cones contains no points of $S$.
A closed polygonal line of the complex is called a *weakly-simple polygon* if the line divides the complex into two areas equivalent to a disc. Less formally, we can say that a weakly-simple polygon can have sides that touch but do not intersect.
By connecting with a geodesic segment each pair of consecutive convex hulls constructed at the previous step, we obtain a weakly-simple polygon in $\mathcal{K}$.

We denote by $P$ the union of the weakly-simple polygon and a geodesic segment connecting two points $p$ and $q$, where $p$ and $q$ are the closest points, such that $p$ belongs to the boundary of $P$ and $q$ belongs to the boundary of $\mathcal{K}$. We show that the complex $\mathcal{K} \setminus P$ is CAT(0) and planar. Further we denote by $p^*$ a copy of the point $p$ in $\mathcal{K}$. Finally, we show that the shortest path between $p$ and $p^*$ in the CAT(0) planar complex $\mathcal{K} \setminus P$ coincides with the boundary of the convex hull of $S$ in $\mathcal{K}$. The shortest path between $p$ and $p^*$ in $\mathcal{K} \setminus P$ can be found by the algorithm of previous section.

The following theorem summarizes the main result concerning the convex hull problem in a CAT(0) planar complex.

**Theorem 6** *Given a CAT(0) planar complex $\mathcal{K}$ with $n$ vertices, one can construct a data structure of size $O(n^2 + k)$, such that for any finite set $S$ of $k$ points in $\mathcal{K}$, the algorithm computes the convex hull conv($S$) in $O(n^2 + nk \log k)$.*

## References

[1] F. Ardila, M. Owen and S. Sullivant, Geodesics in CAT(0) cubical complexes, Advances in Applied Mathematics **48** (2012), 142–163.

[2] L.J. Billera, S.P. Holmes and K. Vogtmann, Geometry of the space of phylogenetic trees, Adv. Appl. Math. **27** (2001), 733–767.

[3] M. Bridson and A. Haefliger, Metric Spaces of Non-Positive Curvature, Springer-Verlag, 1999.

[4] J. Chakerian and S.P. Holmes, Computational tools for evaluating phylogenetic and hierarchical clustering trees, JCGS, **21**(3) (2012), 581–599.

[5] V. Chepoi and D. Maftuleac, Shortest path problem in rectangular complexes of global nonpositive curvature, Computational Geometry, **46** (2013), 51–64.

[6] M. Gromov, Hyperbolic groups, Essays in Group Theory (S. M. Gersten, ed.), MSRI Publications, vol. 8, Springer-Verlag (1987), 75–263.

[7] J. Hershberger and S. Suri, An optimal algorithm for Euclidean shortest paths in the plane, SIAM J. Comput. **28** (1999), 2215–2256.

[8] M. Owen, Distance computation in the space of phylogenetic trees, Thesis, Cornell University, 2008.

[9] M. Owen and S. Provan, A fast algorithm for computing geodesic distances in tree space, ACM/IEEE Transactions on Computational Biology and Bioinformatics **8** (2011), 2–13.

[10] D. Maftuleac, Algorithmique des complexes CAT(0) planaires et rectangulaires, Thesis, Aix-Marseille University, 2012.

[11] G.T. Toussaint, Computing Geodesic Properties Inside a Simple Polygon, Revue d'intelligence artificielle, vol. 3, **2** (1989), 9–42.

# On the Complexity of Finding Spanner Paths

Mikael Nilsson*

## Abstract

We study the complexity of determining if a spanner path exists between two given nodes in a given Euclidean graph. This problem that we call the t-path problem is proven to be NP-complete for non-constant spanner stretches (e.g. stretch $(2n)^{3/2}$). An algorithm to solve the problem is given. It improves on the naive $O(2^n)$ complexity to give $O(2^{0.822n})$.

We also study the same problem for a type of one-dimensional graphs that we call Integer Graphs. A more efficient algorithm can be devised for these graphs resulting in a complexity of $O(2^{c(\log n)^2})$, where $c$ is a constant depending only on the spanner stretch.

## 1 Introduction

Andersson et al. [1] present an algorithm for building approximate distance oracles for graphs with dense clusters. The algorithm assumes that the Euclidean input graph is partitioned into so-called islands that are all spanners. Nodes connecting islands are called airports. The gain from using the oracle is the reduction in size that can be achieved compared to the naive $n^2$ lookup table. The size used by the constructed oracle is $O(M^2 + n \log n)$ where $n$ is the total number of nodes and $M$ is the number of airports.

If we want to construct an oracle for a given graph, we first have to put it in the required input format. While doing this we want to minimize the number of airports to keep the oracle's size down. This is a combinatorial optimization problem that has been discussed by Nilsson [5].

If we had knowledge about which nodes could belong to the same island, optimization might become easier. Two nodes can belong to the same island if and only if there is a path linking them that is a spanner path. When applying the spanner concept to paths we get an approximation requirement on the path requiring that any distance between nodes along the path is within the stretch factor times the Euclidean distance between them. Detecting the existence of spanner paths is a complicated problem that we devote this paper to.

*Linköping University, Sweden, `mikael.a.nilsson@liu.se`

## 2 Defining the $t$-path Problem

A Euclidean graph (we work exclusively with these) is a *t-spanner* if the distance between any pair of nodes via edges in the graph is at most $t$ times the Euclidean distance between them. Hence the factor $t$, also called the *stretch*, determines the quality by which the graph approximates Euclidean distances.

Note that by this definition a complete Euclidean graph is a 1-spanner.

A *t-path* between two nodes is a path that is a t-spanner. This means that any distance between nodes along the path must be within a factor $t$ of their Euclidean distance. Do not confuse the definition used in this paper with that of a $t$-path used in the context of Restricted Shortest Path problems (see Hassin [2]).

The *t-path problem* consists of deciding if there is a $t$-path between two given nodes in a Euclidean graph.

## 3 A simple algorithm

First we present a simple algorithm for the $t$-path problem. The algorithm is recursive and in each iteration it uses Dijkstra's algorithm to find the shortest path between the start and end node of the sought $t$-path.

If this shortest path is not a $t$-path there are at least two nodes along the path that conflict with the spanner condition. If there are more conflicting pairs we chose one pair in each iteration. Since the chosen pair is connected along a shortest path there can never be a shorter path between the problem nodes. Hence they can never both be on the $t$-path. The algorithm proceeds by removing one of them and recursively tries to solve the problem on the resulting smaller graph. If this does not succeed, it puts the node back, removes the other problem node and recurses in the same way.

When analyzed, this algorithm is found to run in time $O(2^n)$ (the recurrence equation is $t_n = O(n^2) + 2t_{n-1}$, where $t_n$ denotes the time to run the algorithm on $n$ nodes and $O(n^2)$ comes from finding the shortest path and testing the spanner requirement). It is possible to improve this to $O(2^{0.822n})$; the calculations cannot fit here though (see Nilsson [5]). The idea is that since the order in which nodes are removed is not taken into consideration, several recursions check the same graph. Especially, as the size of the graphs shrinks the number of times they are checked increases. By building a lookup table of small graphs the recursion can be halted earlier.

## 4 NP-Completeness of the $t$-path Problem

We start by observing that the $t$-path problem is in NP. Given a path from a start node $s$ to an end node $e$ we can check in $O(n^2)$ polynomial time if the path is a $t$-path. This is done by comparing distances along the path, incrementally calculated in $O(n)$ steps, to the corresponding Euclidean distances for all pairs of nodes on the path (a total of $O(n^2)$ comparisons). The NP-completeness then follows by reducing a known NP-complete problem to the $t$-path problem.

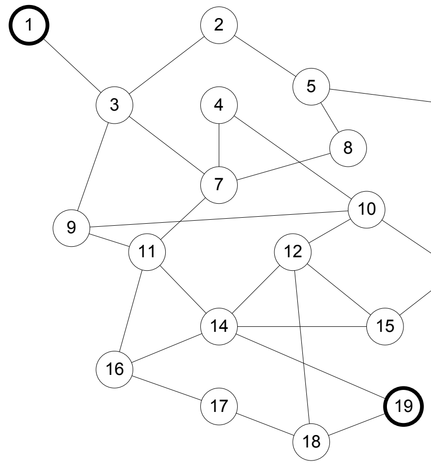
Let $G$ be a directed graph containing the special nodes $s$ and $e$. Furthermore, let $C$ be a list of node pairs. The problem *Path with Forbidden Pairs*, abbreviated *PwFP*, consists of finding out whether there exists a path in $G$ from $s$ to $e$ that at most contains one node from each pair in $C$. The problem was first formulated and examined by Gabow et al. [3].

It has been proven that this problem is NP-complete and that various versions of it are still NP-complete (see for instance Garey and Johnson [4]). The version we need is the one where the graph is undirected and all forbidden pairs are disjoint; this problem is still NP-complete (see Nilsson [5]).

We now reduce PwFP in the specified version to the $t$-path problem showing that this also is NP-complete. First we give the general reduction scheme and then follow this by an example. It might help to look at the example pictures while following the reduction steps.

Given an instance of the PwFP problem (see figure 1), we start by creating a Euclidean graph. The graph is inscribed in a square that is subdivided into a grid of smaller squares (see figure 2). We then put the original nodes in the smaller squares. Nodes that are not part of a forbidden pair are put by themselves in the center of any empty square. Nodes corresponding to a forbidden pair (these are disjoint) are put together in any empty square, close to the square center with a small distance between them (this distance will be established later). This means we need between $n$ and $n/2$ smaller squares. To be on the safe side we assume that $n$ squares are created. If we set the smaller square side to 1 this gives us an outer square with sides measuring $\leq \sqrt{n}$. We now add edges that connect the same nodes as in the original problem. Edges are assigned lengths corresponding to the Euclidean distance between the connected nodes.

We now estimate the longest path distance between two nodes in this new graph. A rough upper bound can be calculated assuming that all edges are of maximum length. Because of the size of the outer square we know that any edge is shorter than $\sqrt{2n}$. This means that the longest path is bounded by $\sqrt{2}n^{3/2}$. The shortest Euclidean distance between two nodes in different squares is at least $1/2$. This means that the highest possible stretch in the graph is below



|  |  |  |
|---|---|---|
|  | (1,4) | (13,19) |
| Forbidden Pairs: | (3,8) | (16,6) |
|  | (5,10) | (14,18) |

Figure 1: Example of a PwFP problem instance.

$\sqrt{2}n^{3/2}/(1/2) = 2\sqrt{2}n^{3/2} = (2n)^{3/2}$ (here we do not count paths visiting two nodes in the same square as these will be prohibited). We denote this stretch by $T$. If a stretch of $T$ is allowed there can be no violations of the spanner constraint along paths as long as only one node in each square is visited.

We now estimate the shortest path between two nodes in the same square. Since they are not connected it is a path having at least two edges. Since each edge is at least $1/2$ this path measures at least 1. By now setting the distance between the forbidden pair nodes to be $T^{-1} - \varepsilon$ we can prevent these nodes from both being part of a $t$-path with stretch $T$. This is because the distance along a path ($\geq 1$) divided by the real distance becomes $\geq 1/(T^{-1} - \varepsilon) > T$.

If we now regard this newly created graph as an instance of the $t$-path problem with stretch $T$ we will see that we can find a solution to this problem if and only if there is a solution to the original PwFP problem instance. We have then shown that any PwFP problem instance can be reduced to a $t$-path problem instance which together with the facts that PwFP is NP-complete and $t$-path is in NP means that the $t$-path problem is also NP-complete.

Suppose there is a solution to the PwFP instance. This means there is a path which goes from $s$ to $e$ without visiting more than one node from each pair. If we consider the corresponding path in the reduction graph, we see that it is a path that never breaks the spanner condition (since the stretch allowed is large enough to allow any path that do not contain two nodes from the same square). Hence we have a $t$-path. If on the other hand we have a $t$-path between $s$ and $e$ in the reduction graph this must correspond to a path which never visits two nodes in the same square since that would break the spanner condition (the distance

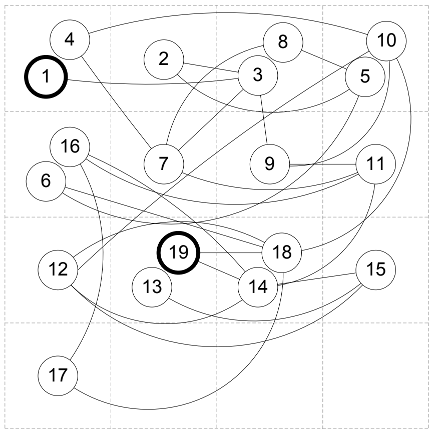


Figure 2: Resulting example reduction graph.

between them was set short enough that including them both violates the condition). This means that we have a path which goes from $s$ to $e$ without visiting a forbidden pair, hence there is a solution to the PwFP problem instance. We see that the reduction presented works and conclude that the $t$-path problem is NP-complete for stretch $T$.

Figures 1 and 2 show an example PwFP instance and an example reduction graph created for it. In the example, a path between node 1 and node 19 is sought. The edges in figure 1 has no edge distances. Distances are present but not shown in figure 2. The edges are drawn so it is possible to see that they still connect the same nodes.

We have shown that for stretches larger than $(2n)^{3/2}$ the $t$-path problem is NP-complete. By increasing the dimension of the constructed graph (fitting it in a hypercube) it is possible to shrink the required stretch to $2\sqrt{k}n^{1+1/k}$ where the integer k denotes the dimension of the constructed graph.

## 5 The $t$-path Problem for Integer Graphs

An *Integer Graph* with $n$ nodes is a one-dimensional Euclidean graph, where all nodes are placed at integer positions on the real line. The leftmost node is placed at 0 and the remaining nodes are placed at $1, 2, 3, \ldots, n-1$. The *t-path problem in the integer graph* consists of finding a $t$-path from node 0 to node n-1. Figure 3 shows an example of an integer graph.

Since nodes in integer graphs are positioned on integer positions the minimum node distance is 1. This together with the fact that the graph is one-dimensional rules out a reduction like the one we just saw. Can the $t$-path problem be solved efficiently in integer graphs?

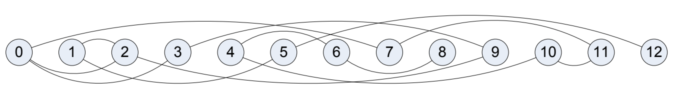


Figure 3: Example of an integer graph.

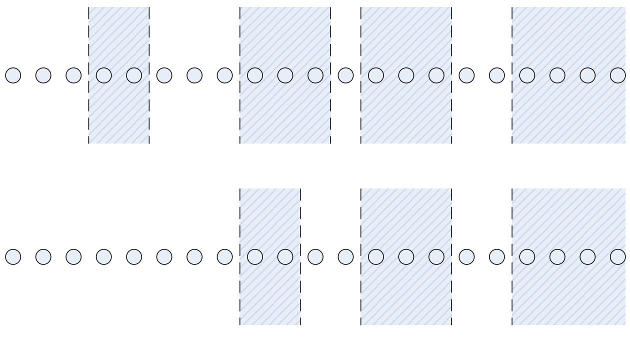


Figure 4: Two comparable images. The dashed intervals can be visited in later stages without breaking the spanner requirement.

The problem can be solved efficiently for stretch 1 and $n^2/2$. Stretch 1 follows since then the path may never go left. Stretch $n^2/2$ follows since this allows all paths (it can be proven that the maximum length path is $\leq n^2/2$, see Nilsson [5]).

It is tempting to guess that since the problem can be efficiently solved for stretches greater than $n^2/2$ and stretch 1 this also applies to the whole interval $[1, n^2/2]$. Although this remains an open problem we will now examine an algorithm for the integer graph $t$-path problem which is more efficient than the general algorithm given in section 3.

Our integer graph algorithm uses a data structure we call an *image*. An image is centered at a node. It contains all intervals of nodes that may be visited in future stages given the path used to reach the center node. An image centered at node 0 consists of a single interval containing all nodes. When a path is built, the interval will be continually subdivided into smaller intervals due to the spanner constraint.

A partial order is imposed on the set of images by the image quality concept. The *image quality* of one image is ‘better’ than that of another if the intervals of the first image cover all intervals of the second image. By the definition, not all images are comparable. Figure 4 shows an example.

The algorithm starts from node 0 and works iteratively towards $n-1$. Every time it extends a path to reach a new node, an image capturing the nodes that can be visited in the future, is created.

The algorithm keeps the invariant “in iteration $k$ all images for nodes $\leq k$ created by paths using only nodes $\leq k$ are found”.

Iteration $k$ starts with checking the images of nodes $< k$. If any of these, e.g. $A$, allows for $k$ to be visited a new image is created for $k$ which is created by shrinking the intervals in $A$ to accommodate this new edge. When images for $k$ are created they are checked to see which nodes $< k$ can be reached from $k$ via these new images. If a node $< k$ is found it is visited and gets a new image. This can result in recursive behavior where the path reaches more nodes $< k$.

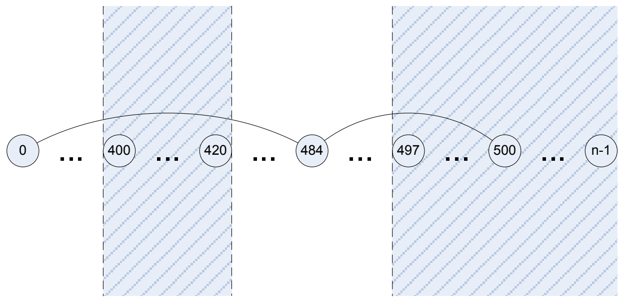


Figure 5: Example of a node close to the pivot node.

The algorithm stores for each node only images that are incomparable. If a comparable image is generated, only the best image will be kept. Figure 4 shows an example where the upper image will cause the lower to be discarded.

Two images can be compared in time $O(\log n)$ if each image consists of a list of interval endpoints. This follows since the maximum number of intervals in any image is bounded by $\log k$ where $k$ is the node location where the image is created (see Nilsson [5]).

The number of intervals has a direct impact on the complexity of the algorithm so it must be examined. Let the stretch of the problem be $1 + t$. We get an upper bound by assuming the worst in each case in the following discussion. First we check the number of intervals that the image for a node can contain on its left side. The distance these intervals occupy is limited due to the spanner constraint (it is not possible to go back too far). In node $k$ the image cannot contain intervals to the left of point $p = 2k/(2 + t)$. This sets the maximum length of intervals to the left of node $k$ to $kt/(2 + t)$. How many times can this interval be subdivided into smaller intervals? The cause for subdivision is that some node or nodes inside the interval are visited on the path to $k$.

There exists a pivot point in the original interval. If the path visits nodes to the left of this there will be no new left interval created in the subdivision. However if the path only visits nodes to the right of the pivot point a new interval to the left will be created in the subdivision. An example of this can be seen in figure 5 where a node close to the pivot point is chosen for the path.

In the example figure the stretch is 1.5 allowing the path to reach as far back as 400 once it has visited 500. The pivot point of the interval [400,500] in an image centered at node 500 is 480. If a node to the left of 480 is visited along the path, the interval will have no left subdivision.

Each time a new interval is created the pivot point moves to the right until it finally reaches $k$. Calculating the number of times the interval can be subdivided gives ${}^{\frac{t}{2+t}}\log\frac{1}{k}$ subdivisions, where $\frac{t}{2+t}$ is the base of the logarithm. In order to get an upper bound on the number of possible images we assume that each of these intervals can be of any length between 1 and the maximum length. This gives a total number of images centered at $k$ which is bounded by $O(2^{\frac{(\log k)^2}{\log a}})$ where $a = (2 + t)/t$. Further details can be found in Nilsson [5].

We now have an upper bound on the number of images that come from intervals to the left of $k$. There are also intervals to the right of the node being processed. These intervals are created by the path visiting nodes to the right of $k$ and then in the end going to $k$. Since the path has already passed by $k$ this right interval will be smaller than the left. However to get an upper bound we let them have the same cardinality. This gives the final number of images in a node during the algorithm's construction of the $t$-path to be bounded by $O(2^{\frac{2(\log n)^2}{\log a}})$.

Factoring in the time to build new images (using a conservative $n^3$ for this) and process all nodes from 1 to $n - 1$ the algorithm's runtime is bounded by $O(2^{c(\log n)^2})$ where $c$ is a constant depending only on the spanner stretch. The constant increases approximately linearly and has for example the value 5.32 at stretch 1.5.

The algorithm does not compute the $t$-path directly since no information of the path which led to a node is kept. However by running the algorithm $n$ times and each time removing a different node it is possible to see which nodes are part of the $t$-path which can then be found by using Dijkstra's algorithm on the remaining nodes.

## Acknowledgments

Thanks go to Christos Levcopoulos for numerous discussions on the $t$-path problem and related topics.

## References

[1] M. Andersson, J. Gudmundsson and C. Levcopoulos. Approximate distance oracles for graphs with dense clusters. *Comput. Geom. Theory Appl.,* 37(3):142-154, 2007.

[2] R. Hassin. Approximation schemes for the restricted shortest path problem. *Math. Oper. Res.,* 17(1):36-42, February 1992.

[3] H.N. Gabow, S.N. Maheshwari and L.J. Osterweil. On Two Problems in the Generation of Program Test Paths. *IEEE Trans. Softw. Eng.,* 2(3):227-231, 1976.

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

[5] M. Nilsson. *Spanneröar och spannervägar.* Institutionen för datavetenskap, Lund, 2009. `http://tinyurl.com/6gbxdnj`.

# Polylogarithmic Approximation for Generalized Minimum Manhattan Networks

Aparna Das* Krzysztof Fleszar† Stephen Kobourov* Joachim Spoerhase† Sankar Veeramoni* Alexander Wolff†

## Abstract

We consider the *generalized minimum Manhattan network problem* (GMMN). The input to this problem is a set $R$ of $n$ pairs of terminals, which are points in $\mathbb{R}^d$. The goal is to find a minimum-length rectilinear network that connects every pair in $R$ by a Manhattan path, that is, a path of axis-parallel line segments whose total length equals the pair's Manhattan distance. This problem is a generalization of the extensively studied minimum *Manhattan network problem* (MMN) in which $R$ consists of all possible pairs of terminals. Another important special case is the well-known *rectilinear Steiner arborescence problem* (RSA). As a generalization of these problems, GMMN is NP-hard but approximation algorithms are only known for MMN and RSA.

We give the first approximation algorithm for GMMN; our algorithm has a ratio of $O(\log^{d+1} n)$ for the problem in arbitrary and fixed dimension $d$. This is an exponential improvement upon the $O(n^\varepsilon)$-ratio of an existing algorithm for MMN in $d$ dimensions [ESA'11]. For the important case of dimension $d = 2$, we derive an improved bound of $O(\log n)$. Finally, we show that an existing $O(\log n)$-approximation algorithm for two-dimensional RSA generalizes to higher dimensions.

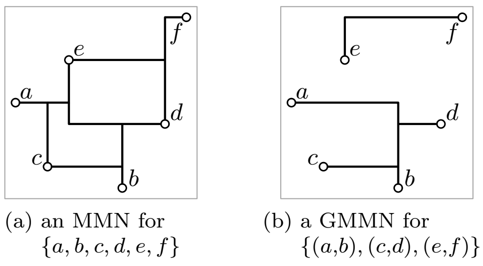
## 1 Introduction

Given a set of terminals, which are points in $\mathbb{R}^d$, the *minimum Manhattan network problem* (MMN) asks for a minimum-length rectilinear network that connects every pair of terminals by a Manhattan path (*M-path*, for short), that is, a path consisting of axis-parallel segments whose total length equals the pair's Manhattan distance.

In the *generalized minimum Manhattan network problem* (GMMN), we are given a set $R$ of $n$ unordered terminal *pairs*, and the goal is to find a minimum-length rectilinear network such that every pair in $R$ is *M-connected*, that is, connected by an M-path.

*Department of Computer Science, University of Arizona, Tucson, AZ, U.S.A., http://www.cs.arizona.edu/~kobourov

†Lehrstuhl I, Institut für Informatik, Universität Würzburg, Germany, http://www1.informatik.uni-wuerzburg.de/en/staff



(a) an MMN for $\{a, b, c, d, e, f\}$ (b) a GMMN for $\{(a,b), (c,d), (e,f)\}$

**Figure 1:** MMN versus GMMN in 2D.

GMMN is a generalization of MMN since $R$ may contain all possible pairs of terminals. Figure 1 depicts examples of both network types. We remark that, in this paper, we define $n$ to be the number of terminal *pairs* of a GMMN instance, whereas previous works on MMN defined $n$ to be the number of *terminals*.

Two-dimensional MMN (2D-MMN) naturally arises in VLSI circuit layout; higher-dimensional MMN has applications in the area of computational biology. MMN requires a Manhattan path between every terminal pair. This assumption is, however, not always reasonable. For example, in VLSI design a wire connection is necessary only for a, often comparatively small, subset of terminal pairs, which may allow for substantially cheaper circuit layouts. In this scenario, GMMN appears to be a more realistic model than MMN.

The currently best known approximation algorithms for 2D-MMN have ratio 2; for example, the $O(n \log n)$-time algorithm of Guo et al. [5]. The complexity of 2D-MMN was settled only recently by Chin et al. [2]; they proved the problem NP-hard. It is not known whether 2D-MMN is APX-hard.

Recently, there has been an increased interest in (G)MMN for higher dimensions. Muñoz et al. [7] proved that 3D-MMN is NP-hard to approximate within a factor of 1.00002. They also gave a constant-factor approximation algorithm for a, rather restricted, special case of 3D-MMN. Das et al. described the first approximation algorithm for MMN in arbitrary, fixed dimension with a ratio of $O(n^\varepsilon)$ for any $\varepsilon > 0$ [4].

GMMN was defined by Chepoi et al. [1] who asked whether 2D-GMMN admits an $O(1)$-approximation.

Apart from the formulation of this open problem, only special cases of GMMN such as MMN have been considered so far.

Another special case of GMMN that has received significant attention in the past is the *rectilinear Steiner arborescence problem* (RSA). Here, we are given $n$ terminals lying in the first quadrant and the goal is to find a minimum-length rectilinear network that M-connects every terminal to the origin $o$. Hence, RSA is the special case of GMMN where $o$ is considered a (new) terminal and the set of terminal pairs contains, for each terminal $t \neq o$, only the pair $(o, t)$. RSA is NP-hard [9] even in dimension $d = 2$. Rao et al. [8] gave a 2-approximation algorithm for 2D-RSA. They also provided a conceptually simpler $O(\log n)$-approximation algorithm based on rectilinear Steiner trees. That algorithm generalizes quite easily to dimensions $d > 2$ (as we show in the long version of this paper [3]). Lu and Ruan [6] described a polynomial-time approximation schemes (PTAS) for 2D-RSA based on Arora's technique.

Our contribution is two-fold. First, we provide an $O(\log^{d+1} n)$-approximation algorithm for GMMN (and, hence, MMN) in $d$ dimensions. For the sake of simplicity, we first present our approach in 2D (see Section 2). Second, we provide an improved and technically more involved $O(\log n)$-approximation for the special case of 2D-GMMN; see Section 3.

## 2 Polylogarithmic Approximation

We present an $O(\log^2 n)$-approximation algorithm for 2D-GMMN and prove the following theorem.

**Theorem 1** *2D-GMMN admits an $O(\log^2 n)$-approximation algorithm running in $O(n \log^3 n)$ time.*

Our algorithm consists of a *main algorithm* that recursively subdivides the input instance into instances of so-called *$x$-separated* GMMN; see Section 2.1. We prove that the instances of $x$-separated GMMN can be solved independently by paying a factor of $O(\log n)$ in the overall approximation ratio. Then we solve each $x$-separated GMMN instance within factor $O(\log n)$; see Section 2.2. This yields an overall approximation ratio of $O(\log^2 n)$. Our presentation follows this natural top-down approach; as a consequence, we will make some forward references to results that we prove later.

### 2.1 Main Algorithm

Our algorithm is based on divide and conquer. Let $R$ be the set of terminal pairs that are to be M-connected. We identify each terminal pair with its bounding box, that is, the smallest axis-aligned rectangle that contains both terminals. As a consequence of this, we consider $R$ a set of rectangles. Let $m_x$ be the median in the multiset of the $x$-coordinates of terminals. We identify $m_x$ with the vertical line at $x = m_x$.

Now we partition $R$ into three subsets $R_{\text{left}}, R_{\text{mid}}$, and $R_{\text{right}}$. $R_{\text{left}}$ consists of all rectangles that lie *completely* to the left of the vertical line $m_x$. Similarly, $R_{\text{right}}$ consists of all rectangle that lie *completely* to the right of $m_x$. $R_{\text{mid}}$ consists of all rectangles that intersect $m_x$.

We consider the sets $R_{\text{left}}$, $R_{\text{mid}}$, and $R_{\text{right}}$ as separate instances of GMMN. We apply the main algorithm recursively to $R_{\text{left}}$ to get a rectilinear network that M-connects terminal pairs in $R_{\text{left}}$ and do the same for $R_{\text{right}}$.

It remains to M-connect the pairs in $R_{\text{mid}}$. We call a GMMN instance (such as $R_{\text{mid}}$) *$x$-separated* if there is a vertical line (in our case $m_x$) that intersects every rectangle. We exploit this property to design a simple $O(\log n)$-approximation algorithm for $x$-separated GMMN; see Section 2.2. Later, in Section 3, we improve upon this and describe an $O(1)$-approximation algorithm for $x$-separated GMMN when terminals lie in 2D.

In the following lemma we analyze the performance of the main algorithm, in terms of $\rho_x(n)$, our approximation ratio for $x$-separated instances with $n$ terminal pairs.

**Lemma 2** *If $x$-separated 2D-GMMN admits a $\rho_x(n)$-approximation, 2D-GMMN admits a $(\rho_x(n) \cdot \log n)$-approximation.*

**Proof.** Let $\rho(n)$ denote the main algorithm's worst-case approximation ratio for instances with $n$ terminal pairs. Now assume that our input instance $R$ is a worst case. More precisely, the cost of the solution of our algorithm *equals* $\rho(n) \cdot \text{OPT}$, where OPT denotes the cost of an optimum solution $N^{\text{opt}}$ to $R$. Let $N^{\text{opt}}_{\text{left}}$ and $N^{\text{opt}}_{\text{right}}$ be the parts of $N^{\text{opt}}$ to the left and to the right of $m_x$, respectively. (We split horizontal segments that cross $m_x$ and ignore vertical segments on $m_x$.)

Due to the choice of $m_x$, at most $n$ terminals lie to the left of $m_x$. Therefore, $R_{\text{left}}$ contains at most $n/2$ terminal pairs. Since $N^{\text{opt}}_{\text{left}}$ is a feasible solution to $R_{\text{left}}$, we conclude that the cost of the solution to $R_{\text{left}}$ computed by our algorithm is bounded by $\rho(n/2) \cdot \|N^{\text{opt}}_{\text{left}}\|$, where $\|\cdot\|$ measures the length of a network. Analogously, the cost of the solution computed for $R_{\text{right}}$ is bounded by $\rho(n/2) \cdot \|N^{\text{opt}}_{\text{right}}\|$. Since $N^{\text{opt}}$ is also a feasible solution to the $x$-separated instance $R_{\text{mid}}$, we can compute a solution of cost $\rho_x(n) \cdot \text{OPT}$ for $R_{\text{mid}}$.

Therefore, we can bound the total cost of our algorithm's solution $N$ to $R$ by

$$\|N\| \leq \rho(n/2) \cdot (\|N^{\text{opt}}_{\text{left}}\| + \|N^{\text{opt}}_{\text{right}}\|) + \rho_x(n) \cdot \text{OPT}\,.$$

Note that this inequality does not necessarily hold if $R$ is *not* a worst case since then $\rho(n)\cdot\mathrm{OPT} > \|N\|$. The networks $N^{\mathrm{opt}}_{\mathrm{left}}$ and $N^{\mathrm{opt}}_{\mathrm{right}}$ are separated by line $m_x$, hence they are edge disjoint and $\|N^{\mathrm{opt}}_{\mathrm{left}}\| + \|N^{\mathrm{opt}}_{\mathrm{right}}\| \le \mathrm{OPT}$. This yields the recurrence $\rho(n) \le \rho(n/2) + \rho_x(n)$, which resolves to $\rho(n) = \log n \cdot \rho_x(n)$. ☐

Lemma 2 together with the results of Section 2.2 allow us to prove Theorem 1.

**Proof.** By Lemma 2, our main algorithm has performance $\rho_x(n)\cdot\log n$, where $\rho_x(n)$ denotes the ratio of an approximation algorithm for $x$-separated 2D-GMMN. In Lemma 3 (Section 2.2), we will show that there is an algorithm for $x$-separated 2D-GMMN with ratio $\rho_x(n) = O(\log n)$. Thus overall, the main algorithm yields an $O(\log^2 n)$-approximation for 2D-GMMN. See the long version [3] for the running time analysis. ☐

## 2.2 Approximating $x$-Separated Instances

We describe a simple algorithm for approximating $x$-separated 2D-GMMN with a ratio of $O(\log n)$. Let $R$ be an $x$-separated instance, that is, all rectangles in $R$ intersect a common vertical line.

The algorithm works as follows. Analogously to the main algorithm we subdivide the $x$-separated input instance, but this time using the line $y = m_y$, where $m_y$ is the median of the multiset of $y$-coordinates of terminals in $R$. This yields sets $R_{\mathrm{top}}$, $R'_{\mathrm{mid}}$, and $R_{\mathrm{bottom}}$, defined analogously to the sets $R_{\mathrm{left}}$, $R_{\mathrm{mid}}$, and $R_{\mathrm{right}}$ of the main algorithm, using $m_y$ instead of $m_x$. We apply our $x$-separated algorithm to $R_{\mathrm{top}}$ and then to $R_{\mathrm{bottom}}$ to solve them recursively. The instance $R'_{\mathrm{mid}}$ is a *$y$-separated* sub-instance with all its rectangles intersecting the line $m_y$. Moreover, $R'_{\mathrm{mid}}$ (as a subset of $R$) is already $x$-separated, thus we call $R'_{\mathrm{mid}}$ an *$xy$-separated* instance. In Section 2.3, we give a specialized algorithm to approximate $xy$-separated instances within a constant factor. Assuming this for now, we show (in the long version [3]), analogously to Lemma 2, the following.

**Lemma 3** *$x$-separated 2D-GMMN admits an $O(\log n)$-approximation.*

## 2.3 Approximating $xy$-Separated Instances

It remains to show that $xy$-separated GMMN can be approximated within a constant ratio. Let $R$ be an instance of $xy$-separated GMMN. We assume, w.l.o.g., that it is the $x$- and the $y$-axes that intersect all rectangles in $R$, that is, all rectangles contain the origin $o$. To solve $R$, we compute an RSA network that $M$-connects the set of terminals in $R$ to $o$. We use a 2-approximation algorithm for RSA, for example, the one of Rao et al. [8]. This yields the following.

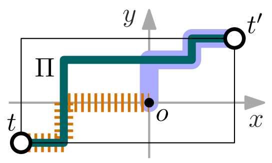


**Figure 2:** Network $N$ connects $t$ to $o$ (dashed path) and $t'$ to $o$ (solid path).

**Lemma 4** *$xy$-separated 2D-GMMN admits a constant-factor approximation.*

**Proof.** Let $T$ be the set of terminals in the $xy$-separated GMMN instance $R$ and $N'$ be an RSA network M-connecting $T$ to $o$, which we compute using the 2-approximation for RSA.

We first claim that $N'$ is a feasible GMMN solution for $R$. To see this, note that $N'$ contains, for every terminal pair $(t,t')\in R$, an M-path $\pi$ from $t$ to $o$ and an M-path $\pi'$ from $o$ to $t'$. Concatenating $\pi$ and $\pi'$ yields an M-path from $t$ to $t'$ as the bounding box of $(t,t')$ contains $o$.

Let OPT denote the cost of an optimal GMMN solution for $R$. We claim there is a solution $N$ for the RSA instance of M-connecting $T$ to $o$ of cost $O(\mathrm{OPT})$.

Let $N^{\mathrm{opt}}$ be an optimum solution to $R$. Let $N$ be the union of $N^{\mathrm{opt}}$ and the projections of $N^{\mathrm{opt}}$ to the $x$-axis and to the $y$-axis. The total length of $N$ is $\|N\| \le 2\cdot\mathrm{OPT} = O(\mathrm{OPT})$ since every line segment of $N^{\mathrm{opt}}$ is projected either to the $x$-axis or to the $y$-axis but not to both. The crucial fact about $N$ is that this network contains, for every terminal $t$ in $R$, an M-path from $t$ to the *origin* $o$. In other words, $N$ is a feasible solution to the RSA instance of M-connecting $T$ to $o$.

To see this, consider an arbitrary terminal pair $(t,t')\in R$. Let $\Pi$ be an M-path connecting $t$ and $t'$ in $N^{\mathrm{opt}}$; see Fig. 2. Note that, since the bounding box of $(t,t')$ contains $o$, $\Pi$ intersects both $x$- and $y$-axis. To obtain an M-path from $t$ to $o$, we follow $\Pi$ from $t$ to $t'$ until $\Pi$ crosses one of the axes. From that point on, we follow the *projection* of $\Pi$ onto this axis. We reach $o$ when $\Pi$ crosses the other axis; see the dotted path in Fig. 2. Analogously, we obtain an M-path from $t'$ to $o$.

Finally, as there is a feasible RSA solution $N$ for terminals $T$ of cost $O(\mathrm{OPT})$, the RSA solution $N'$ that we compute costs at most $2\|N\| = O(\mathrm{OPT})$. ☐

Our algorithm for $d$ dimensions is a generalization of the algorithm for 2D-GMMN. For each dimension, we loose a $(\log n)$-factor in the approximation ratio. The final problem instances can be solved using an adaptation to $d$ dimensions of the $O(\log n)$-approximation algorithm of Rao et al. [8] for 2D-RSA; see the long version [3].

**Theorem 5** *In any fixed dimension $d$, GMMN admits an $O(\log^{d+1} n)$-approximation algorithm running in $O(n^2 \log^{d+1} n)$ time.*

## 3 Improved Algorithm for Two Dimensions

In this section, we show that 2D-GMMN admits an $O(\log n)$-approximation, which improves upon the $O(\log^2 n)$-result of Section 2. To this end, we develop a $(6+\varepsilon)$-approximation algorithm for $x$-separated 2D-GMMN, for any $\varepsilon > 0$; see Lemma 8. Together with Lemma 2, we obtain the following.

**Theorem 6** *For any $\varepsilon > 0$, 2D-GMMN admits a $((6+\varepsilon)\cdot \log n)$-approximation algorithm running in $O(n^{1/\varepsilon} \log^2 n)$ time.*

Our constant-factor approximation for $x$-separated 2D-GMMN and its analysis are technically more involved. Therefore, we only sketch the underlying approach; see the long version [3] for details.

Let $R$ be the set of terminal pairs of an $x$-separated instance of 2D-GMMN. We assume, w.l.o.g., that each terminal pair $(l, r) \in R$ is separated by the $y$-axis, such that $x(l) < 0 \leq x(r)$. Let $N^{\text{opt}}$ be an optimum solution to $R$. Let $\text{OPT}_{\text{ver}}$ and $\text{OPT}_{\text{hor}}$ be the total costs of the vertical and horizontal segments in $N^{\text{opt}}$, respectively. Hence, $\text{OPT} = \text{OPT}_{\text{ver}} + \text{OPT}_{\text{hor}}$.

The algorithm consists of two stages: a *stabbing* stage and a *connection* stage. In the stabbing stage, we compute a set $S$ of horizontal line segments such that each rectangle in $R$ is completely *stabbed* by some line segment in $S$. More precisely, for each rectangle $r$ there is a horizontal line segment $h \in S$ such that the intersection of $r$ and $h$ equals the intersection of $r$ with the supporting line of $h$. We can show the following [3].

**Lemma 7** *Given a set $R$ of rectangles intersecting the $y$-axis, we can compute a set of horizontal line segments of cost at most $4 \cdot \text{OPT}_{\text{hor}}$ that stabs $R$.*

The *connection* stage M-connects the terminals to the $y$-axis so that the resulting network, along with the stabbing $S$, forms a feasible solution to $R$ of cost $O(\text{OPT})$. To this end, we assume that the union of the rectangles in $R$ is connected. Otherwise we apply our algorithm separately to each subset of $R$ that induces a connected component of $\bigcup R$. Let $I$ be the line segment that is the intersection of the $y$-axis with $\bigcup R$. Let $\text{top}(I)$ and $\text{bot}(I)$ be the top and bottom endpoints of $I$, respectively. Let $L \subseteq T$ be the set containing every terminal $t$ with $(t, t') \in R$ and $y(t) \leq y(t')$ for some $t' \in T$. Symmetrically, let $H \subseteq T$ be the set containing every terminal $t$ with $(t, t') \in R$ and $y(t) > y(t')$ for some $t' \in T$. Note that, in general, $L$ and $H$ are not disjoint.

Using a PTAS for 2D-RSA [6], we compute a near-optimal RSA network $A_{\text{up}}$ connecting the terminals in $L$ to $\text{top}(I)$ and a near-optimal RSA network $A_{\text{down}}$ connecting the terminals in $H$ to $\text{bot}(I)$. Then we return the network $N = A_{\text{up}} \cup A_{\text{down}} \cup S$, where $S$ is the stabbing computed by the stabbing stage.

In the long version [3], we show that the resulting network is a feasible solution to $R$, with cost at most constant times OPT.

**Lemma 8** *$x$-separated 2D-GMMN admits, for any $\varepsilon > 0$, a $(6+\varepsilon)$-approximation.*

## References

[1] V. Chepoi, K. Nouioua, and Y. Vaxès. A rounding algorithm for approximating minimum Manhattan networks. *Theor. Comput. Sci.*, 390(1):56–69, 2008.

[2] F. Chin, Z. Guo, and H. Sun. Minimum Manhattan network is NP-complete. *Discrete Comput. Geom.*, 45:701–722, 2011.

[3] A. Das, K. Fleszar, S. G. Kobourov, J. Spoerhase, S. Veeramoni, and A. Wolff. Polylogarithmic approximation for generalized minimum Manhattan networks. Arxiv report, Apr. 2012. Available at http://arxiv.org/abs/1203.6481.

[4] A. Das, E. R. Gansner, M. Kaufmann, S. Kobourov, J. Spoerhase, and A. Wolff. Approximating minimum Manhattan networks in higher dimensions. In *Proc. 19th Annu. Europ. Symp. on Algorithms (ESA'11)*, volume 6942 of *LNCS*, pages 49–60. Springer, 2011. To appear in *Algorithmica*.

[5] Z. Guo, H. Sun, and H. Zhu. Greedy construction of 2-approximate minimum Manhattan networks. *Int. J. Comput. Geom. Appl.*, 21(3):331–350, 2011.

[6] B. Lu and L. Ruan. Polynomial time approximation scheme for the rectilinear Steiner arborescence problem. *J. Comb. Optim.*, 4(3):357–363, 2000.

[7] X. Muñoz, S. Seibert, and W. Unger. The minimal Manhattan network problem in three dimensions. In *Proc. 3rd Int. Workshop Algorithms Comput. (WALCOM'09)*, volume 5431 of *LNCS*, pages 369–380. Springer, 2009.

[8] S. Rao, P. Sadayappan, F. Hwang, and P. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7:277–288, 1992.

[9] W. Shi and C. Su. The rectilinear Steiner arborescence problem is NP-complete. *SIAM J. Comput.*, 35(3):729–740.

# Distributed Computational Geometry and Multi-Robot Systems: Twins Separated at Birth?

James McLurkin*

**Abstract**

This is a talk in three parts. We start with an overview of robotics in general, and why multi-robot systems are the next frontier of robotics research. We focus on the opportunities and challenges presented by large populations; they enable simultaneous coverage of large areas, highly parallel operations, and other novel solutions, but require distributed algorithms for sensing, computation, communication, and actuation. Distributed computational geometry will be required to address these problems

We present an overview of our work with multi-robot systems, including distributed algorithms for robot recovery, angular coordinate systems, and massive manipulation. Our systems are best modeled as geometric graphs embedded in the plane, with various connectivity constraints and self-mobile vertices. Our low-cost robot platform enables this work, and typifies the hardware (read: algorithmic) constraints of large populations.

I conclude with a call for action from the computational geometry community. There is a need for more discussion between our communities on interesting new problems, and we all stand to gain from knowledge of each other's fields. once. I present several engineering problems that require algorithmic solutions, and algorithmic problems that are solved with proper engineering.

*Department of Computer Science, Rice University, USA. jmclurkin@rice.edu

# Topologically Safe Curved Schematization[*]

Arthur van Goethem[†] Herman Haverkort[†] Wouter Meulemans[†]
Andreas Reimer[‡] Bettina Speckmann[†]

## Abstract

Traditionally schematized maps make extensive use of curves. However, automated methods for schematization are mostly restricted to straight lines. We present a generic framework for topology-preserving curved schematization that allows a choice of quality measures and curve types. Our fully-automated approach does not need critical points or salient features. We illustrate our framework with Bézier curves and circular arcs.

## 1 Introduction

A schematized map uses shapes of low complexity to represent geographic objects. So far, most research on automated schematization has concentrated on straight line segments with restrictions on the admissible directions. Many manually produced schematized maps, however, make use of curves [12]. Curves have greater expressive power than line segments: several line segments can often be replaced by a single, low-degree curve. This can make it easier to interpret maps (see Fig. 1).

When using curves in cartography we need topologically sound results. Generalization and other geoprocessing operations are often implemented in recursive processes that repeatedly make expensive checks for topology violations. Removing detected intersections is a challenge as well, triggering more checks for curve-curve intersections. To the best of our knowledge, our framework is the first that ensures topology preservation for curved schematization operations while avoiding repeated checks for intersections.

**Contribution.** We describe a generic framework for computing curved schematizations of simple polygons that maintain the topology of the input. Our framework segments the input polygon at a subset of its vertices and fits curves to the resulting polygonal *chains*, which consist of one or more (consecutive) edges of the input polygon. To ensure that the result of this operation is topologically correct, we use the Voronoi cell of each chain and constrain the curves to remain within these cells.

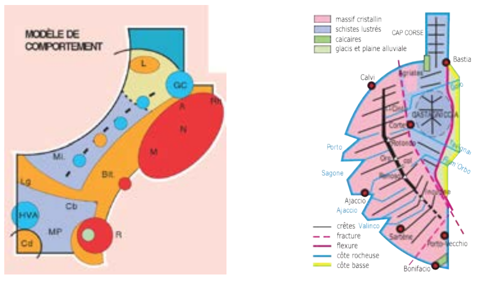


Figure 1: Languedoc-Roussillon [1], Corsica [2].

To use the framework two components need to be specified. The first is a method to fit a simple curve to a polygonal chain. The second is a distance measure which expresses how well the curve fits. There are two requirements for a simple curve fit. First, the curve must start and end at the endpoints of the corresponding chain. Second, the curve may not have self-intersections. The distance measure can be any function that assigns a non-negative value to the combination of a curve and a chain. A low value should indicate a high quality of fit. Straight lines are also curves and could be used in our framework, resulting in "classic" simplification.

The quality of the schematization depends on the segmentation. We formulate the problem of finding the best segmentation as an optimization problem (Section 2). Hence we do not need critical points or salient features to be specified by the user. To illustrate our framework we use Bézier curves (Section 3) and area-preserving circular arcs (Section 4) to generate curved schematizations of territorial outlines.

**Related work.** Automated simplification and smoothing with straight lines have received significant attention over the years. Mustafa *et al.*[11] use Voronoi cells for topologically safe simplification and heuristics to speed up the process. Our framework with straight lines and the directed Hausdorff distance yields a similar setting. Van der Poorten and Jones [18] use constrained Delaunay triangulations to find and simplify features in a topologically safe way.

A variety of methods fit a (cubic) Bézier curve to

[*]A. van Goethem, B. Speckmann, and W. Meulemans are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.102 (AvG) and no. 639.022.707 (BS and WM).

[†]Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands, a.i.v.goethem@tue.nl, cs.herman@haverkort.net, w.meulemans@tue.nl, and speckman@win.tue.nl

[‡]Institute of Geography, University of Heidelberg, Germany, andreas.reimer@geog.uni-heidelberg.de

a polygonal line [9, 15, 16]. However, these methods often do not avoid intersections nor is it obvious how to adapt the fitting process to avoid intersections when fitting multiple curves. Schneider [15] applies a heuristic similar to Douglas-Peucker to fit multiple curves, in essence using critical points. Shao and Zhou [16] also use critical point detection before fitting Bézier curves. B-splines have been applied to approximate polygons [7, 14]. Intersections of different splines, however, still need to be checked and resolved separately. Existing work on automated schematization of shapes is sparse. So far it has concentrated on straight-line drawings.

Drysdale *et al.* [5] present an algorithm to compute an approximation with a minimal number of circular arcs for a given tolerance region and a set of "gates". However, requirements on these gates prevent a significant complexity reduction. Heimlich and Held [8] describe how to generate smooth approximations with circular arcs using tolerance bands. Their framework allows other curves, but cannot guarantee optimal results. Also, smooth approximations are not always suitable for schematization. Neither of these methods can incorporate area-preservation constraints.

## 2 Computing optimal segmentations

As stated earlier, the quality of the final schematization depends on the segmentation. In this section we describe how to efficiently use dynamic programming to find an optimal segmentation.

We first compute the edge-based Voronoi diagram of the input polygon. Every Voronoi cell is a simple polygon (potentially unbounded and with parabolic parts) without holes. The Voronoi cell of a chain is the union of the Voronoi cells of its edges. To guarantee topological correctness, we ensure that the union of cells remains a simple polygon without holes (see Fig. 2).

To be able to quickly change the parameters of the schematization, we pre-compute a two-dimensional table $T$. Let $P[i, j]$ denote the chain of the input polygon from vertex $i$ to vertex $j$. For $i = j$, this is the entire polygon. Each entry $T[i, j]$ in the table will contain the curve fitted to $P[i, j]$ and a value that indicates the quality of the fit. For each entry of $T$

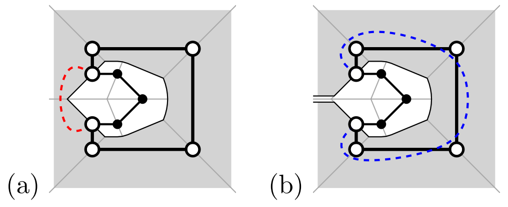


Figure 2: (a) Dashed curve lies in the cell but violates topology. (b) Union considers the pocket.

we compute the associated cell and fit a curve to the chain. If the curve lies within the cell, we compute the quality of the fit and store it. If not, we disallow the use of this curve by setting $T[i, j] = \infty$ to ensure topological correctness.

Let $F(v)$ represent the time required to fit a curve to a chain with $v$ vertices, check whether the curve lies within the chain's cell, and compute the quality of the fit. Let $n$ denote the number of vertices of the polygon. Since the table has $n^2$ entries, it takes $O(n^2F(n))$ time to compute the lookup table.

To obtain a schematization from the table $T$, we select the curves representing a collection of consecutive chains which together constitute the complete polygon. We distinguish two variants, as described below. We first explain, for each of these variants, how to compute the value of the optimal solution, assuming that we start at vertex 1. In the presentation of both variants we assume that $T[i, j]$ denotes only the quality of fit of the curve fitted to $P[i, j]$. For simplicity of explanation we define $T[j, n + 1] = T[j, 1]$.

**Min-# problem.** For the min-# problem, we wish to minimize the number of curves that are used, if the distance of each curve has to be at most $\epsilon$. The minimum number of curves needed can be determined by computing the values of the following expression for $i = 1$ up to $i = n + 1$:

$$OPT[i] = \begin{cases} 0, & \text{if } i = 1 \\ \min_{1 \le j < i \text{ and } T[j,i] \le \epsilon} OPT[j] + 1, & \text{if } i > 1 \end{cases}$$

The computation takes $O(n^2)$ time and the end result is $OPT[n + 1]$. A solution is guaranteed to exist if the curve representing a single line segment is the line segment itself and has distance zero.

**Min-$\epsilon$ problem.** For the min-$\epsilon$ problem, we wish to minimize the maximum distance of the selected curves, while using at most $K$ curves. The minimum achievable distance can be determined by computing the values of the following expression for $i = 1$ up to $i = n + 1$ and for $k = 1$ up to $k = K$:

$$OPT[i, k] =$$

$$\begin{cases} T[1, i], & \text{if } k = 1 \\ \min \left( \begin{array}{c} OPT[i, k-1], \\ \min_{1 \le j < i} (\ \max\left(T[j, i], OPT[j, k-1]\right)\ ) \end{array} \right), & \text{if } k > 1 \end{cases}$$

The computation takes $O(n^2K)$ time and the end result is $OPT[n + 1, K]$. A solution is guaranteed to exist if the curve fitting method can fit a curve to a chain that starts and ends at the same point (i.e. the entire polygon with a given start vertex).

To compute the best solution, we repeat the above computations for each vertex as starting vertex. We obtain the actual schematization by keeping track of the choices made during the computation of $OPT$.

The initialized table $OPT[i,k]$ for the min-$\epsilon$ problem is independent of the scale or complexity parameter. Hence, one table can be used to create multiple schematizations with different parameter values. As a byproduct of querying for $k$ curves, we obtain all results using less than $k$ curves. So if we run the query for $k = n$ once, we obtain all possible schematizations and we can store these instead of the table. A query for different values then becomes a simple lookup. We can also handle queries for the min-# problem by computing the maximum distance used in each of the solutions and performing a binary search.

## 3 Cubic Bézier curves

In this section we show how to use our framework with Bézier curves. We describe a method to fit a Bézier curve to a chain and specify a distance measure.

**Distance measure.** Assume that we are given a Bézier curve $C$ and a chain $S$ with $m$ vertices. In contrast to fitting a curve to a set of data points, we want to fit a curve to a chain. Therefore, we base our distance measure on a dense sampling of $S$. These samples are regularly spaced in the parameter space of $C$. Each of these samples has a corresponding point along $S$ as parameterized by length.

We desire long curves that approximate the data well, since such curves can frequently be observed in manually drawn curved schematizations. The distance measure of $C$ is, therefore, a weighted average of the squared distance between corresponding points, divided by the length of $S$. Formally, our distance measure between curve $C$ and chain $S$ is:

$$\frac{\sum_{i=0}^{2m} \|C(i/2m) - S(i/2m)\|^2 * (i-m)^2}{length(S) * \sum_{i=0}^{2m}(i-m)^2}\text{ , where}$$

$length(S)$ : Euclidean length of $S$;

$C(t), S(t)$ : position on $C$ or $S$, as parameterized by $t$;

$\|u - v\|$ : Euclidean distance between points $u$ and $v$.

When curves do not match the local contour at their endpoints, this may create visually salient points not present in the input data (see Fig. 3a). To avoid such visual artifacts we use a weighted average that assigns a high weight to samples near the endpoints and a low weight to the midsection. As a result, features along the midsections of curves may disappear (see Fig. 3b). We deem the disappearance of features less disturbing than the appearance of features that do not exist.

To exclude self-intersecting curves we define the distance measure of a self-intersecting curve to be infinity. Self-intersection of a cubic Bézier curve is tested by examining the control points [17].

**Fitting a cubic Bézier curve.** For fixed tangents at the endpoints, Schneider [15] shows how to algebraically compute the Bézier curve optimizing the least-squares measure in linear time. A similar method can be used to optimize a weighted measure. We direct the initial tangents towards the furthest vertex on either side of the straight line connecting the endpoints (see [9]). Based on these tangents, the algorithm by Schneider optimizes the distance of the second and third control point. We subsequently fix the distance between control points $P_0$ and $P_1$, and between $P_2$ and $P_3$ and optimize the tangents for either side algebraically. If desired, this process of subsequently optimizing the distance and tangents can be repeated. In our experiments, a few iterations were sufficient to obtain a stable solution.

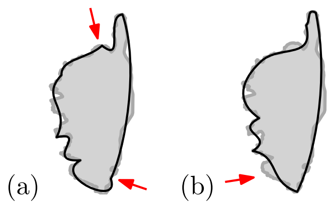


Figure 3: (a) Regular squared error. (b) Weighted squared error.

Topological validity is tested by checking for intersections between a polygonal approximation of the Voronoi cell and the convex hull of the control points. If required the curve can be subdivided up to a constant number of times using De Casteljau's algorithm [4] to obtain a tighter fit around the curve.

**Algorithmic complexity.** Fitting a Bézier curve takes $O(n)$ time for a single iteration. We repeat the process until the solution is stable, which takes 3 to 6 iterations in our experiments. Thus, assuming a constant upper bound on the number of iterations, the total preprocessing time within our framework is $O(n^3)$.

**Results.** Fig. 4 and 5 show results of our framework. Each depicts a schematization of different complexity on top of the input polygon. Even a small number of curves result in recognizable and pleasing shapes.

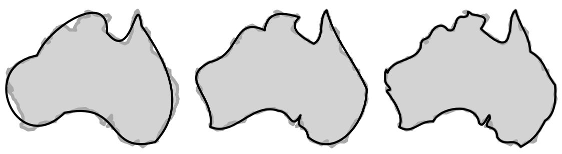


Figure 4: Australia with 5, 10, and 20 curves.

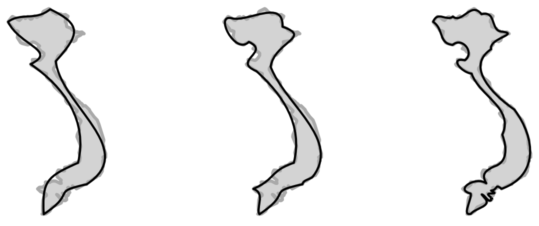


Figure 5: Vietnam with 7, 10, and 20 curves.

## 4 Circular arcs

Inspired by recent work on area-preserving schematization [3] and circular-arc graph drawing [6], we use our framework for area-preserving circular-arc schematizations. Given a chain, the circular arc with the same start and end vertex that preserves the area on each side of it is unique. It is computed by fitting a circular segment with a signed area equal to that of the chain. This requires numerical methods. As distance measure we use the continuous Fréchet distance, extended for curves [13]. We find that $F(n) = O(n \log n)$ and thus, precomputing the lookup table requires $O(n^3 \log n)$ time.

**Results.** Fig. 6 and 7 show some results using area-preserving circular arcs. A very low number of arcs results in a very stylized shape. The results typically retain some features, but are hard to recognize without context. A few more arcs give a stylized or playful appearance and make the shape more recognizable.

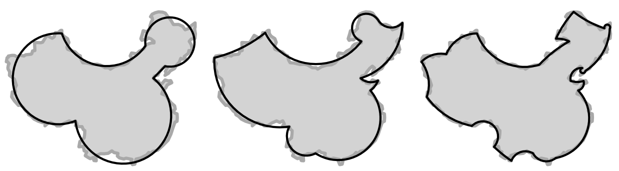


Figure 6: China with 5, 10, and 20 arcs.

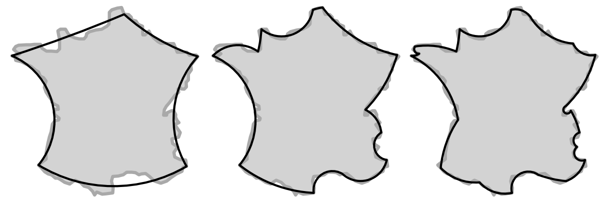


Figure 7: France with 5, 12, and 20 arcs.

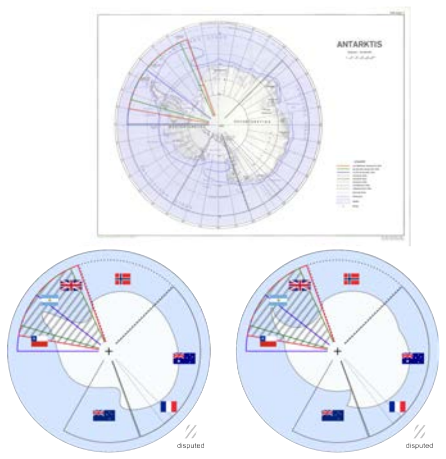


Figure 8: (Top) Downsized map of Antarctica [10]. (Bottom) Diagrams using results of our framework.

### References

[1] R. Brunet. La population du Languedoc-Roussillon en 1990 et la croissance récente. *MappeMonde*, 91(1):34–36, 1991.

[2] R. Brunet. La Corse, région d'Europe. *Mappemonde*, 76(4):1–16, 2004.

[3] K. Buchin, W. Meulemans, and B. Speckmann. A new method for subdivision simplification with applications to urban-area generalization. In *Proc 19th ACM GIS*, pages 261–270, 2011.

[4] P. De Casteljau. Outillages méthodes calcul. *Technical report, A. Citroën*, 1959.

[5] R. Drysdale, G. Rote, and A. Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs and biarcs. *CGTA*, 41(1-2):31–47, 2008.

[6] C. Duncan, D. Eppstein, M. Goodrich, S. Kobourov, and M. Löffler. Planar and poly-arc Lombardi drawings. *Graph Drawing (LNCS 7034)*, pages 308–319, 2012.

[7] E. Guilbert and H. Lin. Isobathymetric line simplification with conflict removal based on a B-spline snake model. *Marine Geodesy*, 30(1-2):169–195, 2007.

[8] M. Heimlich and M. Held. Biarc approximation, simplification and smoothing of polygonal curves by means of Voronoi-based tolerance bands. *IJCGA*, 18(3):221–250, 2008.

[9] A. Masood and S. Ejaz. An efficient algorithm for robust curve fitting using cubic Bezier curves. In *Proc 6th ICIC (LNAI 6216)*, pages 255–262, 2010.

[10] Militärgeographisches Amt (ed.). Antarktis 1:30.000.000. *Atlas der militärischen landeskunde DMG-1982*, 1982.

[11] N. Mustafa, E. Koutsofios, S. Krishnan, and S. Venkatasubramanian. Hardware-assisted view-dependent map simplification. In *Proc 17th SCG*, pages 50–59, 2001.

[12] A. W. Reimer. Understanding chorematic diagrams: towards a taxonomy. *The Cartographic Journal*, 47(4):330–350, 2010.

[13] G. Rote. Computing the Fréchet distance between piecewise smooth curves. *Computational Geometry: Theory and Appl.*, 37(3):162–174, 2007.

[14] E. Saux and M. Daniel. Data reduction of polygonal curves using B-splines. *Computer-Aided Design*, 31(8):507–515, 1999.

[15] P. J. Schneider. An algorithm for automatically fitting digitized curves. In *Graphic Gems*, pages 612–626. Academic Press Professional, 1990.

[16] L. Shao and H. Zhou. Curve fitting with Bezier cubics. *Graphical models and image processing*, 58(3):223–232, 1996.

[17] M. Stone and T. DeRose. A geometric characterization of parametric cubic curves. *ACM Trans. on Graph.*, 8(3):147–163, 1989.

[18] P. van der Poorten and C. Jones. Characterisation and generalisation of cartographic lines using Delaunay triangulation. *International Journal of GIS*, 16(8):773–794, 2002.

# Straight Line Triangle Representations[*]

Nieke Aerts[†] Stefan Felsner[†]

## Abstract

Which plane graphs admit a straight line representation such that all faces have the shape of a triangle? We present a characterization based on flat angle assignements, i.e., selections of angles of the graph that have size $\pi$ in the representation. Another characterization is in terms of contact systems of pseudosegments. We use discrete harmonic functions to show that contact systems of pseudosegments that respect certain conditions are stretchable.

The drawback of the characterization is that we are not able to effectively check whether a given graph admits a flat angle assignment that fulfills the conditions. Hence it is still open to decide whether the recognition of graphs that admit straight line triangle representation is polynomially tractable.

## 1 Introduction

Planar graphs and their geometric representations have been studied intensively over the years. Tutte showed that 3-connected planar graphs have convex drawings (rubber band representation) [11]. Koebe has shown that they can be represented as circle contact graphs [9]. They also admit triangle contact representations [4].

In this paper we aim at characterizing the class of planar graphs that admit a straight line representation in which all faces are triangles. Haas et al. present a necessary and sufficient condition for a graph to be a pseudo-triangulation [7], however this condition is not sufficient for a graph to have the desired straight line triangle representation.

The problem has been studied in the dual setting, i.e., in the setting of side contact representations of planar graphs with triangles. Gansner, Hu and Kobourov [6] show that outerplanar graphs, grid graphs and hexagonal grid graphs can be represented as Touching Triangle Graphs (TTG's) and give a linear time algorithm to find the representation. Alam, Fowler and Kobourov [2] consider proper TTG's, i.e., the union of all triangles of the TTG is a triangle and there are no holes. They present a necessary and a slightly weaker sufficient condition for biconnected outerplanar graphs to have a TTG. Kobourov, Mondal and Nishat [8] present construction algorithms for proper TTG's of 3-connected cubic graphs and some grid graphs and a recognition algorithm for proper TTG's.

Here is the formal introduction of the main character for this paper.

**Definition 1** *A plane drawing of a graph such that*

- *all the edges are straight line segments and*
- *all the faces, including the outer face, bound a non-degenerate triangle*

*is called* a straight line triangle representation *(SLTR).*

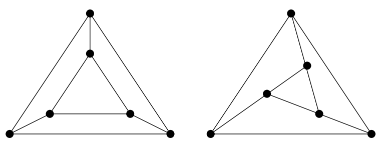


Figure 1: A graph and one of its SLT Representations.

Clearly every straight line drawing of a triangulation is an SLTR. So the class of planar graphs admitting an SLTR is rich. On the other hand, graphs admitting an SLTR can not have a cut vertex. Being well connected, however, is not sufficient as shown e.g. by the cube graph.

To simplify the discussion we assume that the input graph is given with a plane embedding and a selection of three vertices of the outer face that are designated as corner vertices for the outer face. These three vertices are called *suspension vertices.*

The two edges of a degree two vertex that is not a suspension are aligned in every SLTR. Hence, we may replace the vertex and its incident edges by a single edge. Such an operation is called a *vertex reduction.* With vertex reductions we eliminate all the degree two vertices except for degree two vertices that are suspensions.

A plane graph $G$ with suspensions $s_1, s_2, s_3$ is said to be *internally 3-connected* when the addition of a new vertex $v_\infty$ in the outer face, that is made adjacent to the three suspension vertices, yields a 3-connected graph.

**Proposition 2** *If a graph $G$ admits an SLTR with $s_1, s_2, s_3$ as corners of the outer triangle and no vertex reduction is possible, then $G$ is internally 3-connected.*

[*]The full version of this paper can be found online [1]

[†]Institut für Mathematik, Technische Universität Berlin, Germany, {aerts,felsner}@math.tu-berlin.de, Partially supported by DFG grant FE-340/7-2 and ESF EuroGIGA project GraDR

By Proposition 2 we may assume that the graphs we consider are internally 3-connected since any graph that is not internally 3-connected but does admit an SLTR, is a subdivision of an internally 3-connected graph.

In Section 2 we present necessary conditions for the existence of an SLTR in terms of what we call a flat angle assignment. A flat angle assignment that fulfills the conditions induces a partition of the set of edges into a set of pseudosegments. With the aid of discrete harmonic functions we can show that in our case the set of pseudosegments is stretchable. Hence, the necessary conditions are also sufficient. The drawback of the characterization is that we are not aware of an effective way of checking whether a given graph admits a flat angle assignment that fulfills the conditions.

In Section 3 of the full paper, we use our result to give a new, simpler proof of a theorem of de Fraysseix and Ossona de Mendez [3] about stretchable systems of pseudosegments.

## 2 Necessary and Sufficient Conditions

Consider a plane, suspended, internally 3-connected graph $G = (V, E)$. Suppose that $G$ admits an SLTR. This representation induces a set of *flat angles*, i.e., incident pairs $(v, f)$ such that vertex $v$ has an angle of size $\pi$ in the face $f$.

Since $G$ is internally 3-connected every vertex has at most one flat angle. Therefore, the flat angles can be viewed as a partial mapping of vertices to faces. The outer angle of suspension vertices exceeds $\pi$, hence, suspensions have no flat angle. Since each face $f$ (including the outer face) is a triangle, each face has precisely three angles that are not flat. In other words every face $f$ has $|f| - 3$ incident vertices that are assigned to $f$. This motivates the definition:

**Definition 3** *A* flat angle assignment *(FAA) is a mapping from a subset $U$ of the non-suspension vertices to faces such that*
*[$C_v$] Each vertex of $U$ is assigned to exactly one face,*
*[$C_f$] For every face $f$, precisely $|f| - 3$ vertices are assigned to $f$.*

Not every FAA induces an SLTR. An example is given in Figure 2.

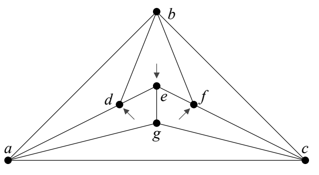


Figure 2: A graph with an FAA given by the arrows, an arrow assigns a vertex to a face. When all assigned angles are stretched, all vertices but $b$ have to be aligned between $a$ and $c$.

Hence, we have to identify another condition. To state this we define: If $H$ is a connected subgraph of the plane graph $G$, then we call the closed walk corresponding to the outer face of $H$ the *outline cycle $\gamma(H)$ of $H$.*

An *outline cycle* of $G$ is a closed walk that can be obtained as outline cycle of some connected subgraph of $G$. Outline cycles may have repeated edges and vertices, see Fig. 3. The interior $\mathsf{int}(\gamma)$ of an outline cycle $\gamma = \gamma(H)$ consists of $H$ together with all vertices, edges and faces of $G$ that are contained in the area enclosed by $\gamma$.

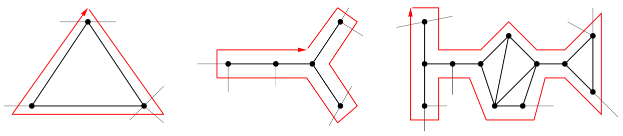


Figure 3: Three examples of outline cycles.

**Proposition 4** *An SLTR obeys the following condition $C_o$:*
*[$C_o$] Every outline cycle that is not the outline cycle of a path, has at least three geometrically convex corners.*

Condition $C_o$ has the disadvantage that it depends on a given SLTR, hence, it is useless for deciding whether a planar graph $G$ admits an SLTR. The following definition allows to replace $C_o$ by a combinatorial condition on an FAA.

**Definition 5** *Given an FAA a vertex $v$ of an outline cycle $\gamma$ is a* combinatorial convex corner *for $\gamma$ if*

- *$v$ is a suspension vertex, or*
- *$v$ is not assigned to a face and there is an edge $e$ incident to $v$ with $e \notin \mathsf{int}(\gamma)$, or*
- *$v$ is assigned to a face $f$, $f \notin \mathsf{int}(\gamma)$ and there exists an edge $e$ incident to $v$ with $e \notin \mathsf{int}(\gamma)$.*

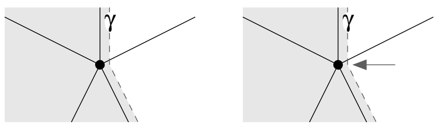


Figure 4: Combinatorially Convex Corners. Left, $v$ is not assigned and has an edge reaching to a vertex outside of $\mathsf{int}(\gamma)$. Right, the arrow represents the assignment of the vertex.

**Proposition 6** *Let $G$ admit an SLTR $\Gamma$, that induces the FAA $\psi$ and let $H$ be a connected subgraph of $G$. If $v$ is a geometrically convex corner of the outline cycle $\gamma(H)$ in $\Gamma$, then $v$ is a combinatorially convex corner of $\gamma(H)$ with respect to $\psi$.*

**Proof.** If $v$ is a suspension vertex it is clearly geometrically and combinatorially convex.

Let $v$ be geometrically convex and suppose that $v$ is not a suspension and not assigned by $\psi$. In this case

$v$ is interior and, with respect to $\gamma$, the outer angle at $v$ exceeds $\pi$. Therefore at least two incident faces of $v$ are outside of $\gamma$. These faces can be chosen to be adjacent, hence, the edge between them is an edge $e$ with $e \notin \mathsf{int}(\gamma)$. This shows that $v$ is combinatorially convex.

Let $v$ be geometrically convex and suppose that $v$ is assigned to $f$ by $\psi$. If $f \in \mathsf{int}(\gamma)$, then the inner angle of $v$ with respect to $\gamma$ is at least $\pi$. This contradicts the fact that $v$ is geometrically convex. Hence $f \notin \mathsf{int}(\gamma)$. If there is no edge $e$ incident to $v$ such that $e \notin \mathsf{int}(\gamma)$, then $v$ has an angle of size $\pi$ with respect to $\gamma$. This again contradicts the fact that $v$ is geometrically convex. Therefore, if $v$ is geometrically convex and assigned to $f$, then $f \notin \mathsf{int}(\gamma)$ and there exists an edge $e$ incident to $v$ such that $e \notin \mathsf{int}(\gamma)$. This shows that $v$ is a combinatorial convex corner for $\gamma$. □

The proposition enables us to replace the condition on geometrically convex corners w.r.t. an SLTR by a condition on combinatorially convex corners w.r.t. an FAA.

[$\text{C}_o^*$] Every outline cycle that is not the outline cycle of a path, has at least three combinatorially convex corners.

From Proposition 4 and Proposition 6 it follows that this condition is necessary for an FAA that belongs to an SLTR. It is shown in Theorem 11 below, that if an FAA obeys $\text{C}_o^*$ then it belongs to an SLTR. In anticipation of this result we say that an FAA obeying $\text{C}_o^*$ is a *good flat angle assignment* and abreviate it as a *GFAA*.

Next we argue that a GFAA induces a contact family of pseudosegments.

**Definition 7** *A contact family of pseudosegments is a family $\{c_i\}_i$ of simple curves $c_i : [0,1] \to \mathbb{R}^2$, with $c(0) \neq c(1)$, such that any two curves $c_i$ and $c_j$ $(i \neq j)$ have at most one point in common. If $c_i$ and $c_j$ have a common point, then this point is an endpoint of (at least) one of them.*

A GFAA $\psi$ on a graph $G$ gives rise to a relation $\rho$ on the edges: Two edges, both incident to $v$ and $f$ are in relation $\rho$ if and only if $v$ is assigned to $f$. The transitive closure of $\rho$ is an equivalence relation.

**Proposition 8** *The equivalence classes of edges of $G$ defined by $\rho$ form a contact family of pseudosegments.*

**Definition 9** *Let $\Sigma$ be a family of pseudosegments and $S$ a subset of $\Sigma$. A point $p$ of a pseudosegment from $S$ is a free point for $S$ if*

1. *$p$ is an endpoint of a pseudosegment in $S$, and*
2. *$p$ is not interior to a pseudosegment in $S$, and*
3. *$p$ is incident to the unbounded region of $S$, and*
4. *$p$ is a suspension or $p$ is incident to a pseudosegment that is not in $S$.*

We now give a relation between a GFAA and a family of pseudosegments that is induced by the GFAA.

**Lemma 10** *Let $\psi$ a GFAA on a plane, internally 3-connected graph $G$. For every subset $S$ of the family of pseudosegments associated with $\psi$, it holds that, if $|S| \geq 2$ then $S$ has at least 3 free points.*

Given an internally 3-connected, plane graph $G$ with a GFAA. To find a corresponding SLTR we aim at representing each of the pseudosegments induced by the FAA as a straight line segment. If this can be done, every assigned vertex will be between its two neighbors that are part of the same pseudosegment. This property can be modeled by requiring that the coordinates $p_v = (x_v, y_v)$ of the vertices of $G$ satisfy a harmonic equation at each assigned vertex.

Indeed if $uv$ and $vw$ are edges belonging to a pseudosegment $s$, then the coordinates satisfy

$$\begin{aligned} x_v &= \lambda_v x_u + (1-\lambda_v) x_w, \\ y_v &= \lambda_v y_u + (1-\lambda_v) y_w. \end{aligned}$$

The parameter $\lambda_v$ is some number stricly between 0 and 1. The theory of harmonic functions applied to (plane) graphs is well explained in [10].

In the SLTR every not assigned vertex is placed in a weighted barycenter of its neighbors. In terms of coordinates this can be written as

$$x_v = \sum_{u \in N(v)} \lambda_{vu} x_u, \qquad y_v = \sum_{u \in N(v)} \lambda_{vu} y_u$$
$$\text{with} \qquad \sum_{u \in N(v)} \lambda_{vu} = 1 \quad \text{and} \quad \lambda_{vu} > 0.$$

These are again harmonic equations. The vertices whose coordinates are not restricted by harmonic equations are called *poles*. In the given case the suspension vertices are the three poles of the harmonic functions for the $x$ and $y$-coordinates. The coordinates for the suspension vertices are the corners of some non-degenerate triangle.

The theory of harmonic funcions [10] implies that for every choice of the parameters $\lambda_v$ and $\lambda_{vu}$ complying with the conditions, the system has a unique solution.

Now we state our main result, it shows that the necessary conditions are also sufficient.

**Theorem 11** *Given an internally 3-connected, plane graph $G$ and $\Sigma$ a family of pseudosegments associated to an FAA, such that each subset $S \subseteq \Sigma$ has three free points or cardinality at most one. The unique solution of the system of equations that arises from $\Sigma$, is an SLTR.*

**Proof.** The proof consists of 7 arguments, which together yield that the drawing induced from the GFAA via a solution of the harmonic system is a non-degenerate, plane drawing. The proof has been inspired by the proof of Colin de Verdière [5] for convex straight line drawings of plane graphs via spring

embeddings. For this abstract we only indicate the steps.
1. *Pseudosegments get Segments.*
2. *Outer Face is Convex.*
3. *No Concave Angles.*
4. *No Degenerate Vertex.* A vertex is degenerate if it is placed on a line, together with at least three of its neighbors. The proof of this property is quite involved, it makes use of the absence of $K_{3,3}$ minors in $G$.
5. *Preservation of the Embedding.* The rotation system at all vertices is preserved.
6. *No Edge Crossings.*
7. *No Edges or Angles of Size* 0. $\square$

Hence, we obtained equivalence between the SLTR, the FAA satisfying $\mathrm{C}_v$, $\mathrm{C}_f$ and $\mathrm{C}_o^*$, and a stretchable system of pseudosegments that arises from this FAA.

## 3 Stretchability of Systems of Pseudosegments

A contact system of pseudosegments is *stretchable* if it is homeomorphic to a contact system of straight line segments.

De Fraysseix and Ossona de Mendez characterized stretchable systems of pseudosegments [3]. Their result is based on the notion of an extremal point.

**Definition 12** *Let $\Sigma$ be a family of pseudosegments and $S$ a subset of $\Sigma$. A point $p$ is an* extremal point *for $S$ if*
1. *$p$ is an endpoint of a pseudosegment in $S$, and*
2. *$p$ is not interior to a pseudosegment in $S$, and*
3. *$p$ is incident to the unbounded region of $S$.*

**Theorem 13 (De Fraysseix & O. de Mendez)** *A contact family $\Sigma$ of pseudosegments is stretchable if and only if each subset $S \subseteq \Sigma$ of pseudosegments with $|S| \geq 2$, has at least 3 extremal points.*

Our notion of a free point (Def. 9) is more restrictive than the notion of extremal points from [3, Section 5.2]. However, in the case of families of pseudosegments that live on a plane graph via an FAA, the two notions coincide. The following reformulation of Theorem 11 replaces free points by extremal points:

**Proposition 14** *Given an internally 3-connected, plane graph $G$ and $\Sigma$ a family of pseudosegments associated to an FAA, such that each subset $S \subseteq \Sigma$ has three extremal points or cardinality at most one. The unique solution of the system of equations that arises from $\Sigma$, is an SLTR.*

## 4 Conclusion and Open Problems

We have given necessary and sufficient conditions for a 3-connected planar graph to have an SLT Representation. Given an FAA and a set of rational parameters $\{\lambda_i\}_i$, the solution of the harmonic system can be computed in strongly polynomial time. Hence, a degenerate solution can be identified in polynomial time. It shows that the FAA is not good.

Checking whether an FAA satisfies $\mathrm{C}_o$ can be done in polynomial time, but a graph may admit different assignments of which only some are good. We are not aware of an effective way of finding a GFAA. We therefore leave the problem: Is the recognition of graphs that have an SLTR (GFAA) in $P$?

Given a 3-connected planar graph and a GFAA, interesting optimization problems arise, e.g. find the set of parameters $\{\lambda_i\}_i$ such that the smallest angle in the graph is maximized, or the set of parameters such that the length of the shortest edge is maximized.

## References

[1] N. Aerts and S. Felsner, *Straight line triangle representations.* `http://page.math.tu-berlin.de/~aerts/pubs/sltr.pdf`.

[2] M. J. Alam, J. Fowler, and S. G. Kobourov, *Outerplanar graphs with proper touching triangle representations.* unpublished manuscript.

[3] H. de Fraysseix and P. O. de Mendez, *Barycentric systems and stretchability*, Discrete Applied Mathematics, 155 (2007), pp. 1079–1095.

[4] H. de Fraysseix, P. O. de Mendez, and P. Rosenstiehl, *On triangle contact graphs*, Combinatorics, Probability & Computing, 3 (1994), pp. 233–246.

[5] Y. C. de Verdière, *Comment rendre géodésique une triangulation d'une surface?*, L'Enseignement Mathématique, 37 (1991), pp. 201–212.

[6] E. R. Gansner, Y. Hu, and S. G. Kobourov, *On touching triangle graphs*, in Graph Drawing, Proceedings GD'10, vol. 6502 of Lecture Notes in Computer Science, 2011, pp. 250–261.

[7] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, and W. Whiteley, *Planar minimally rigid graphs and pseudo-triangulations*, in Proceedings of the nineteenth annual symposium on Computational geometry, SCG '03, New York, NY, USA, 2003, ACM, pp. 154–163.

[8] S. G. Kobourov, D. Mondal, and R. I. Nishat, *Touching triangle representation for 3-connected planar graphs*, in Graph Drawing, Proceedings GD'12, 2012.

[9] P. Koebe, *Kontaktprobleme der konformen abbildung*, Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Kl., 88 (1936), pp. 141–164.

[10] L. Lovász, *Geometric representations of graphs.* `http://www.cs.elte.hu/~lovasz/geomrep.pdf`, Draft version December 11, 2009.

[11] W. T. Tutte, *How to draw a graph*, Proc. of the London Math. Society, 13 (1963), pp. 743–767.

# Reconstructing Polygons from Embedded Straight Skeletons

Therese Biedl* Martin Held† Stefan Huber†

## Abstract

A straight skeleton is a well-known geometric structure, and several algorithms exist to construct the straight skeleton for a given polygon. In this paper, we ask the reverse question: Given the straight skeleton (in form of a tree with a drawing in the plane, but with the exact position of the leaves unspecified), can we reconstruct the polygon? We show that in most cases there exists at most one polygon; in the remaining case there is an infinite number of polygons determined by one angle that can range in an interval. We can find this (set of) polygon(s) in linear time in the Real RAM computer model.

## 1 Introduction

The straight skeleton $\mathcal{S}(P)$ of a polygon $P$ is a well-known geometric data structure. It is defined by offsetting a polygon inwards, thereby moving all edges at constant speed, and tracing the movement of the polygon's vertices. The straight skeleton is always a tree for a polygon without holes. We refer to Huber and Held [3] for an extensive and up-to-date discussion of theory and applications of straight skeletons.

In this paper, we consider the following problem: Given a straight skeleton, can we re-construct its defining polygon? And if so, is the polygon unique (up to offsetting)?

The answer to this problem depends on what we mean by "given a straight skeleton." If the locations of all nodes of the tree are known, then the vertices of the polygon are the leaves of the straight skeleton and the problem is trivial. At the other end of the spectrum, if the straight skeleton is given as an abstract tree, then there always exists a polygon (even a convex polygon) for which this is the straight skeleton [1], and it is not unique. Somewhat in-between is a problem from phylogenetic tree reconstruction, where we are given the edge lengths and the cyclic order of edges at vertices; here partial results are known for trees that are stars or caterpillars [1].

In related work, Liotta and Meijer [4] showed that for any abstract tree $T$, there exists a convex point set $S$ such that the Voronoi diagram of $S$ equals $T$. (Their result actually follows from Dillencourt's result [2] that all outerplanar graphs can be realized as Delaunay triangulations of points in convex position.) The proof in [4] resembles the one in [1]. Questioning whether perhaps their proof could be used to show the results in [1] led us to the following problem:

**Problem 1 (Voronoi-matching problem (VMP))** *Given a Voronoi diagram $T$ of a set of points in convex position, does there exist a polygon $P$ such that its straight skeleton $\mathcal{S}(P)$ coincides with $T$?*

Hence our setup of "given a straight skeleton" is that we are given a drawing of the tree, but the vertices of the polygon may be anywhere on rays towards the leaves at infinity. Our results will not use the fact that the tree came from a Voronoi diagram, and we hence solve the more general problem:

**Problem 2 (Tree-matching problem (TMP))** *Given a planar drawing of a tree $T$ with $n$ leaves on rays towards infinity, does there exist a polygon $P$ with $n$ vertices that lie on the edges incident to the leaves such that $\mathcal{S}(P)$ coincides (inside $P$) with $T$?*

We often call the edges incident to the leaves (i.e., on the infinite rays) the *leaf edges* while all other edges are called *inner edges*.

The next problem is derived from TMP by only considering the leaf edges, and will be a crucial first step towards solving TMP.

**Problem 3 (Bisector-matching problem (BMP))** *Given a set of $n$ non-intersecting rays $b_1, \ldots, b_n$, find a polygon $P$ with vertices $v_1, \ldots, v_n$ such that $v_i$ lies on $b_i$ and the edge-bisector at $v_i$ has the same supporting line as $b_i$, for $1 \leq i \leq n$, with the ray leading towards the outside.*

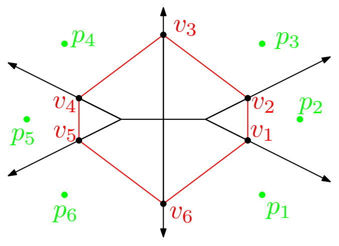


Figure 1: The straight skeleton of the (red) polygon $v_1, \ldots, v_6$ matches the Voronoi diagram of the (green) points $p_1, \ldots, p_6$.

*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.

†Universität Salzburg, Computerwissenschaften, A–5020 Salzburg, Austria. [held,shuber]@cosy.sbg.ac.at

We will make two simplifying assumptions on the input bisectors which hold if the input comes from VMP, see Sec. 2.

**Our results.** In this paper, we first solve BMP, and then use it to solve TMP. In particular, given a drawing of a tree, we show that either there exists a unique polygon for which this is the straight skeleton (up to offsetting the polygon), or there exist infinitely many such polygons. The solution set is described by an interval for one angle at a vertex.

## 2 Notation and Basics

The input to TMP is a tree $T$ that has a planar (crossing-free) drawing in 2D, and that has $n$ leaves (at infinity). Enumerating the leaves of $T$ in counterclockwise order, let $b_i$ (for $i = 1, \dots, n$) be the ray that represents the edge incident to the $i$-th leaf, and consider $b_i$ to be directed towards the leaf. For BMP, we are only given these rays $b_1, \dots, b_n$. Denote by $\ell_i$ the line that supports $b_i$.

Our first assumption is that $\ell_i$ and $\ell_{i+1}$ intersect in a point, say $o_i$, for all $i$, see Fig. 2. (All indices $k$ outside $\{1, \dots, n\}$ are taken as $1 + (k-1) \bmod n$.) Our second assumption is that $o_i$ is outside the relative interior of $b_i$ and $b_{i+1}$. If the input stems from VMP then these assumptions are automatically satisfied since the rays form increasing angles with the $x$-axis.

By definition line $\ell_i$ contains $o_i, o_{i-1}$ and $b_i$. Our second assumption implies that while going along line $\ell_i$, we encounter (in order) either $o_i$, $o_{i-1}$ and $b_i$ or $o_{i-1}$, $o_i$ and $b_i$. Let $d_i$ be $\|o_i o_{i-1}\|$ if the order along line $\ell_i$ is $o_i, o_{i-1}, b_i$ and $-\|o_i o_{i-1}\|$ otherwise.

Let $\beta_i$ be the angle between $\ell_i$ and $\ell_{i+1}$, measured in the wedge at $o_i$ that contains both $b_i$ and $b_{i+1}$. We will need the following aggregation of $\beta_i$:

$$B_i := \beta_1 - \beta_2 + \cdots + (-1)^{i+1}\beta_i = \sum_{k=1}^{i} (-1)^{k+1}\beta_k.$$

To solve BMP we would like to find vertices $v_1, \dots, v_n$ that lie on $b_1, \dots, b_n$. We can parameterize them by giving for each vertex $v_i$ the distance $s_i$ from $o_i$. Each edge $(v_i, v_{i+1})$ forms an inner angle $\alpha_i$ with the line $\ell_i$ supporting $b_i$, see Fig. 2. We want the $b_i$'s to be bisectors at $v_i$ and, hence, demand that $\alpha_i$ is also the inner angle of edge $(v_{i-1}, v_i)$ with $\ell_i$. For TMP, we additionally require the inner edges of the given tree $T$ to be bisectors of the corresponding edges of the polygon.

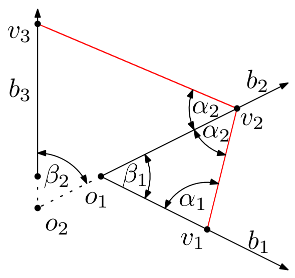


Figure 2: Figure to illustrate $b_i$, $o_i$, $\beta_i, \alpha_i$ for (part of) the Voronoi tree of Fig. 1.

## 3 Solving the Bisector-Matching Problem

**Incremental polygon construction.** We first discuss an incremental construction that simply tries to satisfy all conditions, and from it, derive an algorithm to solve the problems later.

Recall that we can describe the position of vertex $v_1$ by specifying its distance $s_1$ from $o_1$. As our main parameter, we use the angle $\alpha_1$ subtended between $b_1$ and the edge $(v_1, v_2)$.

We claim that knowledge of $s_1$ and $\alpha_1$ determines the full polygon. To see this, consider the triangle $\Delta(o_1, v_1, v_2)$. This triangle has angles $\alpha_1, \beta_1$ and $\alpha_2$, hence $\alpha_2 = \pi - \beta_1 - \alpha_1$. Also, by the sine-equation $\frac{\|o_1 v_2\|}{\sin \alpha_1} = \frac{\|o_1 v_1\|}{\sin \alpha_2}$, which with $s_i = \|o_i v_i\|$ and the definition of $d_i$ gives

$$s_2 = \|o_2 v_2\| = \|o_1 v_2\| + d_2 = \frac{\sin \alpha_1}{\sin \alpha_2} s_1 + d_2.$$

Therefore, we can determine $\alpha_2$ and $s_2$ from $\alpha_1$ and $s_1$, and so forth, and hence the whole polygon.

We observe that the above iteration may run into problems in the following cases:

1. Extending the line from $v_i$ at angle $\alpha_i$ yields a ray that does not hit line $\ell_{i+1}$. This occurs if $\pi - \beta_i - \alpha_i < 0$. In terms of the equations above this means $\alpha_{i+1} < 0$.
2. Extending the line from $v_i$ at angle $\alpha_i$ we hit line $\ell_{i+1}$, but not within $b_{i+1}$. This occurs if the computed value of $s_{i+1}$ is smaller than what is needed to be within $b_{i+1}$.
3. $\alpha_{n+1} \neq \alpha_1$, i.e., the last computed angle does not equal the initial starting angle, and hence $b_1$ is not a bisector at $v_1$.
4. $s_{n+1} \neq s_1$, i.e., while all angles are correct, the polygon has either opened up or closed down like a spiral and the vertices do not match up.

In all these cases, we have two possible sources for errors: The starting angle $\alpha_1$ could have been wrong, or the distance $s_1$ of $v_1$ to $o_1$ was too small. We now show how to use this error-information to first find a suitable value of $\alpha_1$ (if one exists), and then a suitable value of $s_1$ (if one exists), thus solving BMP.

**Achieving $\alpha_{n+1} = \alpha_1$.** Recall that we computed that $\alpha_2 = \pi - \alpha_1 - \beta_1$, and generally $\alpha_{i+1} = \pi - \alpha_i - \beta_i$. A simple proof by induction gives:

**Lemma 1** *We have*

$$\alpha_i = \begin{cases} \alpha_1 + B_{i-1} & \text{if } i \text{ is odd,} \\ \pi - (\alpha_1 + B_{i-1}) & \text{if } i \text{ is even.} \end{cases}$$

In order to obtain a closed polygon in our incremental construction scheme it is necessary that $\alpha_{n+1} = \alpha_1$. By Lemma 1, we have $\alpha_{n+1} = \alpha_1 + B_n$ if $n$ is odd and $\alpha_{n+1} = \pi - (\alpha_1 + B_n)$ otherwise. So achieving $\alpha_{n+1} = \alpha_1$ is equivalent to

$$B_n = \begin{cases} 0 & \text{if } n \text{ is even,} \\ 2\alpha_1 - \pi & \text{if } n \text{ is odd.} \end{cases} \tag{1}$$

Therefore, if $n$ is odd, then $\alpha_1 = \frac{1}{2}(\pi + B_n)$ is the only possible value of $\alpha_1$ for which the iterative procedure could give a set of feasible angles. If $n$ is even, then all values of $\alpha_1$ yield $\alpha_{n+1} = \alpha_1$ if $B_n = 0$, and none does otherwise.

**Achieving $s_{n+1} = s_1$.** Recall that $s_i$ denotes the distance of $v_i$ from $o_i$, and that we showed that $s_2 = \frac{\sin \alpha_1}{\sin \alpha_2} s_1 + d_2$, and generally

$$s_{i+1} = \frac{\sin \alpha_i}{\sin \alpha_{i+1}} s_i + d_{i+1}.$$

A simple proof by induction then shows:

**Lemma 2** *We have*

$$s_{i+1} = \frac{1}{\sin \alpha_{i+1}} \left( s_1 \sin \alpha_1 + \sum_{k=2}^{i+1} d_k \sin \alpha_k \right).$$

This lemma implies $s_{n+1} = s_1$ if and only if

$$\sum_{k=2}^{n} d_k \sin \alpha_k = 0. \tag{2}$$

Note that this equation is independent of the initial choice of $s_1$. In particular, we can simply test whether $s_{n+1} = s_1$ if there is only a unique choice of $\alpha_1$ (as in the case of $n$ odd.) If there are multiple choices of $\alpha_1$, then Equation (2) can be used to determine a suitable $\alpha_1$ as follows. Recall that $\sin(\pi - \alpha) = \sin \alpha$, and therefore $\sin \alpha_k = \sin(\alpha_1 + B_{k-1})$ regardless of whether $k$ is even or odd. Therefore

$$\begin{aligned} \sum_{k=2}^{n} d_k \sin \alpha_k &= \sum_{k=2}^{n} d_k \sin(\alpha_1 + B_{k-1}) \\ &= \sum_{k=2}^{n} d_k \left( \sin \alpha_1 \cos B_{k-1} + \cos \alpha_1 \sin B_{k-1} \right) \\ &= \sin \alpha_1 \underbrace{\sum_{k=2}^{n} d_k \cos B_{k-1}}_{=:A} + \cos \alpha_1 \underbrace{\sum_{k=2}^{n} d_k \sin B_{k-1}}_{=:B} \end{aligned}$$

Note that $A$ and $B$ are independent of $\alpha_1$. If $A \neq 0$, then the only $\alpha_1$ that satisfies (2) (and hence yields $s_{n+1} = s_1$) is $\alpha_1 = \arctan(-B/A)$. If $A = 0$ but $B \neq 0$, then the only such value would be $\alpha_1 = 0$ (but we disallow this elsewhere, so there is no feasible solution.) If $A = B = 0$, then any value $\alpha_1$ gives $s_{n+1} = s_1$.

**Achieving $\alpha_i > 0$.** Recall that we must maintain positive angles to ensure that each edge $(v_i, v_{i+1})$ actually hits the line through $b_{i+1}$. But by Lemma 1, there is a linear dependence of $\alpha_i$ on $\alpha_1$. Therefore, we can turn a lower bound on $\alpha_i$ into a lower or upper bound on $\alpha_1$. By intersecting all these bounds we obtain an interval $(L_\alpha, U_\alpha)$ such that any choice of $\alpha_1 \in (L_\alpha, U_\alpha)$ results in angles $\alpha_i$ that satisfy $\alpha_i > 0$.

Note that exactly the same technique can also accommodate upper bounds on $\alpha_i$. (For example, we could demand $\alpha_i < \pi/2$, and hence restrict the attention to polygons that are strictly convex. This would be a natural restriction for VMP, since here the input was points in convex position.) Any upper or lower bound onto $\alpha_i$ turns into an upper or lower bound for $\alpha_1$ via the linear relation, and intersecting these bounds gives an interval $(L_\alpha, U_\alpha)$ for feasible values of $\alpha_1$. (We omit the precise formula for $L_\alpha$ and $U_\alpha$ for space reasons.)

**Hitting $b_i$.** The last problem to avoid is that edge $(v_i, v_{i+1})$ hits line $\ell_i$ but not at a point within $b_i$. This can be avoided by developing lower bounds for $s_1$. We require that $s_i$ is at least the distance from $o_i$ to the closer end of $b_i$. By Lemma 2 this gives a lower bound $L_s(\alpha_1)$ on $s_1$, which depends on $\alpha_1$.

**Putting it all together.** We summarize the algorithm for the bisector-matching problem as follows. Initially start with $I^*_\alpha = (0, \pi)$ for the interval of possible values of $\alpha_1$. Compute the $B_i$'s. If $n$ is odd, set $I^*_\alpha = \{\frac{1}{2}(\pi + B_n)\}$. If $n$ is even and $B_n \neq 0$, set $I^*_\alpha = \emptyset$. Either way, if $I^*_\alpha \neq \emptyset$ then using $\alpha_1 \in I^*_\alpha$ guarantees that $\alpha_{n+1} = \alpha_1$.

Next compute values $A$ and $B$, and if $A \neq 0$, set $I^*_\alpha = I^*_\alpha \cap \{\arctan(-B/A)\}$. Now using $\alpha_1 \in I^*_\alpha$ additionally guarantees that $s_{n+1} = s_1$.

Next, compute the interval $(L_\alpha, U_\alpha)$ imposed by lower (and, perhaps, upper) bounds on the $\alpha_i$'s, and set $I^*_\alpha = I^*_\alpha \cap (L_\alpha, U_\alpha)$. Any $\alpha_1 \in I^*_\alpha$ now additionally guarantees that $\alpha_i > 0$ for all $i$.

Finally, to determine suitable values for $s_1$, compute the lower bound required for each $s_i$, and determine from it a lower bound $L_s(\alpha_1)$ for $s_1$. Then any polygon for which the vertex-bisectors coincide with $b_1, \ldots, b_n$ is described by $\alpha_1, s_1$ with $\alpha_1 \in (L_\alpha, U_\alpha)$ and $s_1 \geq L_s(\alpha_1)$, and vice versa, any such choice gives a polygon via the iterative procedure.

Clearly all computations use a linear number of operations involving real numbers and trigonometric functions. In fact, all trignometry can be avoided (if sqrt is allowed) if the input consists of coordinates for all vertices, and one additional point on each ray $b_i$. For the coordinates determine $\sin(\gamma)$ and $\cos(\gamma)$ for

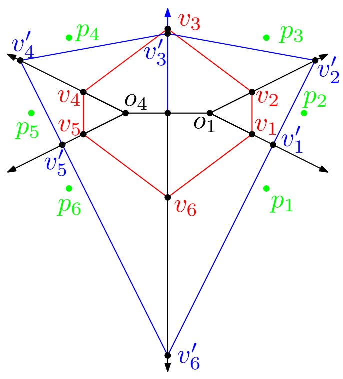


Figure 3: An input Voronoi tree that has an infinite number of solutions to BMP but only one for TMP.

angles $\gamma$ related to $\beta_1, \dots, \beta_n$, and we can compute all other needed sines and co-sines using addition formulas and solving $2 \times 2$-systems of equations. This then gives a range into which $\sin \alpha_1$ and $\cos \alpha_1$ must fall, which we output as solution. Hence a Real RAM computer model is enough, and we have:

**Lemma 3** *The bisector-matching problem can be solved in linear time in the Real RAM computer model.*

## 4 Solving the Tree-Matching Problem

If BMP yields a unique (up to off-setting) solution $P$ then TMP can be solved trivially (and in linear time) by checking whether the bisectors of $T$ do indeed form a straight skeleton of $P$. If, however, a family of polygons satisfies BMP then the situation is more tricky. If the input tree is a star then an infinite number of solutions to BMP translates to an infinite number of solutions to TMP. If the tree is not a star then this need no longer be true.

Fig. 3 shows six points $p_1, p_2, \dots, p_6$ (in green) and their Voronoi diagram (in black) together with two polygons (in red and blue) that solve the bisector-matching problem. However, only the red polygon $(v_1, v_2, \dots, v_6, v_1)$ also solves TMP, while the blue polygon $(v'_1, v'_2, \dots, v'_6, v'_1)$ only solves BMP.

So, how can we characterize solutions for TMP? The key insight is that a line $\ell$ is the bisector of edges $e_i, e_j$ of $P$ if and only if $e_i$ and $e_j$ form the same angle with a line $\ell'$ perpendicular to $\ell$. This yields for each inner edge another (linear) equation for $\alpha_1$.

Still, there do exist input trees $T$ for which all these additional equations for $\alpha_1$ are of the form $0 = 0$, i.e., where no additional constraint on $\alpha_1$ can be obtained. See for example Fig. 4. However, this happens only if all nodes of the tree have even degree. Otherwise, there exists a subtree $T'$ for which the number of edges is odd; the constraints defined by $T'$ then give a unique solution for $\alpha_1$ similar to Eqn. 1.

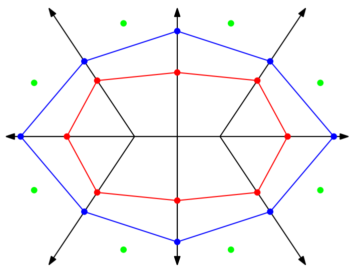


Figure 4: Example of an input Voronoi tree that has an infinite number of solutions to TMP. (Two solutions are shown in red and blue.)

**Lemma 4** *The tree-matching problem can be solved in linear time in the Real RAM computer model.*

## 5 Discussion

In this paper, we asked whether a polygon can be reconstructed if we are given a straight skeleton (but with the position of a leaf only specified within a ray.) We show that this is possible, and that the answer is unique (up to off-setting) in most cases. We can find the polygon (or a description of the infinite set of polygons) in linear number in the Real RAM computer model.

Our problem was motivated by finding a straight skeleton that matches the Voronoi diagram of a given point set. One can also turn the question around: Given the straight skeleton of a polygon $P$ or, more generally, given a planar straight-line graph $G$, can we find a set $S$ of points such that $\mathcal{VD}(S)$ matches $G$?

This is very easy if $G$ has two vertices of odd degree on one face; in particular the answer is unique. If all nodes of $G$ have even degree then $S$ need not be unique: Fig. 4 shows a tree for which both an infinite number of defining point sets $S$ (for the Voronoi diagram) and polygons $P$ (for the straight skeleton) exist. We will present a solution to this problem (which uses an entirely different approach) in a forthcoming paper.

## References

[1] O. Aichholzer, H. Cheng, S.L. 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.

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

[3] S. Huber and M. Held. A Fast Straight-Skeleton Algorithm Based on Generalized Motorcycle Graphs. *Internat. J. Comput. Geom. Appl.*, in press 2013.

[4] G. Liotta and H. Meijer. Voronoi Drawings of Trees. *Comput. Geom. Theory and Appl.*, 24(3):147–178, 2003.

# A conceptual take on invariants of low-dimensional manifolds found by integration

Mathijs Wintraecken [*] Gert Vegter [†]

**Introduction** Recently a lot of effort has been put into the the study of topological properties of Gaussian random fields [2, 6, 15], using among others Euler integration [4]. This is especially important in the context of cosmology, because these random fields are believed to describe the density fluctuations in the early universe [3] or are at least a good approximation of these fluctuations. The topological aspect has gained popularity over the last few years as there is a great deal of numerical evidence of the fact that the topology of the density field is very sensitive to small deviations from Gaussianity. A nice closed formula for the expectation value of the Euler characteristic of the level sets of these fields has been found [3, 8]. This result relies on the Gauss-Bonnet theorem which relates the integral of some intrinsic quantity whose origins lie in the field of differential geometry, namely the Gaussian curvature, to some topological invariant, the Euler characteristic. The Gauss-Bonnet theorem can be generalized to higher dimensional manifolds, using the theory of characteristic classes. For a very elegant exposition we refer to Milnor and Stasheff [12] or alternatively Spivak [13]. The generalization only goes so far, in fact Abrahamov [1] proved that the invariants thus produced (the so-called Chern numbers) are unique, up to some equivalence. See Gilkey [7] for a modern (and more extensive) treatment. This result implies that no expressions can be found, using a similar straightforward integration techniques, for all other interesting topological invariants associated to Gaussian random fields in three dimensions, such as Betti numbers. Both the proof by Abrahamov and the modern treatment by Gilkey involve a significant amount of analysis and machinery. Below we provide a proof of a similar statement for two and three dimensional manifolds, which does not need to call in an elaborate set of calculations.

The formulation of the main result will be along the lines of the following question proposed by I.M. Singer: ‘Suppose that $f$ is a scalar valued invariant of the metric such that $t = \int f \mathrm{dvol}$ is independent of the metric. Then is there some universal constant $c$ so that $t = c\chi(M)$?’ This question has reportedly ([7]) been answered in the affirmative by E. Miller. Our proof relies heavily on the classification of two dimensional closed surfaces and on Heegaard splitting. A discussion of the classification can be found in [9] or [10] , for the latter we refer to [5] or [14]. We complete our discussion by some remarks on generalizations.

## The two dimensional case

**Theorem 1** *Let $M$ be an orientable[1] two dimensional real compact manifold and $f$ a function on $M$ which is completely determined by the metric and its derivatives, that is locally $f$ can be written as $f(g, \partial g, \ldots)$ with $g$ the metric, such that*

$$\int_M f \, \mathrm{d}vol,$$

*where* d*vol indicates the volume form, is a topological invariant $t$. Then there exists a real number $c$ such that $t = c\,\chi(M)$, where $\chi$ denotes the Euler characteristic.*

*Proof* First we note that the space of Riemannian metrics on a manifold is connected. This is obvious because if $g$ and $\tilde{g}$ are metrics then so is $\lambda g + (1-\lambda)\tilde{g}$ for all $\lambda \in [0, 1]$. This means that we can assume without loss of generality that $M$ is isometrically embedded in $\mathbb{R}^3$, because we can choose $\tilde{g}$ to be the standard metric of $M$. Now let $f$ be a function as described in the theorem, such that

$$\int_M f \, \mathrm{dvol} = t$$

is a topological invariant. Suppose that for the two sphere $\mathbb{S}^2$ we have

$$\int_{\mathbb{S}^2} f \, \mathrm{dvol} = 2c,$$

where $c$ is some constant. From this we can conclude that for the sphere $t = c\chi(M)$.

We can now deform the two-sphere as follows. A small region is pushed outwards and bent -in a sufficiently smooth manner- such that this region contains three equally spaced parallel cylinders pieces all of the same radius. We can now cut in the cylindrical part along the plane orthogonal to the cylinder and

[*]JBI, University of Groningen, m.h.m.j.wintraecken@rug.nl

[†]JBI, University of Groningen, g.vegter@gmail.com

[1]Clearly the integral over a non-orientable manifold does not make sense.

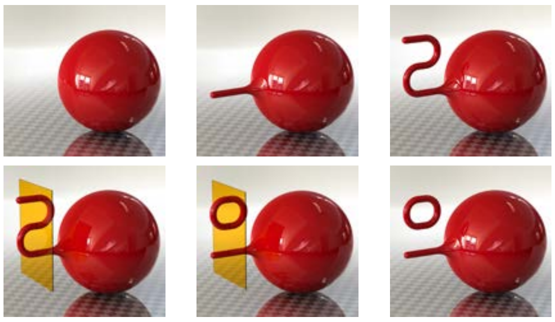


Figure 1: From left to right, top to bottom we have depicted: the sphere, the deformation (in two steps), the deformed surface with cutting lines indicated by the yellow glass plane and the reassembled surfaces (with and without cutting lines).

reassemble the parts so that we recover a topological sphere but also get a torus. The integral is not altered because integrals are additive. The procedure is illustrated in figure 1. Because the integral is clearly additive for unions this implies that

$$\int_{C_1} f\,\mathrm{dvol} = 0,$$

where $C_1$ is a surface of genus 1. Generally we shall denote a surface of genus $g$ by $C_g$.

The rest of the proof is inductive in nature. We begin with a topological genus-$g$ torus and two spheres. We deform these surfaces so that the spheres contain a piece of a cylinder, both of the same radius, and the $n$-torus such that it contains two pieces of the cylinder, again of the same radius, so that if these pieces are deleted one of the remaining surfaces is itself a topological cylinder. We again cut the cylindrical pieces in half and reassemble the part so that we have a genus-$g-1$ torus and a sphere. As sketched in figure 2.

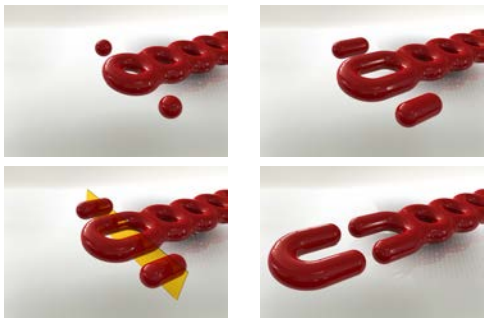


Figure 2: From left to right, top to bottom we have depicted: An $n$-torus and two spheres, the same surfaces deformed, the deformed surfaces with the lines along which we cut indicated by yellow glass plane and the reassembled surfaces.

We can now conclude that

$$\int_{C_g} f\mathrm{dvol} + 2\int_{\mathbb{S}^2} f\mathrm{dvol} = \int_{C_{g-1}} f\mathrm{dvol} + \int_{\mathbb{S}^2} f\mathrm{dvol}$$

and thus by induction that

$$\int_{C_g} f\mathrm{dvol} = c(2-2g) = c\,\chi(C_g).$$

By the classification of all 2-manifolds we have proven the theorem for all two dimensional real manifolds embedded in $\mathbb{R}^3$. □

**Three dimensions** We will now focus on the three dimensional case. The intuition for the following proof is much strengthened by the remark that a Morse function $h$ on some manifold $M$ can always be interpreted as height function. This can be easily seen as follows: Let $M$ be isometrically embedded in $\mathbb{R}^n$, possibly using the Nash embedding theorem. Then we can add the value of the Morse function as another coordinate to a point $p \in M \subset \mathbb{R}^n$, so that the manifold $M$ is embedded in $\mathbb{R}^{n+1}$ and the last coordinate is the height.

**Theorem 2** *Let $M$ be a three-dimensional real manifold and $f$ a function which is completely determined by the metric and its derivatives such that*

$$\int_M f\,\mathrm{d}vol,$$

*is a topological invariant $t$. Then we have $t = 0$.*

*Proof* The first step in our proof will consist of showing that if $M = C_g \times \mathbb{S}^1$ we have that

$$\int_M f\,\mathrm{dvol} = 0.$$

To show this we shall consider any manifold $N$, that admits a Heegaard splitting of genus $g$. This means that the manifold $N$ can be represented as the attachment of two three-dimensional manifolds, which are both homeomorphic to a three-dimensional ball with $g$ handles, with respect to a diffeomorphism of their boundaries. We further have that there exists a Morse function $h$ on $N$ with one minimum and one maximum and all critical points of index 1, 2 correspond to the critical values $c_1$ and $c_2$ respectively with $c_1 < c_2$, see [5]. This has been schematically represented in the leftmost picture in figure 3. [2]

[2]Note that conversely a Heegaard splitting also gives a Morse function in a natural manner. Namely we start with Morse functions on both g-handled balls, by simply taking a Morse function on the standard g-handled ball and pulling back via the diffeomorphisms to the g-handled balls in question. Now theorem 1.4 and lemma 3.7 of [11], give a differentiable structure on the union with a smooth structure compatible with the given differentiable structure on the different parts, moreover such that the Morse functions on both parts piece together to a smooth function.

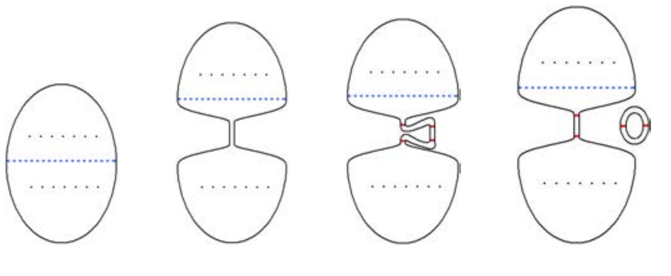


Figure 3: From left to right we have sketched: A manifold admitting a Heegaard splitting; the critical points of the Morse function are indicated as dots and the attachment by a blue dashed line, the same manifold with a small part of it brought to a standard $C_g \times [-\delta, \delta]$ metric, the deformed surface with cutting lines (red) indicated and the reassembled surfaces.

We now define for every surface $C_g$ of genus $g$, some metric induced by an embedding in $\mathbb{R}^3$, exhibiting $\mathbb{Z}_2$ symmetry. We shall refer to this Riemannian manifold as the standard surface of genus $g$. In the following we view $N$ as embedded in $\mathbb{R}^k$. Let $f$ be as in theorem 1 such that

$$\int_N f \, \mathrm{dvol},$$

is a topological invariant $t$. For some sufficiently small $[a_1, b_1] \subset \mathbb{R}$, with $c_1 < a_1 < \alpha_1 < \beta_1 < b_1 < c_2$, we smoothly and isotopically deform $h^{-1}([a_1, b_1]) \cap M \sim C_g \times [a_1, b_1]$, so that $h^{-1}([\alpha_1, \beta_1]) \cap M$ becomes isometric to the standard $C_g \times [\alpha_1, \beta_1] \subset \mathbb{R}^4 \subset \mathbb{R}^k$ given by the standard $C_g$ and the ordinary Cartesian product. This standard form is referred to as straight. We shall now deform this part of the manifold so that it consists of a straight piece and two pieces which are straight at the beginning and the end but are bent in the middle so that if we cut along the the boundaries of the pieces and reassemble we recover the original manifold and $C_g \times \mathbb{S}^1$. The procedure is sketched in figure 3. From this we conclude that

$$\int_N f \, \mathrm{dvol} = \int_N f \, \mathrm{dvol} + \int_{C_g \times \mathbb{S}^1} f \, \mathrm{dvol},$$

where we again used local isotopy and the additivity of integration. Therefore,

$$\int_{C_g \times \mathbb{S}^1} f \, \mathrm{dvol} = 0.$$

The next part of the proof relies on the fact that the sphere ($\mathbb{S}^3$) allows a Heegaard splitting of every genus $g$, see [5]. Let $M$ be a manifold which allows a Heegaard splitting of genus $g$. We now deform two pieces of the manifold into parts isometric to $C_g \times [\alpha_1, \beta_1]$ and $C_g \times [\alpha_2, \beta_2]$, with $\alpha_1 < \beta_1 < \alpha_2 < \beta_2$, so that for all $p_1 \in (\alpha_1, \beta_1)$ and $p_2 \in (\alpha_2, \beta_2)$ both

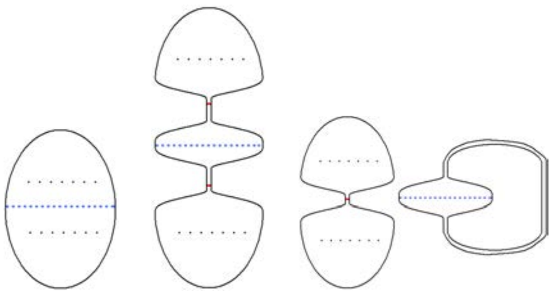


Figure 4: A Heegaard splitting, then the same manifold with two small parts brought to a standard metric both on another side of the 'attachment line', cutting lines (red) are also indicated, and finally the reassembled surface (two connected components).

$h^{-1}((-\infty, p_1)) \cap M$ and $h^{-1}(p_2, \infty) \cap M$ are topological spheres with $g$ handles whose boundary is isometric to the standard genus $g$ surface, as discussed above. We can now smoothly deform $h^{-1}((p_1, q_1)) \cap M$ and $h^{-1}((q_2, p_2)) \cap M$, with $p_1 < q_1 < \beta_1$ and $\alpha_2 < q_2 < p_2$ (see figure 5), such that if we cut along the $p_i$ and $q_i$ lines and reassemble (possibly using $\mathbb{Z}_2$ symmetry) we recover two topological manifolds, with given topology. One of the manifolds we thus construct is a manifold admitting a Heegaard splitting of genus $g$. The attachment diffeomorphism, of the latter, on the boundary of the sphere with $g$ handles is the identity. This manifold shall be denoted by $M_g^{S(3\mathrm{D})}$. The other manifold is a topological $C_g \times \mathbb{S}^1$-manifold. The entire procedure is sketched in figure 4.

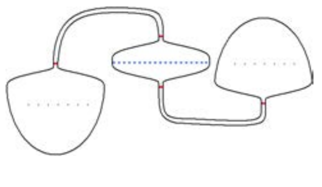


Figure 5: A sketch of the deformed manifold with the cutting lines (red) and the 'attachment line' (blue).

This means that by deforming, cutting and pasting a manifold $M$, which allows a Heegaard splitting of genus $g$, we find the following equalities

$$\begin{aligned} \int_M f \, \mathrm{dvol} &= \int_{M_g^{S(3\mathrm{D})}} f \, \mathrm{dvol} + \int_{C_g \times \mathbb{S}^1} f \, \mathrm{dvol} \\ &= \int_{M_g^{S(3\mathrm{D})}} f \, \mathrm{dvol} + 0, \end{aligned}$$

where $f$ is as defined in the theorem. If we now use that the sphere ($\mathbb{S}^3$) allows a Heegaard splitting of every genus $g$ we find that

$$\int_M f\,\mathrm{dvol} = \int_{M_g^{S(3\mathrm{D})}} f\,\mathrm{dvol} = \int_{\mathbb{S}^3} f\,\mathrm{dvol}. \quad (1)$$

Following this observation, we are able to use the result of the first part of the proof,

$$\int_{C_g \times \mathbb{S}^1} f\,\mathrm{dvol} = 0.$$

This immediately translates into

$$\int_{\mathbb{S}^2 \times \mathbb{S}^1} f\,\mathrm{dvol} = 0.$$

We notice that both $\mathbb{S}^3$ and $\mathbb{S}^2 \times \mathbb{S}^1$ allow a Heegaard splitting of genus 1, so that

$$\int_{\mathbb{S}^3} f\,\mathrm{dvol} = \int_{M_1^{S(3\mathrm{D})}} f\,\mathrm{dvol} = \int_{\mathbb{S}^2 \times \mathbb{S}^1} f\,\mathrm{dvol} = 0. \quad (2)$$

Combining equations (1) and (2) yields

$$\int_M f\,\mathrm{dvol} = 0,$$

for any manifold $M$ and $f = f(g, \partial g, \ldots)$ a function determined by the metric and all its derivatives. $\square$

**Discussion** One can wonder about generalizations of the methods stated above to manifolds of general dimension. Some of these generalizations are immediately obvious, for example the procedure sketched in figure 3 can be used in any dimension so see that for $f$ and $t$ as in the theorem

$$\int_{M^{d-1} \times \mathbb{S}^1} f\,\mathrm{dvol} = t$$

implies that $t = 0$, where $M^{d-1}$ is any manifold of dimension $d-1$ occurring as level set. However a full classification of all integrals yielding a topological invariant does not seem feasible because there is no easy classification of manifolds of dimension $d-1$ for $d > 3$ (and none for $d > 4$), occurring as the level sets of a Morse function on a manifold of dimension $d$.

**Acknowledgements** The authors are greatly indebted to Erik van den Ban for discussion and suggestions. We further acknowledge Rien van de Weijgaert and Ramsay Dyer for discussion and Arthemy Kiselev for discussion and use of his knowledge of Russian. This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL Computational Geometry Learning).

## References

[1] A.A. Abrahamov. On the topological invariants of Riemannian spaces obtained by integration of tensor fields. *Doklady Akademii Nauk*, 81:125–128, 1951.

[2] J.A. Adler, O. Bobrowski, M.S. Norman, E. Subag, and S. Weinberger. Persistent homology for random fields and complexes. *Borrowing Strength: Theory powering applications*, 6:124–143, 2010.

[3] F.M. Bardeen, J.R. Bond, N. Kaiser, and Szalay. The statistics of peaks of Gaussian random fields. *The astrophysical Journal*, 304:15–61, 1986.

[4] Y. Baryshnikov and R. Ghrist. Target enumeration via Euler characteristic integrals. *SIAM Journal on Applied Mathematics*, 70:825–844, 2009.

[5] A.T. Fomenko. *Differential Geometry and Topology.* Contemporary Soviet Mathematics. Consultants Bureau, New York and London, 1987.

[6] C. Gay, C. Pichon, and D. Pogosyan. Non-Gaussian statistics of critical sets in 2D and 3D: Peaks, voids, saddles, genus, and skeleton. *Physical Review D*, 85(2):023011, January 2012.

[7] P.B. Gilkey. *Invariance Theory, The Heat Equation, And the Atiyah-Singer Index Theorem.* Mathematics Lecture Series. Publich or Perish, inc., 1984.

[8] A.S.J. Hamilton, J.R. Gott III, and D. Weinberg. The topology of the large-scale structure of the universe. *The astrophysical Journal*, 309:1–12, 1986.

[9] M.W. Hirsch. *Differential Topology.* Springer-Verlag: New York, Heidelberg, Berlin, 1976.

[10] W.S. Masssey. *Algebraic Topology: An Introduction.* Springer-Verlag: New York, Heidelberg, Berlin, 1977.

[11] J. Milnor. *Lectures on the h-cobordism theorem.* Princton Mathematical notes. Princeton university press, 1965.

[12] J.W. Milnor and J.D. Stasheff. *Characteristic Classes.* Number 76 in Annals of Mathematics Studies. Princeton University Press and University of Tokyo Press, Princeton, New Jersey, 1974.

[13] M Spivak. *A comprehensive introduction to differential geometry: Volume V.* Publish or Perish, 1975.

[14] J. Stillwell. *Classical Topology and Combinatorial Group Theory.* Graduate Texts in Mathematics. Springer, 1993.

[15] Rien van de Weygaert, Gert Vegter, Herbert Edelsbrunner, Bernard J.T. Jones, Pratyush Pranav, Changbom Park, Wojciech A. Hellwing, Bob Eldering, Nico Kruithof, E.G.P.(Patrick) Bos, Johan Hidding, Job Feldbrugge, Eline ten Have, Matti van Engelen, Manuel Caroli, and Monique Teillaud. Alpha, betti and the megaparsec universe: On the topology of the cosmic web. *Transactions on Computational Science*, 14:60–101, 2011.

# Cut Equivalence of $d$-Dimensional Guillotine Partitions

Andrei Asinowski* Gill Barequet† Toufik Mansour‡ Ron Y. Pinter†

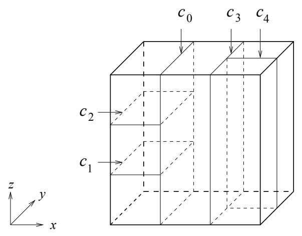


Figure 1: A guillotine partition of a 3D box.

## Abstract

A *guillotine partition* of a $d$-dimensional axis-aligned box $B$ is a recursive partition of $B$ by axis-aligned hyperplane cuts. Two guillotine partitions are *box-equivalent* if their boxes satisfy compatible order relations wrt. the axes. In this work we define *cut-equivalence* of guillotine partitions, derived in a similar way from order relations of cuts. We study structural properties of these kinds of equivalence, and enumerate cut-equivalence classes of $d$-dimensional guillotine partitions.

## 1 Introduction

### 1.1 Basic Definitions

Let $B$ be a $d$-dimensional ($d$-D, in short) axes-aligned box. A *guillotine partition* of $B$ is either trivial ($B$ itself) or a partition obtained by recursively cutting $B$ by a hyperplane perpendicular to an axis $x_i$, $1 \leq i \leq d$, into two sub-boxes whose inner partitions are also guillotine. $B$ often denotes the partition as well as the box. Fig. 1 shows such a partition of a 3D box.

Guillotine partitions have been studied intensively due to their important role in geometric algorithms, visualization of scientific data, integrated circuit design, and many more fields; see, e.g., [4, 6, 7]. The number of guillotine partitions was discussed in 2D [8] and in higher dimension [1, 3].

Understanding the structure of guillotine partitions is important both from the combinatorial point of view and for analyzing data structures that hold the partitions and algorithms that generate them. In many works, guillotine partitions that have the same recursive structure wrt. their boxes are considered identical. Another kind of elements in guillotine partitions is their defining cuts, whose structure, in some applications, is more relevant to the complexity or running-time analysis than that of the boxes. In this paper we study these two types of structures. To this aim, we define two kinds of equivalence of $d$-D partitions: B-equivalence and C-equivalence ($B$ for *boxes* and $C$ for *cuts*), in terms of order relations between boxes and cuts, resp. We show that B-equivalence is the intuitive way to identify guillotine partitions, while C-equivalence is a more coarse way to do it. Finally, we show how C-equivalence is related to B-equivalence, and enumerate C-equivalence classes.

The intersection of a box $B$ with a hyperplane that splits it into two sub-boxes is a *primary cut* (for example, $c_0$ and $c_3$ in Fig. 1). A *cut* in a guillotine partition is either a primary cut, or (in a recursive manner) a cut in the partition of one of the sub-boxes. It is assumed that parallel cuts do not intersect, that is, they cannot share a $(d-2)$-dimensional "edge." A guillotine partition $B$ with $n$ cuts consists of $n+1$ boxes; in this case we say that the *size* of $B$ is $n+1$.

Throughout this paper, the dimension $d$ is assumed fixed, and all the guillotine partitions are assumed to be $d$-dimensional. For shortness, we shall usually just write "partition" instead of "guillotine partition."

If a nontrivial partition $B$ has several primary cuts, then they are all perpendicular to the same axis. If the primary cut(s) of a nontrivial partition $B$ is (are) perpendicular to the $x_i$ axis, we say that $B$ is an $x_i$-*aligned* partition. The parts of $B$ bounded by two adjacent primary cuts, as well as the part below the lowest (wrt. $x_i$) primary cut, and the part above the highest primary cut, are called *slices* and denoted by $S_1, \ldots, S_k$ (ordered from bottom to top wrt. $x_i$). A *trivial slice* and a 2-*slice* are slices of size 1 and 2, resp. Primary cuts of any slice are not parallel to those of $B$. The partition in Fig. 1 is an $x$-aligned partition into three slices: $S_1, S_2, S_3$ are a $z$-aligned slice of size 3, a trivial slice, and a $y$-aligned 2-slice, resp. The lowest primary cut is called the *principal cut* of $B$. The sub-boxes obtained by cutting $B$ along the principal cut are denoted by $B^-$ and $B^+$ (the part of $B$ below and above the principal cut, resp.). In Fig. 1, the principal cut of $B$ is $c_0$, the principal cut of $B^-$ is $c_1$, and the principal cut of $B^+$ is $c_3$.

*Dept. of Mathematics, The Technion, Haifa 32000, Israel. E-mail: andrei@tx.technion.ac.il; Current address: Inst. für Informatik, FU-Berlin, Takustr. 9, 14195 Berlin, Germany.
†Dept. of Computer Science, The Technion, Haifa 32000, Israel. E-mail: {barequet,pinter}@cs.technion.ac.il
‡Dept. of Mathematics, Univ. of Haifa, Mount Carmel, Haifa 31905, Israel. E-mail: toufik@math.haifa.ac.il

## 1.2 Order Relations in Guillotine Partitions

We define $d$ order relations between boxes and between cuts in $d$-dimensional guillotine partitions.

**Definition 1** *Consider a nontrivial $d$-dimensional guillotine partition $B$ with principal cut $c$.*

- *Let $K, L$ be two distinct boxes in the partition. Box $K$ is* below *$L$ (or $L$ is* above *$K$) wrt. the axis $x_i$ ($1 \le i \le d$), to be denoted by $K \underset{i}{\Leftarrow} L$, if*
  - *$B$ is $x_i$-aligned, $K$ is in $B^-$, $L$ is in $B^+$;*
  - *$K$ and $L$ are in $B^-$, and $K \underset{i}{\Leftarrow} L$ in the partition of $B^-$; or*
  - *$K$ and $L$ are in $B^+$, and $K \underset{i}{\Leftarrow} L$ in the partition of $B^+$.*
- *Let $u, v$ be two distinct cuts in the partition. Cut $u$ is* below *$v$ (or $v$ is* above *$u$) wrt. the axis $x_i$ ($1 \le i \le d$), to be denoted by $u \underset{i}{\leftarrow} v$, if*
  - *$B$ is $x_i$-aligned, $u$ is in $B^-$, $v$ is in $B^+$;*
  - *$B$ is $x_i$-aligned, $u$ is in $B^-$, $v = c$;*
  - *$B$ is $x_i$-aligned, $u = c$, $v$ is in $B^+$;*
  - *$u$ and $v$ are in $B^-$, and $u \underset{i}{\leftarrow} v$ in the partition of $B^-$; or*
  - *$u$ and $v$ are in $B^+$, and $u \underset{i}{\leftarrow} v$ in the partition of $B^+$.*

All relations $\underset{i}{\Leftarrow}, \underset{i}{\leftarrow}$ are (irreflexive) partial orders.

Let $B$ be an $x_i$-aligned partition. If boxes $K, L$ lie, resp., in $B^-, B^+$, then they are *$i$-comparable* (comparable in $\underset{i}{\Leftarrow}$) but by no other order $\underset{j}{\Leftarrow}$. A similar claim holds for cuts. By induction, we obtain the following:

**Observation 1** *In any $d$-D guillotine partition,*

1. *Each pair $K, L$ of distinct boxes is $i$-comparable ($K \underset{i}{\Leftarrow} L$ or $L \underset{i}{\Leftarrow} K$) for a* unique *$i$, $1 \le i \le d$.*
2. *Each pair $u, v$ of distinct cuts is $i$-comparable ($u \underset{i}{\leftarrow} v$ or $v \underset{i}{\leftarrow} u$) for a* unique *$i$, $1 \le i \le d$.*

**Definition 2** *Let $B$ and $D$ be two $d$-dimensional guillotine partitions, both of size $n+1$. We say that*

- *$B, D$ are* B-equivalent *if one can label the boxes of $B$ by $K_1, \ldots, K_{n+1}$ and the boxes of $D$ by $L_1, \ldots, L_{n+1}$ s.t. for all $1 \le \{\ell, m\} \le n+1$ we have $K_\ell \underset{i}{\Leftarrow} K_m$ iff $L_\ell \underset{i}{\Leftarrow} L_m$. Two such labelings are said to be* B-compatible.
- *$B, D$ are* C-equivalent *if one can label the cuts of $B$ by $u_1, \ldots, u_n$ and the cuts of $D$ by $v_1, \ldots, v_n$ s.t. for all $1 \le \{\ell, m\} \le n$ we have $u_\ell \underset{i}{\leftarrow} u_m$ iff $v_\ell \underset{i}{\leftarrow} v_m$. Two such labelings are said to be* C-compatible.

## 1.3 Enumeration of C-Equivalence Classes

Denote by $G_{d,n}$ the number of B-equivalence classes of $d$-D partitions of size $n+1$, and by $H_{d,n}$ the number of C-equivalence classes of $d$-D partitions of size $n+1$. Let $G_d(t) = \sum_{n \ge 0} G_{d,n} t^n$ and $H_d(t) = \sum_{n \ge 0} H_{d,n} t^n$ be the associated generating functions.

Ackerman *et al.* [1] proved that for $n \ge 1$ $G_{d,n} = \frac{1}{n} \sum_{k=0}^{n-1} \binom{n}{k} \binom{n}{k+1} (d-1)^k d^{n-k}$ (clearly, $G_{d,0} = 1$), and that $G_d(t) = \frac{1-t-\sqrt{(1-t)^2-4(d-1)t}}{2(d-1)t}$. Asinowski et al. [2] proved that $H_{2,n} = \sum_{i=0}^{\lfloor (n+1)/2 \rfloor} (-1)^i \binom{n+1-i}{i} G_{2,n-i}$, which can also be written as $H_2(t) = (1-t)\ G_2(t(1-t))$. Here we generalize this formula to the $d$-D case.

**Theorem 1**

$$H_{d,n} = \sum_{i=0}^{\lfloor (n+1)/2 \rfloor} (-1)^i (d-1)^i \binom{n+1-i}{i} G_{d,n-i}, \quad (1)$$

*or, equivalently,*

$$H_d(t) = (1-(d-1)t)\ G_d(t(1-(d-1)t)). \quad (2)$$

## 2 Structural Properties of B- and C-Equivalence

In this section we list a series of results. In particular, they will be used in the proof of Theorem 1. For most of them, the proofs are omitted in this version.

Let $\leftarrow$ be a partial order in a finite set $X$. A *series decomposition* of $X$ is a partition $(X_1, X_2)$ of $X$ ($X_1, X_2 \ne \varnothing$) s.t. for each $\alpha \in X_1, \beta \in X_2$ we have $\alpha \leftarrow \beta$. $X$ is *series decomposable* if it has a series decomposition. A component of such a decomposition may be further decomposable. A *final series decomposition* of $X$ is a partition $(X_1, X_2, \ldots, X_k)$ of $X$ ($X_1, X_2, \ldots, X_k \ne \varnothing$) s.t. for $i = 1, 2, \ldots, k-1$ we have $\alpha \leftarrow \beta$ for each $\alpha \in X_i, \beta \in X_{i+1}$, and the components $X_i$ are not further decomposable. The final series decomposition of $X$ is uniquely determined.

**Lemma 2** *Let $X$ be a finite set, and let $\underset{i}{\leftarrow}$ and $\underset{j}{\leftarrow}$ be two partial orders in $X$. If $X$ is series decomposable wrt. both orders, then there are two members of $X$ which are comparable in both orders.*

**Proof.** Let $(I^-, I^+)$ and $(J^-, J^+)$ be series decompositions of $X$ wrt. $\underset{i}{\leftarrow}$ and $\underset{j}{\leftarrow}$, resp. Assume wlog. that $I^- \cap J^- \ne \varnothing$. Let $\alpha \in I^- \cap J^-, \beta \in I^+, \gamma \in J^+$. If $\beta \in J^+$, then $\alpha \underset{i}{\leftarrow} \beta$ and $\alpha \underset{j}{\leftarrow} \beta$; if $\gamma \in I^+$, then $\alpha \underset{i}{\leftarrow} \gamma$ and $\alpha \underset{j}{\leftarrow} \gamma$; and if $\beta \in J^-$ and $\gamma \in I^-$, then $\gamma \underset{i}{\leftarrow} \beta$ and $\beta \underset{j}{\leftarrow} \gamma$. □

For the order $\underset{i}{\Leftarrow}$ between boxes of $B$, define the *inherited* order $\underset{i}{\Leftarrow}$ in $[n+1]$ as follows: $\alpha \underset{i}{\Leftarrow} \beta$ in $[n+1]$ if and only if $K_\alpha \underset{i}{\Leftarrow} K_\beta$ in $B$.

**Lemma 3** *Let $B$ be a nontrivial $x_i$-aligned partition of size $n+1$. Let $K_1, \dots, K_{n+1}$ be the boxes of $B$. Let $S_1, \dots, S_k$ be the slices of $B$ (ordered from bottom to top wrt. the $x_i$ axis). For each $\ell = 1, \dots, k$, denote $X_\ell = \{\alpha \in [n+1] : K_\alpha \in S_\ell\}$. Then, $(X_1, \dots, X_k)$ is the final series decomposition of $[n+1]$ wrt. $\underset{i}{\Leftarrow}$.*

**Lemma 4** *If $B$ and $D$ are nontrivial B-equivalent guillotine partitions, then they are similarly aligned.*

We say that a cut is *$i$-universal* if it is $i$-comparable with all other cuts.

**Observation 2** *Let $B$ be an $x_i$-aligned guillotine partition of size at least 3.*

1. *A cut $u$ in $B$ is $i$-universal if and only if either $u$ is a primary cut or $u$ is the inner cut of a 2-slice.*
2. *If $u$ is the inner cut of a 2-slice $S_m$, then $u$ has a unique below-neighbor $u'$ which is a primary cut of $B$ (unless $S_m$ is the lowest slice of $B$) and a unique above-neighbor $u''$ which is a primary cut of $B$ (unless $S_m$ is the highest slice of $B$).*

**Corollary 5** *For $j \neq i$, a partition $B$ of size at least 3 cannot have both $i$- and $j$-universal cuts.*

**Lemma 6** *If $B, D$ are C-equivalent partitions of size at least 3, then they are similarly aligned.*

The next lemma says that the usual way to identify partitions that have the same recursive structure wrt. boxes, coincides with our notion of B-equivalence.

**Lemma 7** *Let $B, D$ be two nontrivial partitions of the same size. $B$ and $D$ are B-equivalent iff*
*– The principal cuts of $B$ and $D$ are parallel;*
*– Partition $B^-$ is B-equivalent to partition $D^-$; and*
*– Partition $B^+$ is B-equivalent to partition $D^+$.*

Next, we study the structure of C-equivalence. Since any two trivial partitions are C-equivalent and any two partitions of size 2 are C-equivalent, we state the results for partitions of size at least 3.

We start with some definitions. A *block* is either a trivial partition or a nontrivial partition in which $\underset{i}{\leftarrow}$ is linear for some $i$, $1 \leq i \leq d$. If $B$ is such a block of size at least 3, then it is $x_i$-aligned; see Fig. 2. A partition $B$ is a block iff each slice of $B$ is either trivial or a 2-slice. A *block in a guillotine partition $B$* is a set of boxes of $B$ whose union is a box which, with its inner partition, is a block. The blocks in $B$ are ordered by inclusion and each box in $B$ belongs to exactly one maximal block. A *semiblock* is a partition $B$ which is not a block and for some $i$, $1 \leq i \leq d$, there is a (unique) minimum element wrt. $\underset{i}{\leftarrow}$. In such a case $B$ is of size at least 3 and it is $x_i$-aligned.

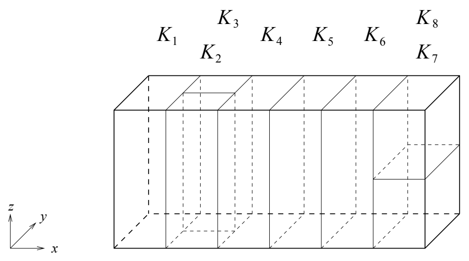


Figure 2: An $x$-aligned block of size 8.

**Lemma 8** *Let $B$ be an $x_i$-aligned semiblock. Let $u$ be the cut s.t. $u$ and all cuts below it (wrt. $\underset{i}{\leftarrow}$) are $i$-universal, and $u$ is a (uniquely determined) maximum element (wrt. $\underset{i}{\leftarrow}$) with this property. Then, $u$ is a primary cut.*

The cut $u$ as in Lemma 8 will be called the *special cut* of a semiblock $B$. The part of $B$ below the special cut is a block, and the special cut is the highest primary cut with this property. Denote by $\dot{B}$ (resp., $\ddot{B}$) the part of $B$ below (resp., above) the special cut.

**Lemma 9** *Let $B$ be a semiblock, and $D$ be a partition of the same size. $B, D$ are C-equivalent iff*
*– The principal cuts of $B$ and $D$ are parallel;*
*– Partition $\dot{B}$ is C-equivalent to partition $\dot{D}$; and*
*– Partition $\ddot{B}$ is C-equivalent to partition $\ddot{D}$.*
*In particular, a partition which is C-equivalent to a semiblock, is itself a semiblock.*

A partition which is neither a block nor a semiblock will be called *regular*. It is easy to see that a partition $B$ is regular if and only if $|B| > 3$ and $|B^-| > 2$.

**Observation 3** *Let $B$ be an $x_i$-aligned regular partition. Let $u$ be the minimal (wrt. $\underset{i}{\leftarrow}$) $i$-universal cut. Then, $u$ is the principal cut of $B$.*

**Lemma 10** *Let $B$ be a* regular *partition, and $D$ be a partition of the same size. $B, D$ are C-equivalent iff*
*– The principal cuts of $B$ and $D$ are parallel;*
*– Partition $B^-$ is C-equivalent to partition $D^-$; and*
*– Partition $B^+$ is C-equivalent to partition $D^+$.*
*In particular, a partition which is C-equivalent to a regular partition is also a regular partition.*

Next, we mention two facts concerning partitions with "small" lowest slice.

**Lemma 11** *Let $B, D$ be partitions of size at least 3 s.t. $|B^-| = |D^-| = 1$. $B$ and $D$ are C-equivalent iff*
*– The principal cuts of $B$ and $D$ are parallel; and*
*– Partition $B^+$ is C-equivalent to partition $D^+$.*

**Observation 4** *Let $B$ be a partition s.t. $|B^-| = 2$. Then, there exists a partition $D$ which is C-equivalent to $B$ and satisfies $|D^-| = 1$.*

The next two claims show how C-equivalence is related to B-equivalence.

**Proposition 12** *If two guillotine partitions $B$ and $D$ are B-equivalent, then they are C-equivalent.*

**Proposition 13** *Let $B, D$ be partitions of the same size. $B, D$ are C-equivalent iff there exist partitions $B', D'$ s.t. $B'$ is B-equivalent to $B$, $D'$ is B-equivalent to $D$, and $B'$ and $D'$ can be obtained from each other by replacing some blocks by C-equivalent blocks.*

## 3 Proof of Theorem 1

Since we assume $d$ to be fixed and in order to simplify notation, we shall write $H_n, H(t), G_n, G(t)$ rather than $H_{d,n}, H_d(t), G_{d,n}, G_d(t)$.

First, $H_0 = H_1 = 1$. Let $n \geq 2$. By Lemma 6, two C-equivalent partitions are similarly aligned. Therefore, by symmetry, $H_n = dH_n^{(1)}$, where $H_n^{(1)}$ is the number of C-equivalence classes of $x_1$-aligned partitions of size $n+1$. By Lemmas 10 and 11, a partition $B$ for which $|B^-| \neq 2$ is C-equivalent only to partitions $D$ for which $|D^-| = |B^-|$. By Observation 4, a partition $B$ for which $|B^-| = 2$ is C-equivalent to a partition $D$ for which $|D^-| = 1$. In turn, such $D$ is C-equivalent only to partitions $E$ with $|E^-| = 1$ or 2. Consequently,

$$H_n^{(1)} = \sum_{\substack{0 \leq k \leq n-1 \\ k \neq 1}} H_{n,k}^{(1)},$$

where $H_{n,k}^{(1)}$ is the number of classes containing $x_1$-aligned partitions of size $n+1$ with $k$ cuts below the principal cut.

By Lemmas 10 and 11, these classes are in 1-to-1 correspondence with ordered pairs $(\mathcal{C}_1, \mathcal{C}_2)$ of classes of resp. size $k+1$ and $n-k$, where $\mathcal{C}_1$ (if nontrivial) is *not* $x_1$-aligned. By Lemma 6, the number of choices for $\mathcal{C}_1$ is $\frac{d-1}{d}H_k$ (for $k \geq 2$). Therefore,

$$H_{n,k}^{(1)} = \begin{cases} H_0 H_{n-1}, & \text{if } k = 0; \\ \frac{d-1}{d} H_k H_{n-1-k}, & \text{otherwise.} \end{cases}$$

This yields, for $n \geq 2$,

$$H_n = d \cdot \left( H_0 H_{n-1} + \frac{d-1}{d} \sum_{k=2}^{n-1} H_k H_{n-1-k} \right) =$$

$$= dH_{n-1} + (d-1) \sum_{k=0}^{n-1} H_k H_{n-1-k} - (d-1)(H_{n-1} + H_{n-2}),$$

which implies
$H(t) = \frac{1-t+(d-1)t^2 - \sqrt{(1-t+(d-1)t^2)^2 - 4(d-1)t(1-(d-1)t)}}{2(d-1)t}$.

As above, Ackerman et al. [1] proved that the generating function for the number of B-equivalence classes of $d$-dimensional guillotine partitions is $G(t) = \frac{1-t-\sqrt{(1-t)^2-4(d-1)t}}{2(d-1)t}$. Substituting $t := t(1-(d-1)t)$ and comparing this with the formula for $H(t)$, we obtain $H(t) = (1-(d-1)t)\, G(t(1-(d-1)t))$. Expanding this expression, we obtain that $H_n$ may be expressed in terms of $G_n$:

$$H_n = \sum_{i=0}^{\lfloor (n+1)/2 \rfloor} (-1)^i (d-1)^i \binom{n+1-i}{i} G_{n-i},$$

as claimed. □

## 4 Asymptotics

A standard singularity analysis [5] yields the asymptotic behavior ($d$ is fixed, $n \to \infty$) $G_{d,n} \sim \ell \lambda^n n^{-3/2}$, where $\ell = \ell(d) = \frac{\sqrt[4]{d(d-1)}\sqrt{\lambda}}{2(d-1)\sqrt{\pi}}$ and $\lambda = \lambda(d) = 2d - 1 + 2\sqrt{d(d-1)}$ (this form also appears in [1]); and $H_{d,n} \sim m\mu^n n^{-3/2}$, where $m = m(d) = \frac{\sqrt[4]{\eta d(d-1)}\sqrt{1+\sqrt{\eta}}}{2(d-1)\sqrt{2\pi\varphi}}$ and $\mu = \mu(d) = \frac{1+\sqrt{\eta}}{2\varphi}$, in which $\varphi = \varphi(d) = 2d - 1 - 2\sqrt{d(d-1)}$ and $\eta = \eta(d) = 8(d-1)\sqrt{d(d-1)} - 3 + 12d - 8d^2$. As $d$ tends to infinity, $\lambda$ and $\mu$, the growth rates of $G_{d,n}$ and $H_{d,n}$, are similar to $4d - 2 - O(1/d)$ and $2d + \sqrt{2}d - 1 - O(\sqrt{1/d})$, respectively.

## References

[1] E. Ackerman, G. Barequet, R.Y. Pinter, and D. Romik, The number of guillotine partitions in $d$ dimensions, *Inf. Processing Let.*, **98** (2006), 162–167.

[2] A. Asinowski, G. Barequet, M. Bousquet-Mélou, T. Mansour, and R.Y. Pinter, Orders induced by segments in floorplan partitions and (2-14-3, 3-41-2)-avoiding permutations, submitted.

[3] A. Asinowski and T. Mansour, Separable $d$-permutations and guillotine partitions, *Annals of Combinatorics*, **14** (2010), 17–43.

[4] M. Cardei, X. Cheng, X. Cheng, and D.-Z. Du, A tale on guillotine cut, *Proc. Novel Approaches to Hard Discrete Optimization*, Waterloo, Canada, 2001, 41–54.

[5] P. Flajolet and R. Sedgewick, *Analytic Combinatorics*, Cambridge University Press, 2009.

[6] J.S.B. Mitchell, Guillotine subdivisions: Part II—A simple polynomial-time approximation scheme for geometric $k$-MST, TSP, and related problems, *SIAM J. on Computing*, **28** (1999), 1298–1309.

[7] L.J. Stockmeyer, Optimal orientations of cells in slicing floorplan designs, *Information and Control*, **57** (1983), 91–101.

[8] B. Yao, H. Chen, C.K. Cheng, and R. Graham, Revisiting floorplan representations, *Proc. Int. Symp. on Phys. Design*, Sonoma, CA, 2001, 138–143.

# Approximating Weighted Geodesic Distances on 2-Manifolds in $\mathbb{R}^3$

Christian Scheffer
Department of Computer Science,
University of Münster, Einsteinstr. 62,
48149 Münster, Germany
christian.scheffer@uni-muenster.de

Jan Vahrenhold
Department of Computer Science,
University of Münster, Einsteinstr. 62,
48149 Münster, Germany
jan.vahrenhold@uni-muenster.de

## Abstract

We present the first algorithm for approximating weighted geodesic distances on 2-manifolds in $\mathbb{R}^3$. Our algorithm works on an weighted $\varepsilon$-sample $S$ of the underlying manifold and computes approximate geodesic distances between all pairs of points in $S$. The approximation error is *multiplicative* and depends on the density of the sample. The algorithm has a running time of $\mathcal{O}(|S|^{2.25}\log^2|S|)$ and an optimal space requirement of $\mathcal{O}(|S|^2)$; the approximation error is bound by $1 \pm \mathcal{O}(\varepsilon)$.

## 1 Introduction

Computing shortest paths in two and three dimensions is one of the fundamental problems in Computational Geometry. In this paper, we consider a variant of this problem, namely to compute all-pairs geodesic distances on a weighted 2-manifold in three-dimensional space, i.e., on a 2-manifold $\Gamma \subset \mathbb{R}^3$ together with a continuous weight function $w : \Gamma \to \mathbb{R}^+$.

**Definition 1** *For a 2-manifold $\Gamma$ and a continuous weight function $w$, the weighted geodesic distance $L^w_\Gamma(x,y)$ between $x, y \in \Gamma$ is the minimal weighted lengths of all connecting curves on the surface, i.e., $L^w_\Gamma(x,y) := \min_f \int_f w(\xi)d\xi$, where $f$ is a curve on $\Gamma$ connecting $x$ and $y$.*

Despite a variety of approaches, exactly solving the weighted shortest path problem in $\mathbb{R}^3$ still remains wide open—see [2] and the references therein—and even for the case of a weighted *polyhedral* surface, only approximation algorithms are known [2].

The surfaces we consider are represented by a discrete subset of sample points. To be able to build on previous results on surface reconstruction and approximation, we require the quality of this subset to be related to the *local feature size* of $\Gamma$. The local feature size is represented by the function $\mathit{lfs} : \Gamma \to \mathbb{R}^+$ which maps each point on $\Gamma$ to its distance to the medial axis of $\Gamma$, the closure of all points having at least two nearest neighbors on $\Gamma$. In line with previous approaches to surface reconstruction, the input then is required to be an $\varepsilon$-sample:

**Definition 2** *A discrete subset $S$ of a smooth 2-manifold $\Gamma \subset \mathbb{R}^3$ is an $\varepsilon$-sample of $\Gamma$ for some $\varepsilon > 0$ if, for each point $x \in \Gamma$, there is a sample point $s \in S$ with $|xs| \leq \varepsilon \cdot \mathit{lfs}(x)$.*

As usual [1, 3, 4, 5, 7], we assume that $\varepsilon$ is sufficiently small; in particular, we require $\varepsilon \leq 1/22$ [8].

## 2 Description of the Algorithm

From a high-level point-of-view, our algorithm for approximating weighted geodesic distances closely resembles our algorithm for the unweighted case [8]: first reconstruct a polyhedral approximation of $\Gamma$ and then use the results of shortest-path calculations on this object to approximate the geodesic distances on $\Gamma$. However, there are two non-trivial obstacles to overcome: first of all, in contrast to the unweighted case, there is no efficient exact algorithm for computing distances on weighted polyhedral surfaces and thus we have to rely on the best approximation algorithm [2]. This algorithm needs an input parameter $\delta$ controlling its approximation quality, and this implies that this parameters needs to depend on $\varepsilon$ if the approximation error of our algorithm shall be bounded by any function involving $\varepsilon$—see [8, Sec. 4]. Unfortunately, the exact value of $\varepsilon$ is not available as part of the input; both in theory (Definition 2) and in practice (data coming from scanning devices) only an upper bound is known and needed. The second issue is that truthfully reconstructing a *weighted* surface $\Gamma$ from a discrete sample $S$ requires that $S$ not only captures folding and local curvature (this can be enforced by requiring $S$ to be an $\varepsilon$-sample) but also the (gradients of the) weight of $\Gamma$. Otherwise, an approximation of the weighted length of a geodesic could differ arbitrarily from the exact value—see Figure 1.

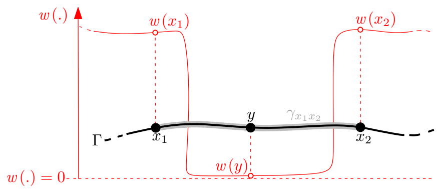


Figure 1: Unweighted and weighted distances.

To resolve the second issue we extend the definition of the local feature size to the weighted case. Recall that in the unweighted case the local feature size is used to bound the Euclidean distance between a point $x \in \Gamma$ and a point $s \in S$. As can be seen in Figure 1, we need to also bound the difference in weight between a point $x \in \Gamma$ and a point $s \in S$. As a first step, we bound the aperture of a cone with apex at $x \in \Gamma$ by setting $m_x := \sup_{x \neq v \in \Gamma} \{|w(v) - w(x)|/|xv|\}$. If $m_x = 0$, i.e., if $w$ is constant, we set $\Delta_w(x) := \mathit{lfs}(x)$, otherwise we set $\Delta_w(x) := w(x)/m_x$. We can prove:

**Observation 1** *The weights as measured by $\Delta_w$ are upper-bounded locally: for $x_1, x_2 \in \Gamma$ and $c > 0$ we have $[|x_1x_2| \leq c \cdot \Delta_w(x_1) \Rightarrow w(x_2) \leq (1+c) \cdot w(x_1)]$*

To be able to also locally lower-bound the weights, we use a standard technique, namely to compute a Lipschitziation of $\Delta_w$. More formally, we define $\nabla_w(x) := \inf_{y \in \Gamma} \{\Delta_w(y) + |xy|\}$. Then, we have $\nabla_w(x) \leq \Delta_w(x)$ and can prove:

**Observation 2** *The weights as measured by $\nabla_w$ are lower-bounded locally: for $x_1, x_2 \in \Gamma$, $c > 0$, we have $[|x_1x_2| \leq c \cdot \nabla_w(x_1) \Rightarrow w(x_2) \leq (1+c) \cdot w(x_1)$ and $w(x_1) \leq (1 + c/(1-c)) \cdot w(x_2)]$.*

Using the above definitions, we are ready to define a local feature size for the weighted case and a corresponding sampling condition:

**Definition 3** *For a smooth 2-manifold $\Gamma \subset \mathbb{R}^3$ and a continuous weight function $w : \Gamma \to \mathbb{R}$, we define the weighted local feature size as $\mathit{lfs}_w(x) := \min\{\nabla_w(x), \mathit{lfs}(x)\}$. A discrete subset $S$ of $\Gamma$ is a weighted $\varepsilon$-sample for some $\varepsilon > 0$ if, for each point $x \in \Gamma$, there is an $s \in S$ with $|xs| \leq \varepsilon \cdot \mathit{lfs}_w(x)$.*

**Observation 3** *The weighted local feature size is 1-Lipschitz and each weighted $\varepsilon$-sample is an $\varepsilon$-sample.*

**Outline of the Approach** As mentioned above, the new algorithm follows our template for the unweighted case [8]: For each $s \in S$, we first reconstruct an approximation of $\Gamma$ and then compute (approximate) single-source shortest paths from $s$ on this surface. Just as in the unweighted case, however, we first need to decimate the point set in a way that maintains the properties of a weighted $\varepsilon$-sample but also guarantees that it is not too dense (and a reconstruction thus too costly). These points are re-inserted (by projecting them onto the reconstruction) prior to computing the shortest paths. In a nutshell, we first construct a so-called control function $\psi$ with the following (non-trivial) properties:

- $\psi$ is $\frac{1}{18}$-Lipschitz.
- For each $s \in S$: $\psi(s) \leq 1.19 \cdot \varepsilon \cdot \mathit{lfs}_w(s)$.

These properties are then used to show that for each $s \in S$ the intersection of $\Gamma$ with the Voronoi cell $\mathit{Vor}_S(s)$ is contained in the ball $B_{\psi(s)}(s)$ with radius $\psi(s)$ centered at $s$. This leads to Algorithm 1:

**Algorithm 1** Compute subsample $S^{uni} \subset S$ [7, 8].

```
Require: $S$
1: $S^{uni} \leftarrow \emptyset$
2: while $S \neq \emptyset$ do
3:     $s \leftarrow$ arbitrary from $S$
4:     $S^{uni} \leftarrow S^{uni} \cup \{s\}$
5:     $S \leftarrow S \setminus (S \cap B_{\psi(s)})$
6: return $S^{uni}$
```

For each decimated point set $S^{uni}$ constructed this way we can prove the following:

**Lemma 1** *For each $x \in \Gamma$, there exists a point $s' \in S^{uni}$ such that $|xs'| \leq \left(2 + \frac{1}{18}\right) \cdot \psi(s')$. Furthermore, $S^{uni}$ is a weighted $\varepsilon'$-sample for $\varepsilon' \leq 3 \cdot \varepsilon$.*

We now sketch how to compute asymptotic bounds for $\varepsilon \cdot \mathit{lfs}$ and $\mathit{lfs}$ and thus how resolve the first issue, i.e., to approximate $\varepsilon$.

**Computation of a Control Function** To compute the control function $\psi$ used in Algorithm 1 we follow exactly the same approach as in the unweighted case: we use a simplified, quadratic-time version of an algorithm by Funke and Ramos [7] to compute the Voronoi diagram of $S$ and, based upon the properties of its cells, for each sample point $s \in S$ an approximation $\widetilde{T}_s$ of the plane $T_s$ tangent to $\Gamma$ in $s$. This leads to the following definitions:

**Definition 4 ([8])** *Let the function $\phi$ be defined as $\phi(s) := 1.002263 \cdot \max_{v \in \widetilde{T}_s \cap \mathit{Vor}_S(s)} |sv|$. Then the control function $\psi$ used in Algorithm 1 can be defined as $\psi(s) := \max_{s' \in S} \left\{\phi(s') - 1.19 \cdot \frac{1}{23} \cdot |ss'|\right\}$.*

The next lemma (Lemma 2 in [3]) is crucial for proving the following two lemmas which ultimately allow to prove that the control function $\psi$ indeed has the properties stated above:

**Lemma 2 ([3])** *(a) On either side of $\Gamma$, the distance from $s$ to its pole is at least $\mathit{lfs}(s)$.*
*(b)* [...]
*(c) Let $v$ be a vertex of $\mathit{Vor}_S(s)$ such that $|vs| \geq \mathit{lfs}(s)$. The angle at $s$ between the vector to $v$ and the normal to the surface is at most $2 \cdot \arcsin(\varepsilon/(2 - 2 \cdot \varepsilon))$.*

We than can show that for each $x \in \mathit{Vor}_S(s) \cap \Gamma$ there is a "witness" $v \in \mathit{Vor}_S(s) \cap \widetilde{T}_s$ and vice versa such that the distance $|sx|$ is approximate the same as $|sv|$; based upon this, we can show (P2) and conclude that the intersection of $\Gamma$ with the Voronoi cell $\mathit{Vor}_S(s)$ is contained in ball $B_{\psi(s)}(s)$.

**Reconstructing a Weighted Surface** Since, by Lemma 1, each decimated point set $S^{\text{uni}}$ is a weighted $\varepsilon'$-sample and thus, by Observation 3, also an $\varepsilon'$-sample, we can build upon the large body of literature on reconstructing (unweighted) smooth surfaces. In particular, we can follow (one of the approaches of) Funke and Ramos [7] and use the CoCone algorithm [4]. The central insight by Funke and Ramos [7, Sec. 6] is that only a small portion of the point set contributes to each co-cone $CC(s)$ used in the reconstruction algorithm. We can prove a similar result:

**Lemma 3** *After $\mathcal{O}(|S|\log|S|)$ preprocessing, we can report in $\mathcal{O}(\log|S|)$ time a constant-size superset of all sample points contributing to $CC(s) \cap Vor_{S^{\text{uni}}}(s)$.*

While Funke and Ramos use a (different) uniformity property of the point sets in question to prove the above result, we do so by analyzing the properties of the reconstruction the unmodified CoCone algorithm would construct:

**Lemma 4** *Let $N$ be the output of the* CoCone *algorithm when run on a decimated point set $S^{\text{uni}}$. Then the following holds for each triangle $\triangle_{s_1s_2s_3} \in N$:*

1. *The circumradius of $\triangle_{s_1s_2s_3}$ is upper-bounded by $1.11 \cdot (2 + \frac{1}{18}) \cdot \psi(s_i)$ for $i = 1, 2, 3$.*
2. *The circumradius of $\triangle_{s_1s_2s_3}$ is upper-bounded by $2.72 \cdot \varepsilon \cdot lfs_w(s_i)$ for $i = 1, 2, 3$.*
3. *The angles of $\triangle_{s_1s_2s_3}$ are lower-bounded by $11°$.*

The last point is of direct relevance to the running time of the shortest path algorithm to be used later on, since its running time depends on the average minimal interior angle of all faces visited. The lower bound shown in Lemma 4 gives an important lower bound for this and thus implies an upper bound for the running time.

The first two points of Lemma 4, however, allow us to prove which points are not needed locally and thus can be ignored safely:

**Lemma 5** *All sample points contributing to $CC(s) \cap Vor_{S^{\text{uni}}}(s)$ lie inside $B_{5\cdot\psi(s)}(s)$.*

Thus, (approximate) range reporting queries [6] can be used (as in [7]) to compute a superset of the sample points contributing to $CC(s) \cap Vor_{S^{\text{uni}}}(s)$:

**Lemma 6** *After $\mathcal{O}(|S|\log|S|)$ preprocessing, we can report a constant-size superset of $S^{\text{uni}} \cap B_{5\cdot\psi(s)}(s)$ in $\mathcal{O}(\log|S|)$ time.*

We can also prove that $B_{5\cdot\psi(s)}(s)$ contains a vertex of the triangle containing $s$'s nearest point $\nu_s$ on the surface of $N$ (note that $\nu_s = s \iff s \in S^{\text{uni}}$). This implies that Lemma 6 and a standard packing argument can be used to prove that we can efficiently "re-insert" each $s \in S \setminus S^{\text{uni}}$ by identifying it with $\nu_s$ in overall $\mathcal{O}(|S|\log|S|)$ time.

**Approximate Weighted Shortest Path Queries** As mentioned above, the most efficient algorithm to answer weighted shortest path queries is the $(1 \pm \delta)$-approximation algorithm by Aleksandrov *et al.* [2]. This algorithm, however, requires the approximation parameter $\delta$ as part of the input. Since we aim at an approximation quality depending on the unknown parameter $\varepsilon$, we have to overcome the obstacle of not having an exact value of $\delta$ to pass to the shortest path algorithm. We can pass, however, an approximation $\widetilde{\varepsilon}$ of $\varepsilon$ which we define using the concept of a pole:

**Definition 5 ([5])** *The poles $p_s^-$ and $p_s^+$ are the two vertices of its Voronoi cell farthest from $s$, one on either side of the surface.*

Aichholzer *et al.* [1, Lemma 5.1] proved that an asymptotic upper bound for the local feature size of a sample point $s \in S$ is given by the distance of $s$ to its nearest pole $p_s$.

**Lemma 7 ([1])** $|p_s s| \geq 1.2802^{-1} \cdot lfs(s)$.

Based upon this lemma, we are ready to define the approximate sampling density of an $\varepsilon$-sample:

**Definition 6** *Let $\widetilde{lfs}(s) := |p_s s|$ be the approximation of the local feature size as defined by Aichholzer et al. [1]. Then, the approximate sampling density $\widetilde{\varepsilon}$ is defined as $\widetilde{\varepsilon} := \max_{s\in S}\{\psi(s)/\widetilde{lfs}(s)\}$.*

By definition, the local feature size is an upper bound for the weighted local feature size. Since $\psi$ is an asymptotic lower bound for $\varepsilon \cdot lfs_w$ and $\widetilde{lfs}$ is an asymptotic upper bound for $lfs$, Lemma 8 follows:

**Lemma 8** $\widetilde{\varepsilon} \leq 1.53 \cdot \varepsilon$

We now can pass $\delta := \widetilde{\varepsilon}$ as a parameter to the $(1 \pm \delta)$-approximation algorithm by Aleksandrov *et al.* [2] and compute shortest paths on $N$ between all points in $S^{\text{uni}} \cup \{\nu_s | s \in S \setminus S^{\text{uni}}\}$.

## 3 Analysis of the Algorithm

For the analysis of the approximation quality, we first extend a lemma from the unweighted case [8, Lemma 20] to the following "weighted" version:

**Lemma 9** *Let $x_1$ and $x_2$ be two arbitrary points on $\Gamma$ with $|x_1x_2| \leq \mathcal{O}(1)\cdot\varepsilon\cdot\max\{lfs_w(x_1), lfs_w(x_2)\}$. Then we have $(1+\mathcal{O}(\varepsilon)) \cdot L_\Gamma^w(x_1,x_2) \geq w(x_1)\cdot|x_1x_2| \geq (1-\mathcal{O}(\varepsilon))\cdot L_\Gamma^w(x_1,x_2)$.*

**Definition 7** *Let $x_1, x_2 \in \mathbb{R}^3$ be two points and $\alpha < \frac{\pi}{2}$ an angle. Let $C_{x_1}$ be the cone defined by $\{x \in \mathbb{R}^3 | \angle(x_2x_1x) \leq \alpha\}$ and let $C_{x_2}$ be the cone defined by $\{x \in \mathbb{R}^3 | \angle(x_1x_2x) \leq \alpha\}$. The (linear)* bicone *$D_{x_1,x_2,\alpha}$ then is defined as $C_{x_1} \cap C_{x_2}$, see Figure 2.*

**Algorithm 2** Approximating weighted distances.

```
Require: Weighted $\varepsilon$-sample $S$ of 2-manifold $\Gamma \subset \mathbb{R}^3$
 1: Compute control function $\psi$.                                ▷ [8]
 2: for all $s \in S$ do
 3:     Compute nearest pole $p_s$ and $\widetilde{lfs}(s) := |p_s s|$.
 4: Compute $\widetilde{\varepsilon} := \max_{s\in S}\{\psi(s)/\widetilde{lfs}(s)\}$.
 5: for all $s \in S$ do
 6:     Compute decimated set $S^{uni}$ s.t. $s \in S^{uni}$.
 7:     Preprocess $S^{uni}$ for approximate $(1+\frac{1}{10})$-range
        reporting queries.                                          ▷ [6]
 8:     Compute superset of $B_{5\cdot\psi(s)}(s) \cap S^{uni}$.       ▷ [6, 7]
 9:     Compute reconstruction $N$ of $\Gamma$.                        ▷ [4, 7]
10:     for all $\triangle \in N$ do
11:         $w(\triangle) := w($arbitrary vertex of $\triangle)$
12:     Preprocess data structure for $(1 \pm \widetilde{\varepsilon})$-
        approximate weighted single-source shortest
        path (SSSP) queries w.r.t. $s$ on $N$.                       ▷ [2]
13:     for all $s' \in S$ do
14:         $\nu_{s'}$ := point on $N$ nearest to $s'$.                  ▷ [6]
15:         $\widetilde{L}^w_N(\nu_s, \nu_{s'})$ := result of $(1 \pm \widetilde{\varepsilon})$-approxi-
            mate weighted SSSP query on $N$.                       ▷ [2]
16: return $\widetilde{L}^w_N$                 ▷ Approximate distances on $N$.
```

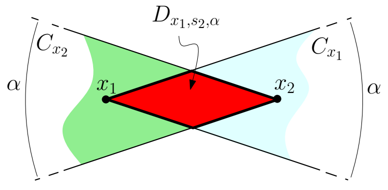


Figure 2: Definition of the (linear) bicone $D_{x_1,x_2,\alpha}$

Using a result by Amenta *et al.* [3, Lemma 1] ("For any two points $p$ and $q$ on $\Gamma$ with $|pq| \leq \rho \cdot \min\{lfs(p), lfs(q)\}$, where $\rho \leq \frac{1}{3}$, the angle between the normals to $\Gamma$ at $p$ and $q$ is at most $\frac{|pq|}{(1-3\cdot\rho)\cdot lfs(p)}$."), we then can show that for each edge $e = (s_1, s_2)$ of a triangle in the reconstruction $N$ there exists a curve $\gamma_{s_1 s_2} \subset \Gamma \cap D_{s_1,s_2,6\cdot\varepsilon'}$. We then use that the triangles examined by the CoCone-algorithm in the reconstruction $N$ of $\Gamma$ lie "flat" w.r.t. the planes tangent to $\Gamma$ in the triangles' vertices:

**Theorem 10 (Theorem 11 in [4])** *The normal to each triangle makes an acute angle of no more than* $\sin^{-1}\left(\frac{1.15\cdot\varepsilon}{1-\varepsilon}\right) + \sin^{-1}\left(\frac{2}{\sqrt{3}} \cdot \sin\left(2 \cdot \sin^{-1}\left(\frac{1.15\cdot\varepsilon}{1-\varepsilon}\right)\right)\right) \in \mathcal{O}(\varepsilon)$ *with the surface normal at the vertex subtending the triangle's largest interior angle.*

Theorem 10 allows us to prove the approximation quality of the approximate distances $\widetilde{L}^w_N$:

**Lemma 11** *For any* $s_1, s_2 \in S$, *we have* $(1 - \mathcal{O}(\varepsilon)) \cdot L^w_\Gamma(s_1, s_2) \leq \widetilde{L}^w_N(\nu_{s_1}, \nu_{s_2}) \leq (1 + \mathcal{O}(\varepsilon)) \cdot L^w_\Gamma(s_1, s_2)$

For the analysis of the running time of Algorithm 2, we can prove the following non-trivial lemma:

**Lemma 12** $\frac{1}{\varepsilon^2} \in \mathcal{O}(|S|)$.

The proof of Lemma 12 uses the next two lemmas:

**Lemma 13** $\forall s \in S : \forall v \in Vor(s) \cap \Gamma : |vs| \leq \widetilde{\varepsilon} \cdot \widetilde{lfs}(s)$

**Lemma 14 (Aichholzer *et al.* [1]: Lemma 4.2)** *Let* $p^\star_s$, $\star \in \{+, -\}$, *be a pole with polar radius* $\rho^\star_s$. *The surface* $\Gamma$ *cannot get closer to* $p$ *than:*

$$\rho^\star_s \cdot \left(\sqrt{1 - 4 \cdot \left(\varepsilon^2 - \frac{\varepsilon^4}{4}\right)} - \varepsilon^2\right) \geq \rho^\star_s \cdot (1 - 3 \cdot \varepsilon^2 - \mathcal{O}(\varepsilon^4)).$$

Algorithm 2 spends $\mathcal{O}(|S|^2)$ time on Steps 1–5. Except for Step 14 which takes time $\mathcal{O}\left(\frac{|S|}{\sqrt{\widetilde{\varepsilon}}} \log \frac{|S|}{\widetilde{\varepsilon}} \log \frac{1}{\widetilde{\varepsilon}}\right)$, all steps inside the for-loop run in time $\mathcal{O}(|S| \log |S|)$.

**Theorem 15** *Algorithm 2 approximates weighted geodesic distances between all pair of points in a weighted* $\varepsilon$*-sample* $S$ *of a weighted 2-manifold in* $\mathbb{R}^3$ *in* $\mathcal{O}\left(|S|^{2.25} \cdot \log\left(|S|^{1.25}\right) \cdot \log\left(|S|^{0.25}\right)\right) = \mathcal{O}\left(|S|^{2.25} \log^2 |S|\right)$ *time up to a multiplicative approximation error of* $1 \pm \mathcal{O}(\varepsilon)$.

## References

[1] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Kornberger, S. Plantinga, G. Rote, A. Sturm, and G. Vegter. Recovering Structure from $r$-Sampled Objects. *Computer Graphics Forum*, 28(5):1349–1360, 2009.

[2] L. Aleksandrov, H. N. Djidjev, H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack. Algorithms for approximate shortest path queries on weighted polyhedral surfaces. *Discrete & Computational Geometry*, 44(4):762–801, 2010.

[3] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[4] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *International Journal of Computational Geometry and Applications*, 12(1–2):125–141, 2002.

[5] N. Amenta, S. Choi, and R. K. Kolluri. The power crust, union of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19(2–3):127–153, 2001.

[6] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[7] S. Funke and E. A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proc. Thirteenth Symp. on Discrete Algorithms*, pages 781–790. 2002.

[8] C. Scheffer and J. Vahrenhold. Approximating geodesic distances on 2-manifolds in $R^3$. *Computational Geometry: Theory & Applications*, 2012. In press. `http://dx.doi.org/10.1016/j.comgeo.2012.05.001`.

# Efficient volume and edge-skeleton computation for polytopes defined by oracles [*]

Ioannis Z. Emiris[†] Vissarion Fisikopoulos[†] Bernd Gärtner[‡]

## Abstract

We design and implement total polynomial-time algorithms for computing the exact edge-skeleton and for approximating the volume of polytopes given by optimization oracles. That is, the time complexity of the algorithm is bounded by a polynomial in the input and the output size. The main algorithmic step is to obtain efficient separation oracles given an optimization oracle, which is reduced to solving a linear program in the polar polytope. This separation oracle is used to yield polynomial-time Monte Carlo algorithms for approximating the volume of the polytope. Next, we use this separation oracle to derive the first total polynomial-time algorithm for the edge skeleton of the polytope, when we are also given a superset of the polytope's edges, with cardinality bounded by a polynomial in the number of those edges. Finally, we briefly discuss our implementation and experimental results of optimization and volume approximation algorithms, based on random walks.

## 1 Introduction

Convex polytopes in general dimension admit a number of alternative representations. The best known, explicit representations for a polytope $P$ is either as the set of its vertices (possibly with additional information about the positive-dimensional faces), or as a bounded intersection of halfspaces. In the latter case, a linear programming problem (LP) on $P$ consists in finding a vertex of $P$ that maximizes the inner product with a given objective vector $c$. In this paper we study the case where a polytope is given by an implicit representation. That is, the only allowable access to $P$ is a black box subroutine (oracle) that solves the LP problem on $P$ for a given vector $c$. Then, we say that $P$ is given by an *optimization*, or *LP*, or *vertex* oracle. Given such an oracle, the entire polytope can be reconstructed and its explicit representation can be found using the Beneath-Beyond method [2]. This is implemented in [8, 4] for a special case of polytopes, called resultant polytopes.

Another important implicit representation for a polytope $P$ is to be given by a *separation* oracle. That is, given a point $x$ the oracle returns yes if $x \in P$ or a hyperplane that separates $P$ from $x$ otherwise. To acquire an optimization oracle for $P$, one has to solve a linear program over $P$, using the separation oracle. This can be done by (variants of) the ellipsoid method (Sect. 2).

Now we pose the opposite (dual) question. Given an optimization oracle for a polytope $P$, compute a separation oracle for $P$. This boils down to solving a linear program in the polar dual space where the optimization oracle of $P$ is a separation oracle for the polar polytope $P^*$ (Sect. 2).

**Proposition 1** *An approximate separation oracle for a well-rounded polytope $P \subseteq \mathbb{R}^n$, given by an optimization oracle of complexity $T$, is computed in time $O^*(nT+n^{3.38})$, where $O^*(\cdot)$ hides polylog factors in the argument.*

Well-rounded means that the radii of some bounding and some inscribed ball differ by a constant factor not depending on $n$. Also, the approximation error is assumed to be constant, so that the bound is simple and only depends on $n$ and $T$. Prop. 1 is the main algorithmic tool used in the sequel. This leads to our first contribution, namely an implementation of a linear program solver based on the randomized algorithm of [1] (Sect. 4), which is also valuable because of its independent interest.

Regarding the *volume* computation problem, it is known that the exact computation is hard. However, randomized poly-time approximation algorithms exist when the polytope is given by a separation oracle, and currently the best complexity is $O^*(n^4)$ oracle calls [13]. The literature on implementing such algorithms is limited. A notable exception is [12], which implements [13] and computes the volume of $n$-cubes, $n = 2, 5, 8$, in $807, 1901, 7551$ secs respectively. Our second contribution consist in providing an implementation, based on the $O^*(n^5)$ algorithm of [9], which is simpler and appears to have competitive performance

*Work partially supported from project "Computational Geometric Learning", which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the 7th Framework Programme for research of the European Commission, under FET-Open grant number: 255827. We also thank K. Fukuda, C. Müller, S. Stich for discussions and bibliographic suggestions.

†Department of Informatics and Telecommunications, University of Athens, Greece. {emiris,vfisikop}@di.uoa.gr

‡Institut für Theoretische Informatik, Swiss Federal Institute of Techology (ETHZ), Zurich, Switzerland. gaertner@inf.ethz.ch

(Sect. 4). Regarding volume computation for polytopes given by optimization oracle, Prop. 1 is used in conjunction with the above randomized algorithm.

Edge skeleton computation and vertex enumeration from edge directions and suitable oracles is a problem of independent interest. It has been studied before [14]; their solution is used as a subroutine in efficiently solving convex integer programming in fixed dimension [11]. Our third contribution is the design of total poly-time algorithms for computing the exact edge-skeleton of polytopes given by optimization oracles, when we are also given a superset of the polytope's edges, with cardinality bounded by a polynomial in the number of those edges.

**Applications.** Two applications have motivated this paper. The first is the Minkowski sum $P \subset \mathbb{R}^n$ of polytopes $P_1, \dots, P_r \subset \mathbb{R}^n$ which are given by the set of their vertices. Here, optimization oracles are naturally and easily available, whereas it is not straightforward to construct the separation oracles. To illustrate this assume we are given a direction. Then, the extremal vertex of each polytope summand can be efficiently computed by computing the interior product of the direction with each vertex of the summand. The extremal vertex of $P$ towards the given direction is the sum of the extremal vertices of the summands. The above can be generalized for *summand polytopes* given by optimization oracles.

**Proposition 2** *If $P_1, \dots, P_r$ are given by optimization oracles, each of complexity bounded by $v$, then by Prop. 1, a separation oracle for $P = \sum_{i=1}^r P_i$ is computed in $O^*(nrv + n^{3.38})$. The edge skeleton of $P$ can be computed in $O^*(m^2 n^{3.5})$ and an approximation of its volume in $O^*(n^{7.38})$.*

Our second application is resultant polytopes. Resultant polynomials are fundamental in algebraic geometry since they generalize determinants to nonlinear systems [6]. The Newton polytope of resultant $R$, or *resultant polytope*, is the convex hull of the exponent vectors corresponding to nonzero terms. A resultant is defined for $d+1$ pointsets in $\mathbb{Z}^d$. If $R$ lies in $\mathbb{Z}^n$, the total number of input points is $n + 2d + 1$, and we assume that they are in generic position. If $m$ is the number of vertices in $R$, typically $m \gg n \gg d$, so $d$ is assumed fixed. A poly-time optimization oracle is described in [4]. This approach can be used for the secondary and discriminant polytopes [6].

**Proposition 3** *Given an optimization oracle for $R \subset \mathbb{Z}^n$ we can compute the edge-skeleton of $R \subset \mathbb{Z}^n$ in $O^*(m^3 n^{\lfloor (d/2)+1 \rfloor} + m^4 n)$ for input points in generic position, and an approximation of its volume in $O^*(n^{\lfloor (d/2)+5 \rfloor})$, where $d > 5$.*

## 2 Polytope oracles

We introduce all tools needed to prove Prop. 1. Following [7], we define 5 basic oracles for polytope $P \subset \mathbb{R}^n$ and, for completeness, describe *exact poly-time* procedures that connect them.
*Optimization* ($OPT_P(c)$): Given $c \in \mathbb{R}^n$, find $x \in P$ maximizing $c^T x, x \in P$, or assert $P = \emptyset$.
*Validity* ($VAL_P(c)$): Given $c \in \mathbb{R}^n$, decide whether $c^T x \leq 1$ holds for all $x \in P$.
*Violation* ($VIOL_P(c)$): Given $c \in \mathbb{R}^n$, call $VAL_P(c)$; if negative, find $y \in P \ : \ c^T y > 1$.
*Membership* ($MEM_P(y)$): Given $y \in \mathbb{R}^n$ decide whether $y \in P$.
*Separation* ($SEP_P(y)$): Given $y \in \mathbb{R}^n$ call $MEM_P(y)$. If it answers negatively, find the normal to a hyperplane that separates $y$ from $P$; i.e. $c \in \mathbb{R}^n \ : \ c^T y > \max\{c^T x \mid x \in P\}$.

We use the *polar dual polytope* of $P$ in dual space $(\mathbb{R}^n)^*$, as defined in e.g. [16]:

$$P^* := \{c \in \mathbb{R}^n : c^T x \leq 1, \text{ for all } x \in P\} \subseteq (\mathbb{R}^n)^*,$$

where we assume that the origin $\mathbf{0} \in int(P)$, the relative interior of $P$, i.e. $\mathbf{0}$ is not contained in any face of $P$ of dimension $< n$. This hypothesis can be ensured by an appropriate affine translation.

It is easy to see that having $OPT_P$ we can derive $VIOL_P$ and then $VAL_P$. Similarly, having $SEP_P$ we can derive $MEM_P$. For a polytope $P \subseteq \mathbb{R}^n$, its polar $P^* \subseteq (\mathbb{R}^n)^*$ and $c \in \mathbb{R}^n$, it holds $VIOL_P(c) = SEP_{P^*}(c^T)$ hence $VIOL_{P^*}(c^T) = SEP_P(c)$ [16, Thm 2.11].

Given $VIOL_P(c)$ we compute $OPT_P(c)$ by performing binary search on the value of $c^T x$, for $x \in P$. That is, we test feasibility, or non-emptiness, of $\{x \in P \ : \ c^T x \leq c_0\}$ for various constants $c_0$, by calling $VIOL(c)$ on suitably chosen translations of $P$.

Let $B(\rho)$ denotes the $n$-ball of radius $\rho$ centered at the origin. Assume that $P \subseteq B(\rho)$ and $B(r) \subseteq P$ if $P$ is not empty. Then, let $L$ denote the log-ratio of the bounding balls of $P$, i.e. $L = \lg(\rho/r)$. Computing $VIOL_P$ from $SEP_P$ is a fundamental question. Some poly-time algorithms for this problem are the ellipsoid method [10] with complexity $O(n^2 L T_S + n^4 L)$ that [15] improves to

$$O(nLT_S + n^{3.38} L) = O^*(nT_S + n^{3.38}) \qquad (1)$$

and the randomized algorithm of [1] which runs in $O(nLT_S + n^7 L)$, where $T_S$ is the complexity of the separation oracle.

For proving Prop. 1 we have to compute a separation oracle for a polytope $P$, $SEP_P$, given an optimization oracle for $P$, $OPT_P$. As it is explained above this can be done by solving a linear program in the polar dual space. The fastest algorithm used to solve this linear program is [15], with complexity of

Eq. (1). However, we implement the slightly slower but simpler algorithm of [1].

## 3 Computing the edge skeleton

We are implicitly given a polytope $P \subseteq \mathbb{R}^n$ via an optimization oracle $OPT_P(x)$ of $P$ and we are explicitly given a superset $E$ of all edge directions of $P$, i.e.

$$E \supseteq D(P) := \{v - w : v, w \text{ adjacent vertices of } P\},$$

with cardinality $|E|$ bounded by a polynomial in $|D(P)|$. The goal is to efficiently compute the edge skeleton of $P$, i.e. its vertices and the edges connecting the vertices. Even if $E = D(P)$, this set does not in general provide enough information to perform the task, so we need additional information about $P$; here we assume an optimization oracle.

**Proposition 4** [14] *Let $P \subseteq \mathbb{R}^n$ be a polytope given by an optimization oracle $OPT_P(c)$, and let $E \supseteq D(P)$ be a superset of the edge directions of $P$, With $O(|E|^{n-1})$ arithmetic operations and $O(|E|^{n-1})$ calls to $OPT_P(c)$, all vertices of $P$ can be computed.*

If $P$ has $m$ vertices, then $|D(P)| \leq \binom{m}{2}$, and this is tight for neighborly polytopes in general position. This means that the bound of Prop. 4 is $\Theta\left(m^{2n-2}\right)$ in the worst case.

We improve over this result and compute the edge skeleton with a number of arithmetic operations and calls to $OPT_P(x)$ which are polynomial in $m, n$, and $L^*$, the ratio between the radii of balls enclosing and included in the polar $P^*$. This ratio comes in since we construct a separation oracle $SEP_P(y)$ from $OPT_P(x)$, according to Prop. 1. We therefore get rid of the exponential dependence on $n$ in Prop. 4, but at the cost of an additional dependence on $L^*$ which in general cannot be bounded by $m, n$. On the other hand, $L^*$ is related to $L$.

The algorithm is as follows. Using $OPT_P(c)$, we find the unique vertex $v$ of minimum $x_n$ coordinate. We maintain sets $V_P, E_P$ of vertices and edges that have already been found, along with a priority queue $W$ of the vertices that are in $V_P$. When we process the next vertex $v$ from the queue, it remains to find its incident edges. To find the neighbors of $v$, we first build a set $V_{cone}$ of candidate vertices. $V_{cone}$ can be constructed using $SEP_P(y)$ that we build from $OPT_P(x)$ as described above. In a final step, we remove the candidates that do not yield vertices. For this, we first solve a linear program to compute a hyperplane separating $v$ from the candidates; w.l.o.g. this hyperplane is $\{x_n = 1\}$. Then we compute the extreme points of $C \cap \{x_n = 1\}$, giving us the extremal rays of $C$. Finally, we remove every point from $V_{cone}$ that is not on an extremal ray, or not highest on its extremal ray.

We bound the *time complexity* of the algorithm: We call $OPT_P(x)$ to find the first vertex of $P$. Then, there are $O(m)$ iterations; one iteration calls $SEP_P(y)$ at most $|E|$ times, each call requiring $O(nL^*T + n^{3.38}L^*)$ time, where $T$ is the time to execute $OPT_P(x)$. Then we compute the (at most $m$) extreme points from a set of at most $|E|$ points, which can be done in

$$O(|E| \cdot \mathrm{LP}^*(n+1, m+1) + n|E|m)$$

time and $O(n|E|)$ space [3], where $\mathrm{LP}^*(a, b)$ is the time to solve a linear program with $a$ variables and $b$ inequality constraints.

Assuming $|E| = O(|D(P)|) = O(m^2)$, we obtain the following result, and since $\mathrm{LP}^*(n, m)$ is polynomial in $n, m$ in the bit model [10], an overall polynomial bound follows.

**Theorem 5** *The algorithm runs in time $O(m^3(nL^*T + n^{3.38}L^* + LP^*(n, m)))$, where $T$ is the time to perform one call to $OPT_P(x)$.*

## 4 Implementation and experiments

We implement optimization and volume computation algorithms based on random walks. The *Hit-and-Run* random walk is used to generate uniformly distributed points in $P \subset \mathbb{R}^n$ in $O^*(n^3)$ per point [13]. In a feasibility problem we have to answer if a given polytope $P$ is empty or compute a feasible point in $P$. We implement optimization algorithms based on [1] that solves the feasibility problem. The advantage of this algorithm is its simplicity and the re-usage of procedures, such as random walks, in volume computation.

We also implement randomized approximate volume computation algorithms of polytopes given by separation oracles. Moreover, we implement the algorithm in [9] which approximates of the volume of a polytope $P$ given by a separation oracle by computing uniformly distributed points in $P$. Assuming $B(1) \subseteq P \subseteq B(\rho)$, the algorithm returns an estimation of vol$(P)$, which lies between $(1 - \varepsilon)$vol$(P)$ and $(1 + \varepsilon)$vol$(P)$ with probability $\geq 3/4$, making

$$O\left(\frac{n^4\rho^2}{\varepsilon^2} \ln n \ln \rho \ \ln^2 \frac{n}{\varepsilon}\right) = O^*(n^4\rho^2)$$

oracle calls with probability $\geq 9/10$. Combining the above implementations we also provide an implementation for volume approximation of Minkowski sums. The code is in C++ and is publicly available at `http://sourceforge.net/projects/randgeom`.

We perform an experimental analysis of the above implemented algorithms on an Intel Core i5-2400 3.1GHz, 6MB L2 cache, 8GB RAM, 64-bit Debian GNU/Linux. The optimization algorithms are able to run in less than a minute for up to dimension 11 when

| $n$ | $vol(P)$ | exact(sec) | $r_2$ | $w_s$ | $(1+\varepsilon)vol(P)$ | min | max | $\mu$ | $\sigma$ | appr(sec) |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 0.06 | 2218 | 8 | 6.09 | 3.84 | 4.12 | 3.97 | 0.05 | 0.23 |
| 4 | 16 | 0.06 | 2738 | 7 | 30.4 | 14.99 | 16.25 | 15.59 | 0.32 | 1.77 |
| 6 | 64 | 0.09 | 5308 | 38 | 121.6 | 60.85 | 67.17 | 64.31 | 1.12 | 39.66 |
| 8 | 256 | 2.62 | 8215 | 16 | 486.4 | 242.08 | 262.95 | 252.71 | 5.09 | 46.83 |
| 10 | 1024 | 388.25 | 11370 | 40 | 1945.6 | 964.58 | 1068.22 | 1019.02 | 30.72 | 228.58 |
| 12 | 4096 | – | 14725 | 82 | 7782.4 | 3820.94 | 4247.96 | 4034.39 | 80.08 | 863.72 |

Table 1: Volume computation for hypercubes; '–': the exact method was unable to compute the volume.

| $n$ | $vol(P)$ | exact(sec) | $r_2$ | $w_s$ | min | max | $\mu$ | $\sigma$ | appr(sec) |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 14.00 | 0.01 | 216 | 11 | 12.60 | 19.16 | 15.16 | 1.34 | 119.00 |
| 3 | 45.33 | 0.01 | 200 | 7 | 42.92 | 57.87 | 49.13 | 3.92 | 462.65 |
| 4 | 139.33 | 0.03 | 100 | 7 | 100.78 | 203.64 | 130.79 | 21.57 | 721.42 |
| 5 | 412.26 | 0.23 | 100 | 7 | 194.17 | 488.14 | 304.80 | 59.66 | 1707.97 |

Table 2: Volume computation of the Minkowski sum of a hypercube and a crosspolytope.

tested on hypercubes and their polar duals, namely crosspolytopes. The volume approximation algorithm tested on hypercubes and crosspolytopes compute the volume up to dimension 12 within minutes, whereas it is intractable to compute in more than 10 dimensions with exact methods, such as Polymake [5]. For 20 runs, the code's computed values have less than 2% error from the average one. Additionally, the minimum and maximum computed values bounds the exact volume providing tighter bounds than the theoretical ones, i.e. $(1 \pm \varepsilon)\mathrm{vol}(P)$. Table 1 shows experimental results, where $r_2$ is the number of random points computed, $w_s$ is the number of the steps of a Hit-and-Run random walk, and max, min, $\mu$ and $\sigma$ denote, respectively, the maximum, the minimum, the average and the average absolute deviation of the computed volume approximation.

Finally, we compute an approximation of the volume of the Minkowski sum of a hypercube and a crosspolytope. We perform 10 experiments for each polytope sum. The results show that the min and max computed values bound the exact one. However, we are unable to compute in dimensions higher than 5 since each membership test for $P$ runs one of the optimization algorithms. On the other hand, there is space for improvement in many places of the prototype implementations of optimization and volume computation, which will improve the Minkowski sum volume computation as well. Table 2 shows experimental results with the same notation as in Table 1.

## References

[1] D. Bertsimas and S. Vempala. Solving convex programs by random walks. *J. ACM*, 51(4):540–556, 2004.

[2] Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[3] K.L. Clarkson. More output-sensitive geometric algorithms. In *Proc. IEEE FOCS*, pages 695–702, 1994.

[4] I.Z. Emiris, V. Fisikopoulos, C. Konaxis, and L. Peñaranda. An output-sensitive algorithm for computing projections of resultant polytopes. In *Proc. SoCG*, pages 179–188, 2012.

[5] E. Gawrilow and M. Joswig. Polymake: an approach to modular software design in computational geometry. In *Proc. SoCG*, pages 222–231. ACM, 2001.

[6] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[7] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, New York, 1988.

[8] P. Huggins. ib4e: A software framework for parametrizing specialized LP problems. In A. Iglesias and N. Takayama, editors, *Mathematical Software - ICMS*, volume 4151 of *LNCS*, pages 245–247. Springer, 2006.

[9] R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $\mathrm{O}^*(\mathrm{n}^5)$ volume algorithm for convex bodies. *Random Struct. Algorithms*, 11(1):1–50, 1997.

[10] L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20(1):191–194, 1979.

[11] J.A. De Loera, R. Hemmecke, S. Onn, U.G. Rothblum, and R. Weismantel. Convex integer maximization via Graver bases. *J. Pure & Applied Algebra*, 213(8):1569–1577, 2009.

[12] L. Lovász and I. Deák. Computational results of an $O(n^4)$ volume algorithm. *European J. Operational Research*, 216(1):152–161, 2012.

[13] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $\mathrm{O}^*(\mathrm{n}^4)$ volume algorithm. *J. Comp. Syst. Sci.*, 72(2):392–417, 2006.

[14] S. Onn and U.G. Rothblum. The use of edge-directions and linear programming to enumerate vertices. *J. Combin. Optim.*, 14:153–164, 2007.

[15] P.M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. IEEE FOCS*, pages 338–343, 1989.

[16] G.M. Ziegler. *Lectures on Polytopes.* Springer, 1995.

# Flip Distance Between Triangulations of a Simple Polygon is NP-Complete

Oswin Aichholzer[*] Wolfgang Mulzer[†] Alexander Pilz[‡]

## Abstract

Let $T$ be a triangulation of a simple polygon. A *flip* in $T$ is the operation of removing one diagonal of $T$ and adding a different one such that the resulting graph is again a triangulation. The flip distance between two triangulations is the smallest number of flips that is necessary to transform one triangulation into the other. We show that computing the flip distance between two triangulations of a simple polygon is NP-hard.

## 1 Introduction

Let $P$ be a simple polygon in the plane, that is, the closed region bounded by a piece-wise linear, simple cycle. A *triangulation* $T$ of $P$ is a geometric (straight-line) maximal outerplanar graph whose outer face is the complement of $P$ and whose vertex set are the vertices of $P$. Let $d$ be a diagonal of $P$ whose removal creates a convex quadrilateral $f$. By replacing $d$ with the other diagonal of $f$, we again get a triangulation of $P$. This operation is called a *flip*. The *flip graph* of $P$ is the abstract graph whose vertices are the triangulations of $P$ and in which two triangulations are adjacent if and only if they differ by a single flip. We study the *flip distance*, that is, the minimum number of flips required to transform a given source triangulation into a target triangulation.

Edge flips became popular in the context of Delaunay triangulations. Lawson [6] proved that the flip graph is connected with diameter $O(n^2)$. Hurtado, Noy, and Urrutia [5] gave an example where the flip distance is $\Omega(n^2)$, and showed that the same bounds hold for triangulations of simple polygons. They also proved that if the polygon has $k$ reflex vertices, then the flip graph has diameter $O(n + k^2)$. This generalizes the well-known fact that the flip distance between any two triangulations of a convex polygon is at most $2n - 10$, for $n > 12$ [11].

Hanke, Ottmann, and Schuierer [4] showed that the flip distance between two triangulations of a point set is at most the number of crossings in the overlay of the source and the target triangulation. Eppstein [3] gave a polynomial-time algorithm for computing a lower bound on the flip distance. This bound is tight for point sets that do not contain empty 5-gons. For a survey on flip operations see Bose and Hurtado [2]. Recently, the problem of finding the flip distance between two triangulations of a point set was shown to be NP-complete by Lubiw and Pathak [7] and, independently, to be APX-hard by Pilz [8]. Here, we obtain the following result.

**Theorem 1 (main result)** *Let $P$ be a simple polygon. It is NP-complete to decide whether the flip distance between two triangulations of $P$ is a most $k$.*

In this extended abstract we omit all proofs; they can be found in the preprint [1]. Our reduction uses a variant of the Rectilinear Steiner Arborescence Problem. Let $S$ be a set of $N$ points in the plane, called *sinks*, whose coordinates are nonnegative integers. A rectilinear Steiner tree $T$ is called a *rectilinear Steiner arborescence* (RSA) for $S$ if (i) $T$ is rooted at the origin; (ii) each leaf of $T$ lies at a sink in $S$; and (iii) for each $s = (x, y) \in S$, the length of the path in $T$ from the origin to $s$ equals $x + y$, i.e., all edges in $T$ point north or east, as seen from the origin [9]. In the RSA problem, we are given a set of sinks $S$ and an integer $k$ and ask whether $S$ has an RSA of length at most $k$. The problem is strongly NP-complete; in particular, it remains NP-complete if $S$ is contained in an $n \times n$ grid, with $n$ polynomially bounded in $N = |S|$ [10]. For our reduction we need a restricted version of the RSA problem, called the YRSA problem. In an instance $(S, k)$ of the YRSA problem, we require that no two sinks in $S$ have the same $y$-coordinate. One can show that this variant of the problem remains NP-complete.

## 2 Double Chains

We use definitions (and illustrations) along the lines of [8]. A *double chain* $D$ consists of two chains, an *upper chain* and a *lower chain*. There are $n$ points

[*]Institute for Software Technology, Graz University of Technology, Austria. Partially supported by the ESF EUROCORES programme EuroGIGA - ComPoSe, Austrian Science Fund (FWF): I 648-N18. oaich@ist.tugraz.at.

[†]Institute of Computer Science, Freie Universität Berlin, Germany. Supported in part by DFG project MU/3501/1. mulzer@inf.fu-berlin.de.

[‡]Recipient of a DOC-fellowship of the Austrian Academy of Sciences at the Institute for Software Technology, Graz University of Technology. Part of this work has been done while this author was visiting the Dept. de Matemáticas, Universidad de Alcalá, Spain. apilz@ist.tugraz.at.

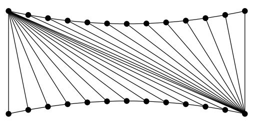


Figure 1: The upper and lower extreme triangulations of $P_D$ with a flip distance of $(n-1)^2$, as shown in [5].

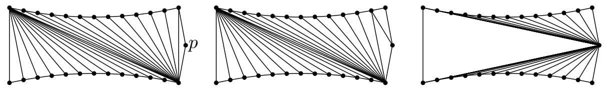


Figure 2: The extra point $p$ in the kernel of $D$ allows flipping one extreme triangulation of $P_D^p$ to the other in $4n-4$ flips.

on each chain, $\langle u_1, \ldots, u_n \rangle$ on the upper chain and $\langle l_1, \ldots, l_n \rangle$ on the lower chain, both numbered from left to right. The upper chain is reflex w.r.t. any point of the lower chain, and vice versa. Let $P_D$ be the polygon defined by $\langle l_1, \ldots, l_n, u_n, \ldots, u_1 \rangle$. We call the triangulation $T_u$ of $P_D$ where $u_1$ has maximum degree the *upper extreme triangulation*; observe that this triangulation is unique. The triangulation $T_l$ of $P_D$ where $l_1$ has maximum degree is called the *lower extreme triangulation*. The flip distance between $T_u$ and $T_l$ is $(n-1)^2$ [5], see Figure 1.

**Definition 1** *Let $D$ be a double chain. Let $W_1$ be the double wedge defined by the supporting lines of $u_1u_2$ and $l_1l_2$ whose interior does not contain a point of $D$. $W_n$ is defined analogously by the supporting lines of $u_nu_{n-1}$ and $l_nl_{n-1}$. Let $W = W_1 \cup W_n$ be called the* wedge *of $D$. A point is* outside *of $D$ if it is not contained in $W \cup P_D$. The* kernel *of $D$ is the intersection of the closed half-planes below $u_1u_2$ and $u_{n-1}u_n$, as well as above $l_1l_2$ and $l_{n-1}l_n$.*

We refer to a polygon as in Figure 2, where $p$ is in the kernel of $D$, by $P_D^p$. As mentioned in [12], the flip distance between the two extreme triangulations from Figure 1 is much smaller in $P_D^p$ than in $P_D$. Figure 2 shows that $4n-4$ flips suffice. It turns out that this is optimal, even for more general polygons:

**Lemma 2** *Let $P$ be a polygon that completely contains $P_D$ and has $\langle l_1, \ldots, l_n \rangle$ and $\langle u_n, \ldots, u_1 \rangle$ as part of its boundary. Further, let $T_1$ and $T_2$ be two triangulations that contain the upper extreme triangulation and the lower extreme triangulation of $P_D$ as a subtriangulation, respectively. Then $T_1$ and $T_2$ have flip distance at least $4n-4$.*

The proof by Lubiw and Pathak [7] for constant-size double chains directly generalizes to the above result. The following is a special case of a result from [8].

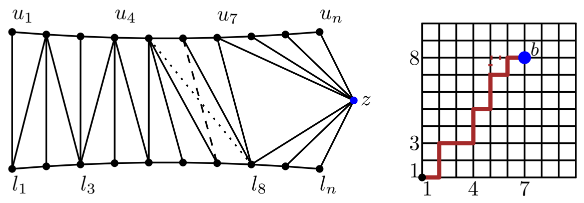


Figure 3: A double chain extended by a vertex $z$. The vertex $z$ is incident to $u_7$ and $l_8$, represented by the blue point $b$ in the grid. The brown chain path represents the chain triangles. If we flip edges to $z$, $b$ will move along that path. A flip between chain triangles (dotted edge replaced by the dashed one) changes a bend in that path (from the dotted one).

**Lemma 3** *Let $P$ be a polygon that completely contains $P_D$ and has $\langle l_1, \ldots, l_n \rangle$ and $\langle u_n, \ldots, u_1 \rangle$ as part of its boundary, and let $T_1$ and $T_2$ be two triangulations that contain the upper extreme triangulation and the lower extreme triangulation of $P_D$ as subtriangulation, respectively. Suppose there is no vertex in the interior of the wedge of $P_D$. Then the flip distance between $T_1$ and $T_2$ is at least $(n-1)^2$.*

Take a polygon $P_D^z$ and consider a triangulation $T$ of $P_D^z$. A *chain edge* is an edge of $T$ between the upper and the lower chain of $D$. A *chain triangle* is a triangle that contains two chain edges. We use the chain edges to define the *chain path*, an abstract path on the $n \times n$ grid. Let $e_1, e_2, \ldots, e_m$ be the chain edges, sorted from left to right according to their intersections with a line $\ell$ that separates the upper from the lower chain. For $i = 1, \ldots, m$, write $e_i = (u_v, l_w)$ and set $c_i = (v, w)$. Note that, in particular, $c_1 = (1, 1)$, which we use as the *root* of our setting. Since $T$ is a triangulation, any two consecutive edges $e_i$, $e_{i+1}$ share one endpoint, while the other endpoints are adjacent on the corresponding chain. Thus, $c_{i+1}$ dominates $c_i$ and $\|c_{i+1} - c_i\|_1 = 1$. The chain path is defined as the path $c_1c_2 \ldots c_m$. See Figure 3 for an example.

The chain path is an $x$- and $y$-monotone path in the $n \times n$ grid. We call its upper right endpoint $b$. By observing the changes of the chain path by flips of different types, the following lemma can be obtained.

**Lemma 4** *Let $T$ be a triangulation of $P_D^z$. Then $T$ uniquely determines an $x$- and $y$-monotone path (i.e., the chain path) in the $n \times n$ grid starting at the root $(1, 1)$. Conversely, any chain path uniquely determines a triangulation of $T$. The possible flips of $T$ correspond to the following operations on the chain path: (i) extend the right endpoint north or east; (ii) shorten the path at the right endpoint; (iii) change an east-north bend to an north-east bend, or vice versa.*

## 3 Installing Sinks

We show how to reduce YRSA to our flip distance problem. Let $S$ be a set of $N$ sinks with root at $(1,1)$ on an $(n-1)\times(n-1)$ grid (recall that $n$ is polynomial in $N$). We describe how to construct a polygon $P_D^*$ for $S$. Our construction has two integral parameters $\beta$ and $d$. With foresight, we set $\beta = 2N$ and $d = nN$.

Let $P_D^z$ be the polygon from Section 2, but with $\beta n$ vertices on each chain. As we saw in Section 2, we can interpret a triangulation of $P_D^z$ as a a chain path in the $\beta n \times \beta n$ grid. We imagine that the sinks of $S$ are in this grid, with their coordinates multiplied by $\beta$. For each sink $s = (x, y)$, we place a (rotated) small double chain $D_s$ of size $d$ such that $l_{\beta y}$ and $l_{\beta y+1}$ correspond to the last point on the lower and upper chain of $D_s$, respectively. In addition, $u_{\beta x}$ is the only point in the kernel of $D_s$ and $u_{\beta x}$ is also the only point in the interior of the wedge of $D_s$. We call the resulting polygon $P_D^*$. If $\beta$ is large enough, the small double chains $D_s$ do not interfere with each other, and $P_D^*$ is simple. Since the $y$-coordinates in $S$ are pairwise distinct, we create at most one double chain at each edge of the lower chain of $P_D^z$. Observe that we have some flexibility for the precise placement of the points of each $D_s$. Thus we can choose their placement in a way that their coordinates are polynomial in $n$.

Next, we describe the source and target triangulation for $P_D^*$. The source triangulation $T_1$ contains all edges of $P_D^z$. The interior of $P_D^z$ is triangulated such that all edges are incident to $z$, i.e., $b$ is at the root. The small double chains are all triangulated with the upper extreme triangulation. The target triangulation $T_2$ is defined similarly, but now all the small double chains are triangulated with the lower extreme triangulation (note that the choice of the upper and lower chain is arbitrary for the small double chains).

Hence, each corresponding pair of small double chains in $T_1$ and $T_2$ has flip distance $(d-1)^2$ due to Lemma 3, unless the appropriate vertex on the upper chain of $P_D^*$ is used. Intuitively, if $d$ is large enough, a shortest flip sequence will have to "traverse" each sink, inducing an arborescence for $S$. Vice versa, every arborescence for $S$ gives a short flip sequence between $T_1$ and $T_2$.

**Lemma 5** *Let $A$ be an arborescence for $S$ of length $k$. Then the flip distance on $P_D^*$ between $T_1$ and $T_2$ is at most $2\beta k + (4d-2)N$.*

Next we consider the opposite direction of the correspondence. In the proof of the following lemma, we will describe a mapping from each triangulation $T$ of $P_D^*$ to a triangulation $T_z$ of $P_D^z$. For each sink $s \in S$, the corresponding chain triangle $t_s$ in $T_z$ is defined as the chain triangle in $P_D^z$ that allows the double chain $D_s$ to be flipped quickly. We say that a flip sequence $\sigma_1$ on $P_D^z$ *visits* a sink $s \in S$, if $\sigma_1$ has at least one triangulation $T$ that contains the corresponding chain triangle $t_s$. We call $\sigma_1$ a *flip traversal* for $S$ if (i) the sequence $\sigma_1$ begins and ends in the same triangulation $T_z$ such that $T_z$ corresponds to $b$ lying on the root; (ii) the sequence $\sigma_1$ visits every sink in $S$.

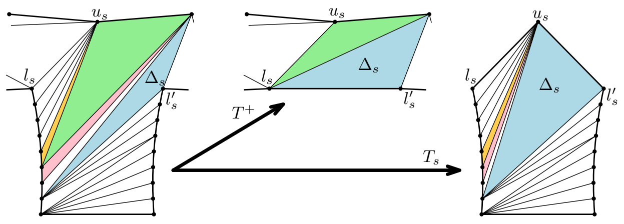


Figure 4: A part of a triangulation of $P_D^*$ and the two corresponding triangulations $T_z$ and $T_s$.

**Lemma 6** *Let $\sigma$ be a flip sequence on $P_D^*$ from $T_1$ to $T_2$ with $|\sigma| < (d-1)^2$. Then there exists a flip sequence $\sigma_1$ on $P_D^z$ such that $\sigma_1$ is a flip traversal for $S$ with $|\sigma_1| \le |\sigma| - (4d-4)N$.*

**Sketch of Proof.** Let $T^*$ be a triangulation of $P_D^*$. Let $D_s$ be a small double chain placed between the vertices $l_s$ and $l'_s$ with $u_s$ being the vertex in the kernel of $D_s$. We define $\Delta_s$ as the triangle that is either the inner triangle (i.e., all three sides are diagonals) incident to two vertices of $D_s$ or the triangle that is incident to both convex vertices of $D_s$ but is not an ear. Note that in the first case the third vertex might be $u_s$ and that in the latter case the third vertex has to be $u_s$. Due to the structure of $P_D^*$ there always exists exactly one such triangle $\Delta_s$ per sink. Let the polygon $P_{D_s}^{u_s}$ consist of the double chain $D_s$ extended by the vertex $u_s$, and let $T_s$ denote a triangulation of it. We define a mapping of any triangulation $T^*$ of $P_D^*$ to a triangulation $T_z$ of $P_D^z$ and to triangulations $T_s$ for all sinks $s$. The triangulation $T_z$ contains every triangle that has all three vertices in $P_D^z$. For each triangle $\nabla$ that has two vertices on $P_D^z$ and one on the left chain of $D_s$, we replace the apex on $D_s$ by $l_s$. The analogous is done if the apex of a triangle $\nabla$ is on the right chain of $D_s$; we replace that apex by $l'_s$. For every sink $s$, the triangle $\Delta_s$ is known to have an apex at a point $u_i$ of the upper chain. In $T_z$, we replace $\Delta_s$ by the triangle $l_s l'_s u_i$. Since these are exactly the triangles needed for a triangulation of $P_D^z$ and no two triangles overlap, $T_z$ is indeed a triangulation of $P_D^z$. Similarly, all triangles in $T^*$ with all three vertices on $P_{D_s}^{u_s}$ are also in $T_s$, and the triangles having two points on $D_s$ and whose apex is not in $P_{D_s}^{u_s}$ get their apex at $u_s$ in $T_s$ (note that this includes $\Delta_s$). See Figure 4.

Using a case analysis, one can show that each flip changes at most one of the triangulations that the original triangulation is mapped to. □

## 4 Arborescences and Traces

Next, we define traces (domains drawn on the grid) and relate them to flip traversals. A *trace* is drawn on the $\beta n \times \beta n$ grid. It consists of *edges* and *boxes*: an edge is a line segment of length 1 whose endpoints have positive integer coordinates; a box is a square of side length 1 whose corners have positive integer coordinates. Similar to arborescences, we require that a trace $R$ (i) is (topologically) connected; (ii) contains the root $(1, 1)$; and (iii) from every grid point contained in $R$ there exists an $x$- and $y$-monotone path to the root that lies completely in $R$. We say $R$ is a *covering trace* for $S$ (or, *$R$ covers $S$*) if every sink in $S$ is part of $R$.

Let $\sigma_1$ be a flip traversal as in Lemma 6. By Lemma 4, we can interpret the sequence $\sigma_1$ as the evolution of a chain path. This gives a covering trace $R$ for $S$ in the following way. For every flip in $\sigma_1$ that extends the chain path, we add the corresponding edge to $R$. For every chain flip in $\sigma_1$, we add the corresponding box to $R$. Afterwards, we remove from $R$ all edges that coincide with a side of a box in $R$. Clearly, $R$ is (topologically) connected. Since $\sigma_1$ is a flip traversal for $S$, every sink is covered by $R$ (i.e., incident to a box or edge in $R$). Note that every grid point $p$ in $R$ is connected to the root by an $x$- and $y$-monotone path on $R$, since at some point $p$ belonged to a chain path in $\sigma_1$. Hence, $R$ is indeed a trace, the unique *trace of* $\sigma_1$.

Next, we define the *cost* of a trace $R$, cost$(R)$, so that if $R$ is the trace of a flip traversal $\sigma_1$, then cost$(R)$ gives a lower bound on $|\sigma_1|$. An edge has cost 2. Let $B$ be a box in $R$. A *boundary side* of $B$ is a side that is not part of another box. The cost of $B$ is 1 plus the number of boundary sides of $B$. Then, cost$(R)$ is the total cost over all boxes and edges in $R$.

**Proposition 7** *Let $\sigma_1$ be a flip traversal and $R$ a trace for $\sigma_1$. Then cost$(R) \leq |\sigma_1|$.*

**Observation 1** *Any shortest path tree $A_{\sigma_1}$ in $R$ for the root w.r.t. $S$ is an arborescence.*

If $\sigma_1$ contains no chain flips, the corresponding trace $R$ has no boxes, but it may not be acyclic. However, due to Observation 1 it contains an arborescence $A_{\sigma_1}$, in particular with $2|A_{\sigma_1}| \leq \text{cost}(R)$.

**Lemma 8** *Let $\sigma_1$ be a flip traversal of $S$. Then there exists a covering trace $R$ for $S$ in the $\beta n \times \beta n$ grid such that $R$ does not contain a box and such that cost$(R) \leq |\sigma_1|$.*

**Corollary 9** *Let $\sigma$ be a flip sequence on $P_D^*$ from $T_1$ to $T_2$ with $|\sigma| \leq 2\beta k + (4d-2)N$. Then there exists a rectilinear Steiner arborescence for $S$ of length at most $k$.*

**Sketch of Proof.** Since there is always an arborescence on $S$ of length less than $2nN$, we may assume that $k < 2nN$. We can use Lemma 6, and then apply Lemma 8 to the resulting sequence to obtain an arborescence $A$ of length at most $\beta k + N$. It is well-known that there exists a minimal arborescence $A'$ for $S$ whose length is a multiple of $\beta$. Thus, since $\beta > N$, we get that $A'$ has length at most $\beta k$, so the corresponding arborescence for $S$ on the original grid has length at most $k$. □

Together with Lemma 5, this implies Theorem 1.

## References

[1] O. Aichholzer, W. Mulzer, and A. Pilz. Flip Distance Between Triangulations of a Simple Polygon is NP-Complete. *ArXiv e-prints*, Sept. 2012. arXiv:1209.0579.

[2] P. Bose and F. Hurtado. Flips in planar graphs. *Comput. Geom.*, 42(1):60–80, 2009.

[3] D. Eppstein. Happy endings for flip graphs. *JoCG*, 1(1):3–28, 2010.

[4] S. Hanke, T. Ottmann, and S. Schuierer. The edge-flipping distance of triangulations. *J.UCS*, 2(8):570–579, 1996.

[5] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete Comput. Geom.*, 22:333–346, 1999.

[6] C. L. Lawson. Transforming triangulations. *Discrete Math.*, 3(4):365–372, 1972.

[7] A. Lubiw and V. Pathak. Flip distance between two triangulations of a point-set is NP-complete. In *Proc. 24th CCCG*, pages 127–132, Charlottetown, Canada, August 2012.

[8] A. Pilz. Flip distance between triangulations of a planar point set is APX-hard. Submitted, preprint available at arXiv:1206.3179, 2012.

[9] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7:277–288, 1992.

[10] W. Shi and C. Su. The rectilinear Steiner arborescence problem is NP-complete. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 780–787, 2000.

[11] D. Sleator, R. Tarjan, and W. Thurston. Rotation distance, triangulations and hyperbolic geometry. *J. Amer. Math. Soc.*, 1:647–682, 1988.

[12] J. Urrutia. Algunos problemas abiertos. In N. Coll and J. Sellares, editors, *Proc. IX ECG*, pages 13–24. Univ. De Girona, July 2001.

# Selecting the Aspect Ratio of a Scatter Plot Based on Its Delaunay Triangulation

Martin Fink* Jan-Henrik Haunert* Joachim Spoerhase* Alexander Wolff*

## Abstract

Scatter plots are diagrams that visualize sets of points in two dimensions. They allow users to detect correlations and clusters in the data. Whether a user can accomplish these tasks highly depends on the *aspect ratio* selected for the plot, i.e., the ratio between the horizontal and the vertical extent of the diagram. We argue that an aspect ratio is good if the Delaunay triangulation of the scatter plot has some nice geometric property, e.g., a large minimum angle or a small total edge length. In order to find an optimum aspect ratio according to a given criterion we present an algorithm that efficiently maintains the Delaunay triangulation of the point set when traversing all aspect ratios.

## 1 Introduction

Scatter plots are diagrams that visualize sets of points in the plane to allow humans to find patterns, clusters, and trends in the data. When drawing a scatter plot, one is usually free to choose its aspect ratio, that is, the ratio between its horizontal and its vertical extent. With a bad choice, however, humans may fail in recognizing a pattern in the data. While the automatic selection of a good aspect ratio for a line chart has been intensively discussed [1–3,8,9], methods that select the aspect ratio for a scatter plot are missing.

Most methods for aspect-ratio selection rely on properties of the line segments displayed in the diagram. To apply a line-segment-based method to scatter plots, Cleveland et al. [2] suggested to consider "virtual line segments", which humans may perceive though they do not physically exist. The virtual line segments may be the segments of a (virtual) polyline connecting all data points, the segments of a regression (poly-)line, or, as suggested by Talbot et al. [8] in order to deal with pairs of variables without a functional relationship, the segments of contour lines yielded by a kernel density estimator (KDE).

Our main concern with their method is that the KDE result (and the final aspect ratio) heavily depends on the aspect ratio of the input data: it makes a difference whether the input data is given, for example, in degree Fahrenheit or in degree Celsius. To overcome this deficiency, we define—based on the output diagram visible to the user (and thus on the aspect ratio chosen)—whether two points are linked via a virtual edge. We argue that the aspect ratio is good if the virtual edges have nice properties. We define a virtual line segment for each edge of the Delaunay triangulation $D(P)$ of the set $P$ of points displayed. This generally defines a meaningful (usually termed *the natural*) neighborhood for $P$: if for two points $u, v \in P$ there exists a point $w \in \mathbb{R}^2$ such that both $u$ and $v$ are nearest neighbors of $w$ in $P$, then there exists an edge $\{u, v\}$ in $D(P)$.

We now need to choose the aspect ratio of a scatter plot such that the displayed points have a nice Delaunay triangulation, for example, one of minimum total edge length. To solve this problem, we present an algorithm that traverses the space of aspect ratios while efficiently maintaining the Delaunay triangulation. This is closely related to the problem of maintaining a Delaunay triangulation for a set of continuously moving points. In fact, our setting corresponds to the special case where each point moves along a horizontal line at an individual but constant speed.

Roos [5] described a data structure for maintaining a dynamic Delaunay triangulation which takes $O(\log n)$ time per topological change of the triangulation. His result requires that the movement of the points meets a (weak) technical assumption that holds for many natural scenarios such as movement along parametric polynomial curves. The question of *how many* topological changes a dynamic Delaunay triangulation can undergo (under some weak and natural assumptions on the movement) is an important field of research. Recently, Rubin [6] showed that there are at most $O(n^{2+\epsilon})$ topological changes for a large class of movements (including our scenario). We argue that in our case there are in fact only $O(n^2)$ topological events. Since updating the Delaunay triangulation requires $O(\log n)$ time, we can traverse all possible topological Delaunay triangulations in $O(n^2 \log n)$ time.

## 2 Problem Statement

Given a set $Q = \{q_1, q_2, \ldots, q_n\} \subset \mathbb{R}^2$ of points, we search for a scale factor $s \in \mathbb{R}^+$ defining the set $P$ of displayed points, i.e., the resulting scatter plot. We denote the coordinates of each point $q_i \in Q$ by $x_i$ and $y_i$ and require the scatter plot $P$ to contain the point

*Lehrstuhl für Informatik I, Universität Würzburg, Germany, URL: www1.informatik.uni-wuerzburg.de/en/staff

$(s \cdot x_i, y_i/s)$. This ensures that the bounding box of $P$ and the bounding box of $Q$ have the same area.

In order to choose a good scale factor, we define various criteria. Our main approach is to measure the quality of a scatter plot with a function $f\colon \mathcal{S} \to \mathbb{R}$, where $\mathcal{S}$ is the set of all possible scatter plots for the given point set. Then, we search for a scale factor whose corresponding scatter plot maximizes $f$.

All our measures are based on the Delaunay triangulation of the point set. A characterization of the Delaunay triangulation is that it maximizes the *smallest angle* among all triangulations. The natural idea is to use this measure also over different scale factors, i.e., to define $f(P)$ as the value of the smallest angle of the Delaunay triangulation of $P$. We will see that we can find the scale factor optimizing this criterion. Nevertheless, there are also other criteria that make sense, and which we can, at least, approximate:

- maximize the *mean inradius* of triangles of $D(P)$,
- maximize the total *compactness* of triangles of $D(P)$; the compactness of triangle $\Delta$ with perimeter $c(\Delta)$ and area $A(\Delta)$ is $\sqrt{A(\Delta)}/c(\Delta)$ [4],
- minimize the total *uncompactness* of triangles of $D(P)$; the uncompactness is $c(\Delta)/\sqrt{A(\Delta)}$ [4],
- minimize the *total length* of all edges of $D(P)$.

## 3 Algorithm

Finding an optimum scale factor $s$ can be seen as a continuous process. We continuously increase $s$ starting with $s = 1$. In doing so, the Delaunay triangulation undergoes *topological changes* at certain *event points* which we keep track of. We output the scale factor at which our objective function is optimized. Symmetrically, we traverse all scale factors $s < 1$.

Our algorithm consists of two layers. The first layer steps through the discrete set of event points in the order in which they occur in the above described process. The second layer optimizes between consecutive event points $s_i$ and $s_{i+1}$, where the topological structure of the Delaunay triangulation does not change. Each of our optimization measures is a continuous function of $s \in [s_i, s_{i+1}]$. We can then compute the scale factor (or an approximation of it) at which this function is maximized within $[s_i, s_{i+1}]$. Doing this for all such intervals allows us to determine the globally optimal scale factor (or an approximation).

Recall that a triangulation $D(P)$ of a point set $P = \{p_1, \ldots, p_n\}$ is Delaunay if the circumcircle of each triangle of $D(P)$ is empty, that is, it does not contain points of $P$ in its interior (c1). Alternatively, for any edge $p_ip_j$ of $D(P)$, there must exist an empty circle whose boundary contains $p_i$ and $p_j$ (c2).

**Maintaining the Delaunay triangulation through scale space** Since the number of points on the convex hull is constant over the whole scale space, the same holds for the number of triangles and edges. Hence, for each triangle (or edge) that disappears from the Delaunay triangulation at some event point $s_h$, a new triangle (or edge) is created and vice versa. For the sake of simplicity, we assume in what follows that no five points are co-circular at any scale factor.

Consider an event point $s_h$ at which some triangle disappears from $D(P)$. According to criterion (c1) there is at least one point $p_l$ that *enters* the circumcircle $C(p_i, p_j, p_k)$ of the triangle at $s_h$. More precisely, the interior of $C(p_i, p_j, p_k)$ contains $p_l$ at any $s > s_h$ but not at $s = s_h$ where $p_l$ is on the boundary.

The following lemma (whose proof we skip) characterizes the situations where topological changes occur.

**Lemma 1** *Assume that the Delaunay triangulation undergoes a topological change at event point $s_h$. Then there is a quadrilateral $p_i, p_j, p_k, p_l$ in the Delaunay triangulation with diagonal $p_ip_k$ such that $p_l$ enters the circle $C(p_i, p_j, p_k)$ at $s_h$.*

Consider a point $p_l$ entering the circumcircle of a triangle $\Delta p_ip_jp_k$ at event point $s_h$ between $p_i$ and $p_k$ as described in Lemma 1. If we replace edge $p_ip_k$ with edge $p_jp_l$ at event point $s_h$, we obtain a new triangulation. We call this operation a *flip*. The crucial observation is that if no further flips are to be performed at $s_h$, the current Delaunay triangulation is valid at $s_h + \epsilon$ for sufficiently small $\epsilon > 0$. Also note that the flip of $p_ip_k$ corresponds to the co-circularity of the unique quadrilateral $p_ip_jp_kp_l$ that contains $p_ip_k$.

Our algorithm determines the sequence $s_1, \ldots, s_m$ of event points one by one in increasing order starting with $s_1 := 1$. Given an event point $s_i$ and the corresponding Delaunay triangulation $D(P(s_i))$, we face the problem of computing the next event point $s_{i+1}$, that is, the smallest scale factor larger than $s_i$ at which we have to perform flips. Note that any flip that we have to perform at $s_{i+1}$ corresponds to the co-circularity of the unique quadrilateral of $D(P(s_{i+1}))$ containing the edge flipped. In other words, for every edge, we have to compute the smallest scale factor larger than $s_i$ at which the corresponding quadrilateral becomes co-circular (if any). For each edge, this event point can be computed in constant time by solving a system of four linear equations [5].

Our algorithm traverses the sequence of event points $s_1, \ldots, s_m$ in increasing order as follows. Initially, we compute the Delaunay triangulation at $s_1 = 1$ and set up a priority queue $Q$ that maintains, for each edge of the current triangulation, the event point at which this edge has to be flipped. We then iteratively compute the sequence of event points. First, we get the next event point $t$ by extracting from queue $Q$ the next scale factor $t$ at which some edge $e$ has to be flipped. When $e$ is flipped, the queue $Q$ has to be updated accordingly. We must update the event points at which the four edges of the quadrilateral containing $e$ have to be flipped.

Let's analyze the running time of the algorithm. The initialization step takes $O(n \log n)$ time for the Delaunay triangulation and $O(n)$ time for building the priority queue. For each flip, we spend $O(\log n)$ time for extracting the minimum of $Q$ and updating the four edges on the corresponding quadrilateral.

It remains to determine the maximum number of flips performed by the algorithm. Consider the situation of Lemma 1 where we flip the edge $p_ip_k$ at event point $s_h$. For every scale factor $s > s_h$, the circle $C(p_i, p_j, p_k)$ contains $p_l$ in its interior. By criterion (c2) we can conclude that the edge $p_ip_k$ can not be part of a Delaunay triangulation for any $s > s_h$. Since there are at most $O(n^2)$ potential edges, and every edge that is flipped cannot be re-inserted into the Delaunay triangulation, there are only $O(n^2)$ flips.

**Theorem 2** *We traverse the sequence $s_1, \ldots, s_m$ of event points in increasing order and compute for $1 \leq i \leq m-1$ the Delaunay triangulation valid in the interval $[s_i, s_{i+1}]$ in a total running time of $O(n^2 \log n)$.*

The event points can be scattered quite unevenly over the scale space. It is therefore not sufficient to consider only the event points as potential solutions.

**Finding an approximate solution** We describe our method for minimizing the uncompactness $c_D$ of the Delaunay triangulation $D$. It can be applied to the other objective functions in a similar manner.

Fix an interval $[s_i, s_{i+1}]$ of consecutive event points and fix an arbitrarily small error parameter $\epsilon > 0$. Our goal is to find an $(1+\epsilon)$-approximate solution for $c_D$, that is, a scale factor $s_a$ for which $c_D(s_a) \leq (1+\epsilon)c_D(s^*)$ where $s^*$ is the globally optimal scale factor. Given an edge $e$ of $D$, its length $l_e(s)$ depends on the scale factor. It is easy to see that $l_e((1+\epsilon)s) \leq (1+\epsilon)l_e(s)$. As the area is constant, for the uncompactness $c_\Delta$ of a triangle $\Delta$ it also holds that $c_\Delta((1+\epsilon)s) \leq (1+\epsilon)c_\Delta(s)$, and similarly the total uncompactness $c_D(s) = \sum_{\Delta \in D} c_\Delta(s)$ is bounded.

We restrict ourselves to scale factors between 1 and $C$ for some large constant $C$, which is sufficient for practical purposes. Let $s_1, \ldots, s_m$ denote the sequence of event points (between 1 and $C$) and let $s_{m+1} := C$. Now consider a fixed interval $[s_i, s_{i+1}]$ with $i = 1, \ldots, m$. Our algorithm computes $c_D(\cdot)$ for all *test values* $t_j := s_{i+1}/(1+\epsilon)^j$ where $j \in \mathbb{N}$ and $t_j \in [s_i, s_{i+1}]$. Let $t_{j^*}$ be the test value at which $c_D(\cdot)$ is minimized and let $s^*$ be an optimum scale factor in the interval $[s_i, s_{i+1}]$. Using the above bound, it can be shown that $c_D(t_{j^*}) \leq (1+\epsilon)c_D(s^*)$, that is, we obtain a $(1+\epsilon)$-approximation for the current interval; hence, taking the optimum over all intervals, we can find a $(1+\epsilon)$-approximation. Let's summarize.

**Theorem 3** *For any fixed $\epsilon > 0$, we can compute a $(1+\epsilon)$-approximate solution for minimizing the uncompactness measure in $O(n^3)$ time. Taking $\epsilon$ into account the running time is $O(n(n^2 + 1/\log(1+\epsilon)))$.*

**Maximizing the minimum angle** We sketch an efficient exact algorithm for determining a globally optimal scale factor for the objective of maximizing the smallest angle of a Delaunay triangulation.

Each angle $\beta$ (formed by two edges) that occurs during traversing the scale space can be described as a function of the scale $s$. Its domain is an interval $[l_\beta, r_\beta]$, the intersection of the life times of the edges. Let $\mathcal{A}$ be the set of all angles (functions) that appear at some scale factor in the Delaunay triangulation, and let $\text{env}(\mathcal{A})$ be the lower envelope of $\mathcal{A}$. Determining the scale factor that maximizes the smallest angle of the Delaunay triangulation amounts to determining the maximum of $\text{env}(\mathcal{A})$.

Consider some angle $\beta \in \mathcal{A}$, and let $p_ip_j$ and $p_jp_k$ be the edges defining $\beta$. Now consider a coordinate system whose origin is located at $p_j$. Then $\beta > \pi/2$ for any $s$ if $p_i$ and $p_k$ lie in diagonally opposite quadrants. As we are only interested in the *lower* envelope, we can safely remove all such angles from $\mathcal{A}$.

Under this assumption, it is not hard to verify that any $\beta \in \mathcal{A}$ can be expressed as

$$\beta(s) = c_1\pi + \arctan\left(c_2 s/(c_3 s^2 + 1)\right)$$

where $c_1 \in \{0, 1\}$ and $c_2, c_3 \in \mathbb{R}$ are easily computable constants that only depend on the edges defining $\beta$ but not on $s$. Elementary calculations reveal that two functions of the above form can have at most one intersection. Let $m = O(n^2)$ be the number of angles in $\mathcal{A}$. Because any two functions in $\mathcal{A}$ intersect at most once, it is known from the theory of Davenport–Schinzel sequences that the lower envelope $\text{env}(\mathcal{A})$ has complexity (number of distinct curve segments) at most $\lambda_3(m) = O(m\alpha(m))$ [7] where $\alpha$ denotes the functional inverse of the Ackermann function.

Agarwal and Sharir [7] show that the lower envelope of $m$ partially defined functions can be computed in $O(\lambda_{r+1}(m) \log m)$ time if any two functions intersect at most $r$ times. Hence, their algorithm runs in $O(n^2 \log n)$ time in our case. For each curve segment, the maximum can be computed in constant time. As the curve complexity is $O(n^2\alpha(n^2))$, we conclude.

**Theorem 4** *The globally optimal scale factor for the objective of maximizing the smallest angle can be computed in $O(n^2 \log n)$ time.*

## 4 Experimental Results

We implemented our algorithms in Java. For maximizing the minimum angle, we used a simplified version of our exact algorithm that is slower but easier

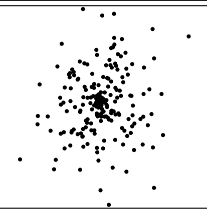
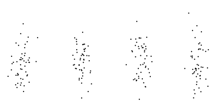
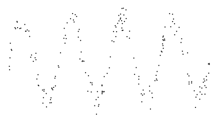
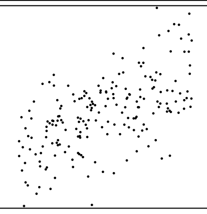
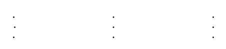
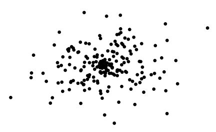
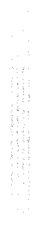
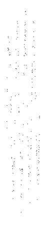
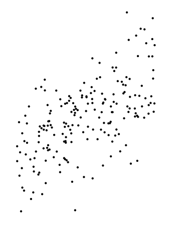
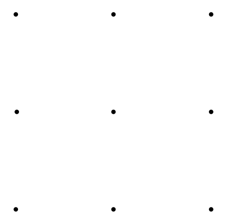
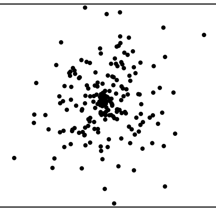
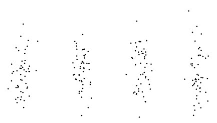
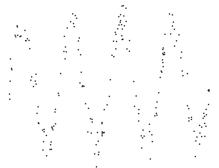
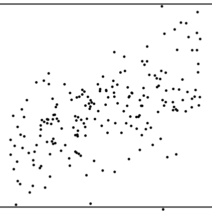
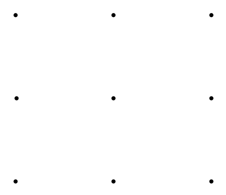
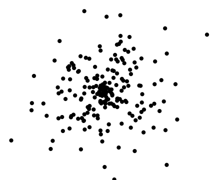
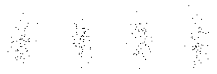
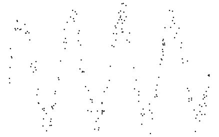
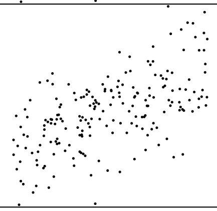
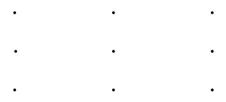
| | normal distrib. | four clusters | noisy sine | rough trend | defect grid |
|---|---|---|---|---|---|
| max. min. angle |  |  |  |  |  |
| max. mean inradius |  |  |  |  |  |
| min. total length |  |  |  |  |  |
| min. uncompact. |  |  |  |  |  |

Table 1: Test results for 4 optimization criteria on 5 generated instances (outputs scaled to fit into the boxes).

to implement. For the other criteria, we used the approximation algorithm with $\epsilon = 0.01$.

Table 1 shows results on five test instances: a cluster of points normally distributed around a center; four such clusters next to each other; points sampled along a sine function with normally distributed distance to it; the same for a linear trend; nine points lying on a grid with the exception of one of them that is moved a bit away from the grid point. We omitted the results for maximizing the compactness as they were quite similar to the ones for minimizing the total length (yet more stretched in $y$-direction for the sine and rough linear trend).

As visible for the four clusters and the noisy sine, the total length and especially the inradius criterion often tend to stretch the plot too much. The angle criterion, and, to a lesser degree, the uncompactness criterion are sensitive to small changes, see the defect grid. Over all tests, however, the uncompactness minimization showed the best results.

## 5 Conclusion and Future Work

Our tests confirm that selecting the aspect ratio of a scatter plot based on the Delaunay triangulation is a promising approach.

## References

[1] W. S. Cleveland. A model for studying display methods of statistical graphics. *J. Comput. Graph. Statist.*, 2(4):323–343, 1993.

[2] W. S. Cleveland, M. E. McGill, and R. McGill. The shape parameter of a two-variable graph. *J. Am. Stat. Assoc.*, 83(289–300), 1988.

[3] J. Heer and M. Agrawala. Multi-scale banking to 45°. *IEEE T. Vis. Comput. Gr.*, pages 701–708, 2006.

[4] A. M. MacEachren. Compactness of geographic shape: Comparison and evaluation of measures. *Geografiska Annaler. Ser. B, Human Geogr.*, 67(1):53–67, 1985.

[5] T. Roos. Voronoi diagrams over dynamic scenes. *Discrete Appl. Math.*, 43(3):243–259, 1993.

[6] N. Rubin. On topological changes in the Delaunay triangulation of moving points. In *Proc. 28th ACM Symp. Comput. Geom. (SoCG'12)*, pages 1–10, 2012.

[7] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, 1995.

[8] J. Talbot, J. Gerth, and P. Hanrahan. Arc length-based aspect ratio selection. *IEEE T. Vis. Comput. Gr.*, 17(12):2276–2282, 2011.

[9] J. Talbot, J. Gerth, and P. Hanrahan. An empirical model of slope ratio comparisons. *IEEE T. Vis. Comput. Gr.*, 18(12):2613–2620, 2012.

# Computational Aspects of Triangulations with Bounded Dilation

Wolfgang Mulzer[*] Paul Seiferth[*]

## Abstract

Let $T$ be a triangulation on a planar point set $S$. If $T$ has bounded dilation, then the shortest path distance between any two vertices approximates their Euclidean distance. We examine if such triangulations can be used to design efficient algorithms for various geometric problems.

First, we show that given a triangulation with bounded dilation, one can find the closest pair of points in $S$ in linear time on a pointer machine.

Afterwards, we consider an algorithm by Krznaric and Levcopoulos to compute a hierarchical clustering for $S$ in linear time, once the EMST of $S$ is known. We study how their result can be generalized to MSTs of triangulations with bounded dilation. It turns out that their algorithm remains (almost) correct for any such MST. In general, however, the resulting running time might be superlinear. We identify a sufficient condition for a linear time bound and construct a triangulation without this condition as counterexample.

It remains open to identify interesting classes of bounded-dilation triangulations with this property.

## 1 Introduction

Delaunay triangulations (DT) constitute perhaps the most famous and most well-studied proximity structure. Given a planar point set $S$, the DT of $S$ encodes many aspects of the distances between the points in $S$, and it enables us to compute in linear time many other structures on $S$, such as the Euclidean minimum spanning tree (EMST) and thus the closest pair of points in $S$, the Gabriel graph, the nearest-neighbor graph, a quadtree, or a well-separated pair decomposition (see, e.g., [5] and the references therein). But what exactly is it that makes DTs so powerful? How much structure is needed in order to represent the proximity information in $S$?

A very general family of triangulations that includes the DT is given by triangulations of *bounded dilation*. In these triangulations, the shortest path distance between any two vertices approximates their Euclidean distance by a constant factor. Examples of other triangulations with bounded dilation are given by the minimum weight and the greedy triangulation [2]. We would like to explore how strong this information is, compared to the DT, and if it can be exploited algorithmically.

As an introductory example we show that the closest pair of points in $S$ can be found in linear time, once a triangulation with bounded dilation is known.

Afterwards, we consider an algorithm by Krznaric and Levcopoulos (KL) for computing a hierarchical clustering for a planar point set $S$ in linear time, given the Euclidean minimum spanning tree $\mathrm{EMST}(S)$ [3]. In particular, KL use the $c$-*clustering*: a subset $U \subseteq S$ is called a $c$-*cluster* for some constant $c \geq 1$ if the distance $d(U, S \setminus U)$ is greater than $c \cdot \mathrm{rdiam}(U)$, where $\mathrm{rdiam}(U)$ is the diameter of the axis-parallel bounding rectangle for $U$. The set of all $c$-clusters for $S$ constitutes a laminar family, i.e., two distinct $c$-clusters are either disjoint or one is a proper subset of the other. Thus, the set of all $c$-clusters can be naturally represented as a $c$-*cluster tree* whose nodes correspond to the $c$-clusters and whose leaves correspond to the points in $S$. A relaxed version of these trees that is more flexible, but retains the essential properties, are $(c_1, c_2)$-*cluster trees*, introduced by Mulzer and Löffler [5]: let $1 \leq c_1 \leq c_2$ be constants. We require that every $c_2$-cluster is represented in the tree, but allow other clusters to be inserted in the hierarchy, as long as they are at least $c_1$-clusters.

KL presented an algorithm to compute a $c$-cluster tree for $S$ from $\mathrm{EMST}(S)$ in linear time and showed that $c$-cluster trees can be used to speed up the computation of various structures for $S$, e.g. a quadtree for $S$ and the (approximated) single and complete linkage clustering [3,4]. The correctness proof is based on a characterization of $c$-clusters in terms of the EMST of $S$. We show that a similar characterization holds for MSTs of triangulations with bounded dilation. This enables us to argue that the KL-algorithm is also correct for such triangulations. To achieve linear running time, KL need a property of the EMST, which unfortunately does not hold for MSTs of general bounded-dilation triangulations. We construct a counterexample to illustrate this issue.

## 2 Preliminaries and Notation

Let $G = (V, E)$ be a graph and $U \subseteq V$. The *induced subgraph* $G[U]$ on $U$ is the graph on vertex set $U$ that contains exactly those edges from $G$ with both endpoints in $U$. Furthermore, we define

[*]Institut für Informatik, Freie Universität Berlin, Germany, {mulzer, pseiferth}@inf.fu-berlin.de. Supported in part by DFG project MU/3501/1.

$I_G(U) := \{uv \in E(G) \mid u \in U \text{ and } v \notin U\}$ as all edges with exactly one endpoint in $U$.

Let $T$ be a planar triangulation. The *dilation* of two vertices $u, v$ of $T$ is the ratio of the shortest path distance $d_T(u, v)$ between $u$ and $v$ in $T$ and their Euclidean distance. When considering the maximum of these ratios we obtain the *dilation* $\delta(T)$ of $T$, i.e., $\delta(T) := \max_{u \neq v \in V(T)} d_T(u, v)/|u, v|$.

## 3 Finding the Closest Pair in Linear Time

To get familiar with the bounded dilation property, we show how to use it to speed up the computation of the closest pair $\mathrm{CP}(S)$ of a point set $S$. Let $T$ be a triangulation on $S$ and $d := \delta(T)$. Note that the shortest edge $xy$ of $T$ approximates $|\mathrm{CP}(S)|$ by a multiplicative factor of $d$, and thus we know $|xy|/d \leq |\mathrm{CP}(S)| \leq |xy|$. In order to find $\mathrm{CP}(S)$, we examine all paths of length at most $d|xy|$. This can be done by using Dijkstra's shortest path algorithm for each vertex $v$ of $T$, but stopping if a path exceeds length $d|xy|$.

**Algorithm 1** Computing $\mathrm{CP}(S)$

```
 1: Closest-pair($T$,$d$)
 2: Find the shortest edge $xy$ in $E(T)$.
 3: Delete all edges in $E(T)$ with length $> d \cdot |xy|$.
 4: Closest pair $\{p, q\} \leftarrow \{x, y\}$
 5: for all $w \in V(T)$ do
 6:     Use Dijkstra's algorithm to find the set of vertices $C$ that are connected with $w$ through a path of length at most $d \cdot |xy|$.
 7:     for all $v \in C$ do
 8:         if $|wv| < |pq|$ then
 9:             $\{p, q\} \leftarrow \{w, v\}$
10:         end if
11:     end for
12: end for
13: return $\{p, q\}$
```

**Theorem 1** *Given a bounded-dilation triangulation $T$, Algorithm 1 computes $\mathrm{CP}(S)$ in linear time.*

**Proof.** Let $xy$ be the shortest edge of $T$. The correctness follows from the fact that we examine for each vertex $v$ all paths of length less than $d \cdot |xy|$. By the dilation property we must encounter the closest pair.

For the running time we argue that there are only a constant number of edges with length less than $d \cdot |xy|$ incident to any vertex. Let $v \in S$ and let $D_v$ be the disk centered at $v$ with radius $|xy|/d$. Observe that there is no vertex $w \in S$ lying in $D_v$ or otherwise, by the dilation property, there must be a path between $w$ and $v$ of length less than $|xy|$. But this would contradict the minimality of $xy$.

Now, let $u$ be an arbitrary vertex and let $A$ be the annulus centered at $u$ with inner radius $|xy|$ and outer radius $d|xy|$. The area of $A$ is $O(|xy|)$. Since every vertex $v$ inside $A$ has an empty disk $D_v$ that covers a constant fraction of $A$, there can be only a constant number of such $v$'s.

Finally, consider the shortest path tree with root $u$ obtained by the execution Dijkstra's algorithm. Let $P$ be path from the the root in the tree. When stopping the computation of $P$ once its length exceeds $d|xy|$, the tree has depth at most $d + 1$. By the above discussion every inner node has constant degree. Thus, the time spend for every vertex is some constant dependent on $d$ only and the overall running time is $O(n)$. □

## 4 Bounded-Dilation Triangulations and $(c_1, c_2)$-cluster Trees

Let $S$ be a planar point set. As mentioned in the introduction, KL describe an algorithm for computing a $c$-cluster tree for $S$ from $\mathrm{EMST}(S)$ in linear time. We explain how to extend this algorithm to triangulations with bounded dilation. However, we will only be able to obtain a $(c_1, c_2)$-cluster tree, which is slightly weaker, though sufficient for all practical purposes. But first, we give some idea of how the KL-algorithm works: the key insight lies in the following characterization of $c$-clusters in terms of the EMST [3, Obs. 5].

**Observation 2** *Let $S$ be a planar point set and $G = \mathrm{EMST}(S)$. A subset $U \subseteq S$ is a $c$-cluster if and only if $G[U]$ is connected and all edges in $I_G(U)$ have length greater than $c \cdot \mathrm{rdiam}(U)$.*

We explain how KL use Observation 2 to find for a given vertex $v$ the smallest $c$-cluster $U$ that contains it, i.e., the parent of $v$ in the $c$-cluster tree. For this, we start at $v$ and explore the EMST until we find an appropriate subgraph that fulfills Observation 2.

**Algorithm 2** Computing the parent $c$-cluster for $v$.

```
 1: ParentCluster($G$, $v$):
 2: Set $U \leftarrow \{v\}$
 3: Queue $Q \leftarrow$ {shortest edge incident to $v$}
 4: Set $P \leftarrow$ {edges incident to $v$ that are not in $Q$}
 5: Set $D \leftarrow \{v\}$
 6: while $Q \neq \emptyset$ do
 7:     remove the first edge $uw$ from $Q$ (with $u \in U$)
 8:     add $w$ to $U$
 9:     update the $xy$-extremes in $D$
10:     add each edge $wz$ (except for $wu$) to $P$
11:     move edges in $P$ of length $< c \cdot \mathrm{rdiam}(U)$ to $Q$
12: end while
13: return $c$-cluster $U$
```

Initially, the set $U$ contains only the vertex $v$, and the algorithm adds to $U$ the closest neighbor $w$ of $v$ in the EMST $G$. Then, as long as $I_G(U)$ contains an edge $uw$ with $|uw| < c \cdot \mathrm{rdiam}(U)$ (and $u \in U$), the endpoint $w$ is added to $U$. Afterwards, all edges in $I_G(U)$ have length at least $c \cdot \mathrm{rdiam}(U)$, so Observation 2 guarantees that $U$ is a $c$-cluster. Note that $I_G(U)$ is represented by $P$ and $Q$, where $Q$ contains the short edges in $I_G(U)$ and $P$ the long edges.

By extending Algorithm 2, we can obtain the whole $c$-cluster tree: every time we find a $c$-cluster $U$, the induced subgraph $G[U]$ is contracted to obtain a smaller graph $G'$. Algorithm 2 is then applied to $G'$. We must make sure that all child clusters of $U$ are identified before the contraction. This is achieved by an appropriate recursion whenever we try to extend $U$ by a vertex that belongs to some child cluster of $U$ (this can be detected efficiently). See [3] for details.

We now show that Observation 2 also holds for general triangulations with bounded dilation, albeit with a slightly weaker conclusion. More precisely, the KL-algorithm produces a $(c_1, c_2)$-cluster tree when applied to such triangulations (see [6] for details).

**Lemma 3** *Let $T$ be a triangulation on a planar point set $S$ with constant dilation $d$. Let $G = \mathrm{MST}(T)$ and $c_2 > d$. Furthermore, let $U \subseteq S$ be a subset of $S$ such that $G[U]$ is connected. If every edge in $I_G(U)$ has length greater than $c_2 \cdot \mathrm{rdiam}(U)$, then $U$ is a $c_1$-cluster for $c_1 = 2(c_2 - 1)/(d + 1)$.*

**Proof.** Let $A := \{b \in \mathbb{R}^2 \mid d(U, b) < (c_2 - 1)\mathrm{rdiam}(U)\}$ be the region with distance less than $(c_2 - 1)\mathrm{rdiam}(U)$ from $U$ (marked by the dashed line in Fig. 1). We will show that all vertices in $S \cap A$ have distance at least $c_1 \cdot \mathrm{rdiam}(U)$ from $U$. Let $a \in S \setminus U$ be a vertex inside $A$. We claim that $a$ is not incident to $U$:

**Claim 4** *The triangulation $T$ contains no edge between $a$ and $U$.*

**Proof.** Suppose there is an edge $va \in E(T)$ with $v \in U$. As $d(U, a) < (c_2 - 1)\mathrm{rdiam}(U)$, the edge $va$ has length less than $c_2 \cdot \mathrm{rdiam}(U)$. Thus, $va$ is not an edge of $G$, since we assumed that all edges in $I_G(U)$ are longer than $c_2 \cdot \mathrm{rdiam}(U)$. Since $G$ is connected, it contains a path from $v$ to $a$ in $G$. This path must use an edge $e \in I_G(U)$. Replacing $e$ by $va$ yields a shorter spanning tree, contradicting the minimality of $G$. ☐

The vertex $a$ is not incident to $U$ in $T$, so the shortest path $P$ from $v$ to $a$ in $T$ uses an edge of $I_T(U)$. This implies that $P$ must leave $A$ at some point. Since $a \in A$, the path $P$ must reenter $A$. Let $h$ be the distance between the point where $P$ reenters $A$ and $a$ (see Figure 1). So,

$$d_T(v, a) \geq (c_2 - 1)\,\mathrm{rdiam}(U) + h. \qquad (1)$$

By the triangle inequality, we have $(c_2 - 1)\,\mathrm{rdiam}(U) \leq h + |va|$ and therefore

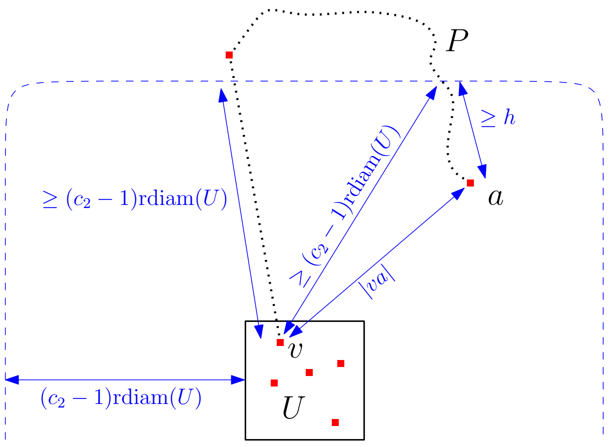

$P$
$\geq h$
$a$
$\geq (c_2-1)\mathrm{rdiam}(U)$
$\geq (c_2-1)\mathrm{rdiam}(U)$
$|va|$
$v$
$(c_2-1)\mathrm{rdiam}(U)$
$U$

Figure 1: The shortest path from $v$ to $a$ has length at least $2(c_2 - 1)\,\mathrm{rdiam}(U) - |va|$.

$h \geq (c_2 - 1)\,\mathrm{rdiam}(U) - |va|$. Plugging this into (1), we get $d_T(v, a) \geq 2(c_2 - 1)\,\mathrm{rdiam}(U) - |va|$. Since $T$ has dilation $d$, it follows that

$$d \cdot |va| \geq 2(c_2 - 1)\,\mathrm{rdiam}(U) - |va|$$
$$\Rightarrow |va| \geq (2(c_2 - 1)/(d + 1))\,\mathrm{rdiam}(U).$$

Thus, for every $a \in S \setminus U$, we have $d(U, a) \geq (2(c_2 - 1)/(d + 1))\,\mathrm{rdiam}(U) = c_1 \cdot \mathrm{rdiam}(U)$, so $U$ is a $c_1$-cluster. ☐

## 5 Running Time

To argue that their algorithm has linear running time KL used the following generalization of the fact that the EMST of a point set $S$ has constant degree [3]:

**Lemma 5** *Let $G = \mathrm{EMST}(S)$, $U \subseteq S$, and $c \geq 1$. If $G[U]$ is connected, then the number of edges in $I_G(U)$ with length greater than $c \cdot \mathrm{rdiam}(U)$ is constant.*

Given the same property for the MSTs of bounded-dilation triangulations, the analysis of the complete adapted algorithm would follow the one for the KL-algorithm. Unfortunately, Lemma 5 does not hold for for such MSTs in general: for every $m \in \mathbb{N}$, we construct a triangulation $T_m$ such that (i) $T_m$ has dilation at most 2; and (ii) the MST of $T_m$ has a vertex of degree $m$. Since a single vertex can have arbitrarily high degree, this holds also for each subgraph.

Let $w$ be a vertex and set $\alpha := \pi/6$. Choose $m$ vertices $v_1, \ldots, v_m$ in clockwise order such that $\angle v_i w v_{i+1} = \alpha/m$ and $|wv_i| = 3^{i-1}$. We add the edges $wv_1, wv_2, \ldots, wv_m$ to $T_m$. In order to ensure that these edges belong to $G = \mathrm{MST}(T_m)$, we need some edges that intersect the line segments $v_i v_{i+1}$. Otherwise, these segments would have to be in $T_m$ and also in $G$. Thus, we place $m - 1$ vertices $u_1, \ldots, u_{m-1}$ on the circle with center $w$ and radius $r = 3^m$ such

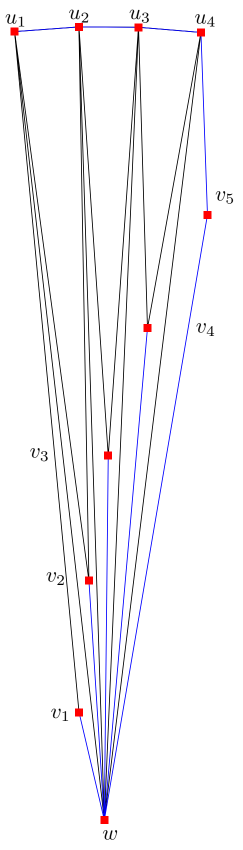


Figure 2: A triangulation with bounded dilation that produces a MST (blue edges) with a vertex of arbitrarily high degree.

that for each $i$ the line $wu_i$ bisects the angle $\angle v_i w v_{i+1}$. To complete $T_m$, we add the following edges for each $1 \le i \le m-1$: (i) $wu_i$; (ii) $v_i u_i$; (iii) $v_{i+1}u_i$; and (iv) $u_i u_{i+1}$. See Figure 2 shows $T_5$ (not drawn to scale).

By construction, we have $|v_i u_{i-1}|, |v_i u_i| > |wv_i|$ for all $i$, so all edges $wv_i$ are in $G$ and $w$ has degree $m$. It remains to show that $T_m$ has bounded dilation. Indeed, any two nonadjacent vertices $a, b$ in $T_m$ are connected by a path with at most two edges and $w$ as intermediate vertex. We show that the dilation between $a$ and $b$ is at most $2$. There are three cases:
**Case 1:** $a = u_i$ and $b = u_j$ with $i < j$. Let $\beta := \angle u_i w u_j$. Then $|u_i u_j| = 2r\sin(\beta/2)$. The length of the path $u_i, u_{i+1}, \dots, u_j$ is bounded by the length of the arc between $u_i$ and $u_j$ with center $w$, i.e., $d_T(u_i, u_j) \le \beta r$. Thus, the dilation between $u_i$ and $u_j$ is at most $(\beta r)/(2r\sin(\beta/2)) \le 2$, as $\beta < \pi/6$.
**Case 2:** $a = v_i$ and $b = v_j$. The largest dilation occurs when $v_i$ and $v_j$ are consecutive, i.e., $j = i+1$. Then $d_T(v_i, v_{i+1}) = |wv_i| + |wv_{i+1}| = 4|wv_i|$, by construction. By the triangle inequality $|v_i v_{i+1}| \ge |wv_{i+1}| - |wv_i| = 2|wv_i|$. The dilation is at most $2$.
**Case 3:** $a = u_i$ and $b = v_j$. The largest dilation occurs for $j = m$ and $i = m-2$. A calculation similar to Case 2 shows that the dilation is at most $2$. Thus, $T_m$ has dilation at most $2$, as claimed.

## 6 Conclusion

It remains as an open question, whether there are triangulations with bounded dilation that yield MSTs fulfilling Lemma 5, besides the Delaunay and the greedy triangulation (where the MST is just the EMST). These triangulations can be used to compute a hierarchical clustering of the point set in linear time. Unfortunately, the third popular bounded-dilation triangulation, the minimum weight triangulation, is NP-hard to compute and thus cannot be considered as reasonable input.

A very general candidate would be triangulations that fulfill the *diamond property*, i.e., there exists an angle $\alpha > 0$ such that for any edge $e$ in the triangulation one of the two isosceles triangles with base $e$ and base angle $\alpha$ must be empty. On the one hand, such triangulations have bounded dilation [2], on the other hand they admit a constant-degree subgraph $G'$ that still has bounded dilation (though with a slightly larger constant) and can be found in linear time [1]. Thus, $G'$ is not concerned by the given counterexample.

Finally, note that all steps related to the correctness of the adapted KL-algorithm in Section 4 work with a larger class of, not only triangulations, but even general planar straight-line graphs with bounded dilation. Thus, we can extend our question and ask what kind of planar straight-line graphs yield spanning trees with the necessary properties.

## References

[1] P. Bose, M. H. M. Smid, and D. Xu. Delaunay and diamond triangulations contain spanners of bounded degree. *IJCGA*, 19(2):119–140, 2009.

[2] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proc. International Symposium on Optimal Algorithms*, pages 168–192. 1989.

[3] D. Krznaric and C. Levcopoulos. Computing hierarchies of clusters from the Euclidean minimum spanning tree in linear time. In *Proc. 15th FSTTCS*, pages 443–455. 1995.

[4] D. Krznaric and C. Levcopoulos. Computing a threaded quadtree from the Delaunay triangulation in linear time. *Nordic J. Comput.*, 5(1):1–18, 1998.

[5] M. Löffler and W. Mulzer. Triangulating the square and squaring the triangle: Quadtrees and Delaunay triangulations are equivalent. *SIAM J. Comput.*, 41(4):941–974, 2012.

[6] P. Seiferth. Computational aspects of triangulations with constant dilation. Master's thesis, FU Berlin, 2012.

# Exploiting Air-Pressure to Map Floorplans on Point Sets

Stefan Felsner[*]

**Abstract.** We prove a conjecture of Ackerman, Barequet and Pinter. Every floorplan with $n$ segments can be embedded on every set of $n$ points in generic position. The construction makes use of area universal floorplans also known as area universal rectangular layouts.

The notion of area used in our context depends on a nonuniform density function. We, therefore, have to generalize the theory of area universal floorplans to this situation. The method is then used to prove a result about accommodating points in floorplans that is slightly more general than the conjecture of Ackerman et al.

## 1 Introduction

In our context a *floorplan* is a partition of a rectangle into a finite set of interior-disjoint rectangles. A floorplan is *generic* if it has no cross, i.e., no point where four rectangles of the partition meet. A *segment* of a floorplan is a maximal nondegenerate interval that belongs to the union of the boundaries of the rectangles. Segments are either horizontal or vertical. The segments of a generic floorplan are internally disjoint. Two floorplans $F$ and $F'$ are *weakly equivalent* if there exist bijections $\phi : S_H(F) \to S_H(F')$ and $\phi : S_V(F) \to S_V(F')$ between their horizontal and vertical segments such that segment $s$ has an endpoint on segment $t$ in $F$ iff $\phi(s)$ has an endpoint on $\phi(t)$. A set $P$ of points in $\mathbb{R}^2$ is *generic* if no two points from $P$ have the same $x$ or $y$ coordinate. Section 2 provides a more comprehensive overview of definitions and notions related to floorplans.

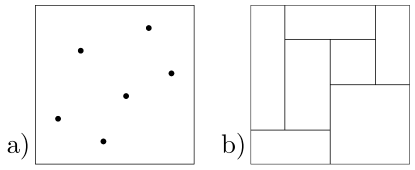


Figure 1: A generic set of six points and a generic floorplan with six segments.

Let $P$ be a set of $n$ points in a rectangle $R$ and let $F$ be a generic floorplan with $n$ segments. A *cover map* from $F$ to $P$ is a floorplan $F'$ weakly equivalent to $F$ with outer rectangle $R$ such that every segment of $F'$ contains exactly one point from $P$. Figure 2 shows an example.

[*]Institut für Mathematik, Technische Universität Berlin. Work was partially supported by DFG grant FE-340/7-2 and ESF EuroGIGA projects COMPOSE and GraDR.

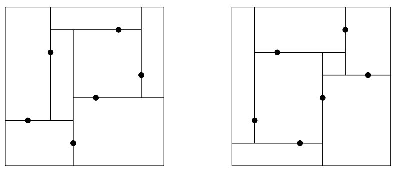


Figure 2: Two cover maps from the floorplan of Fig. 1.b to the point set of Fig. 1.a.

In this paper we answer a question of Ackerman et al. [1] by proving Theorem 1. The proof of the theorem and some variants and generalizations is the subject of Section 4.

**Theorem 1** *If $P$ is a generic set of $n$ points and $F$ is a generic floorplan with $n$ segments, then there is a cover map from $F$ to $P$.*

The proof is based on on results about area representations of floorplans. The following theorem is known, it has been proven with quite different methods, see [12], [10], [4].

**Theorem 2** *Let $F$ be a floorplan with rectangles $R_1, \ldots, R_{n+1}$, let $A$ be a rectangle and let $w : \{1, \ldots, n+1\} \to \mathbb{R}_+$ be a weight function with $\sum_i w(i) = \mathsf{area}(A)$. There exist a unique floorplan $F'$ contained in $A$ that is weakly equivalent to $F$ such that the area of the rectangle $\phi(R_i)$ in $F'$ is exactly $w(i)$.*

In Section 3 we prove the generalization of Theorem 2 that will be needed for the proof of Theorem 1. In the generalized theorem (Theorem 3) the weight of a rectangle is measured as integral over some density function instead of the area.

## 2 Floorplans and Graphs

A *floorplan* is a partition of a rectangle into a finite set of interiorly disjoint rectangles. From a given floorplan $F$ we can obtain several graphs and additional structure. We hint at some of these and close the section by introducing notions of equivalence for floorplans.

The **skeleton graph** $G_{\mathsf{skel}}(F)$ of $F$ has the corners of rectangles of $F$ as vertices. The edges of $G_{\mathsf{skel}}(F)$ are the connecting line segments.

The **rectangular dual** of $F$ is the graph $G^*(F)$ whose vertices are the rectangles of $F$ and edges joining pairs of rectangles that share a boundary segment.

The full paper is available as [5].

The **transversal structure** (also known as regular edge labeling) associated to an floorplan $F$ is an orientation and coloring of the edges of the extended dual $G^*_+(F)$. Transversal structures have been studied in [8], [9], and in [11]. It is known that every transversal structure on an inner triangulation $G$ of a 4-gon is induced by a floorplan.

The **segment contact graph** $G_{\mathsf{seg}}(F)$ of a floorplan $F$ is the bipartite planar graph whose vertices are the segments of $F$ and edges correspond to contacts between segments. From Figure 3 we see that $G_{\mathsf{seg}}(F)$ is indeed planar and that the faces of $G_{\mathsf{seg}}(F)$ are in bijection with the rectangles of $F$ and are uniformly of degree 4. Therefore $G_{\mathsf{seg}}(F)$ is a maximal bipartite planar graph, i.e., a quadrangulation.

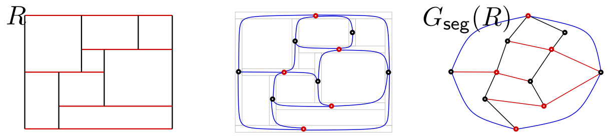


Figure 3: A floorplan $F$ and two drawings of its segment contact graph $G_{\mathsf{seg}}(F)$.

The **separating decomposition** associated to a floorplan is an orientation and coloring of the edges of the segment contact graph. Figure 4 shows an example. Separating decompositions have been studied in [3], [7], and [6]. To us they are of interest because every separating decomposition is induced by a floorplan $F$.

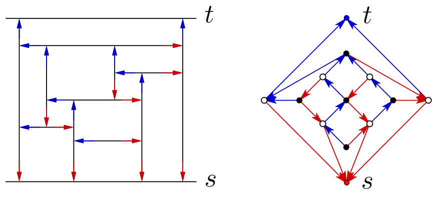


Figure 4: A floorplan $F$ and the separating decomposition induced by $F$ on its segment contact graph $Q$.

## 2.1 Notions of equivalence for floorplans

**Definition 1** *Two floorplans are* weakly equivalent *if they induce the same separating decomposition.*

**Definition 2** *Two floorplans are* strongly equivalent *if they induce the same transversal structure.*

Eppstein et al. [4] use the term *layout* instead of floorplan. Their equivalent layouts correspond to strongly equivalent floorplans and order-equivalent layouts to weakly equivalent floorplans. Asinowski et al. [2] study independent notions of $R$-equivalence and $S$-equivalence for floorplans.

## 3 Realizing Weighted Floorplans via Air-Pressure

In this section we sketch the proof of a generalization of Theorem 2 to situations where the “area” of a rectangle is replaced by the mass defined through a density distribution.

Let $\mu : [0,1]^2 \to \mathbb{R}_+$ be a density function on the unit square with mass 1, i.e., $\int_0^1\int_0^1 \mu(x,y)dxdy = 1$. We assume that $\mu$ can be integrated over axis aligned rectangles and all fibers $\mu_x$ and $\mu_y$ can be integrated over intervals. Moreover, we require that integrals over nontrivial rectangles and intervals are nonzero. The *mass* of an axis aligned rectangle $R \subseteq [0,1]^2$ is defined as $m(R) = \iint_R \mu(x,y)dxdy$.

**Theorem 3** *Let $\mu : [0,1]^2 \to \mathbb{R}_+$ be a density function on the unit square. If $F$ is a floorplan with rectangles $R_1, \dots, R_{n+1}$ and $w : \{1,\dots,n+1\} \to \mathbb{R}_+$ a positive weight function with $\sum_1^{n+1} w(i) = 1$ then there exists a unique floorplan $F'$ in the unit square that is weakly equivalent to $F$ such that $m(R_i) = w(i)$ for each rectangle $R_i$.*

The proof follows the air-pressure paradigm as proposed by Izumi,Takahashi and Kajitani [10]. We first describe the idea: Consider a realization of $F$ in the unit square and compare the mass $m(R_i)$ to the intended mass $w(i)$. The quotient of these two values can be interpreted as the pressure inside the rectangle. Integrating this pressure along a side of the rectangle yields the force by which $R_i$ is pushing against the segment that contains the side. The difference of pushing forces from both sides of a segment yields the effective force acting on the segment. The intuition is that shifting a segment in direction of the effective force yields a better balance of pressure in the rectangles. We show that iterating such improvement steps drives the realization of $F$ towards a situation with $m(R_i) = w(i)$ for all $i$, i.e., the procedure converges towards the floorplan $F'$ whose existence we want to show.

In [10] the air-pressure paradigm was used for situations where the mass of a rectangle is its area. The authors observed fast convergence experimentally but they had no proof of convergence. In the full paper we provide such a proof for the more general case of weights given by integrals over a density function.

A proof of Theorem 3 could also be given along the lines of the proof of Theorem 2 that has been given by Eppstein et al. in [4]. The proof there is quite compact. It has, however, the disadvantage that it is purely existential.

Let $R_i = [x_l, x_r] \times [y_b, y_t]$ be a rectangle of $F$. The pressure $p(i)$ in $R_i$ is the fraction of the intended mass $w(i)$ and the actual mass $m(R_i)$, i.e., $p(i) = \frac{w(i)}{m(R_i)}$. Let $s$ be a segment of $F$ and let $R_i$ be one of the rectangles with a side in $s$. Let $s$ be vertical with $x$-coordinate $x_s$ and let $s \cap R_i$ span the interval $[y_b(i), y_t(i)]$. The (undirected) *force imposed on $s$ by $R_i$* is the pressure $p(i)$ of $R_i$ times the density dependent length of the intersection.

$$f(s,i) = p(i) \int_{y_b(i)}^{y_t(i)} \mu_{x_s}(y)dy.$$

The *force acting on $s$* is obtained as a sum of the directed forces imposed on $s$ by incident rectangles.

$$f(s) = \sum_{R_i \text{ left of } s} f(s,i) - \sum_{R_i \text{ right of } s} f(s,i).$$

Symmetric definitions apply to horizontal segments.

### Balance for rectangles and segments

**Definition 3** *A segment $s$ is in* balance *if $f(s) = 0$. A rectangle $R_i$ is in* balance *if $p(i) = 1$, i.e., if $m(R_i) = w(i)$.*

**Lemma 4** *If all rectangles $R_i$ of $F$ are in balance, then all segments are in balance.*

**Proof.** Since all rectangles are in balance we can eliminate the pressures from the definition of the $f(s,i)$. With this simplification we get for a vertical segment $s$

$$f(s) = \sum_{R_i \text{ left}} \int_{y_b(i)}^{y_t(i)} \mu_{x_s}(y)dy - \sum_{R_j \text{ right}} \int_{y_b(j)}^{y_t(j)} \mu_{x_s}(y)dy.$$

Hence $f(s) = M_s - M_s = 0$, where $M_s$ is the integral of the fiber density $\mu_{x_s}$ along $s$. The symmetric argument applies to horizontal segments. $\square$

Interestingly, the converse of the lemma also holds.

**Proposition 1** *If all segments of $F$ are in balance, then all rectangles are in balance.*

The proof has been omitted in this extended abstract.

### Balancing segments and optimizing the entropy

**Proposition 2** *If a segment $s$ of $F$ is unbalanced, then then we can keep all the other segments at their position and shift $s$ parallel to a position where it is in balance. The resulting floorplan $F'$ is weakly equivalent to $F$.*

The proof has been omitted in this extended abstract.

**Definition 4** *The* entropy *of a rectangle $R_i$ of $F$ is defined as $-w(i)\log p(i)$. The* entropy *of the floorplan $F$ is*

$$E = \sum_i -w(i)\log p(i)$$

The proof of Theorem 3 is completed in five steps:

(1) The entropy $E$ is always nonpositive.

(2) $E = 0$ if and only if all rectangles $R_i$ of $F$ are in balance.

(3) Shifting an unbalanced segment $s$ into its balance position increases the entropy.

(4) The process of repeatedly shifting unbalanced segments into their balance position makes $F$ converge to a floorplan $F'$ such that the entropy of $F'$ is zero.

(5) The solution is unique.

## 4 Accommodating Floorplans on Point Sets

Let $P$ be a generic set of $n$ points in a rectangle $\mathcal{R}$. Let $F$ be a generic floorplan and $S$ be a subset of the segments of $F$ of size $n$. A *cover map* from $(F,S)$ to $P$ is a floorplan $F'$ with outer rectangle $\mathcal{R}$ that is weakly equivalent to $F$ such that every segment from $S' = \phi(S)$ contains exactly one point from $P$. The main result in this paper is the following generalization of Theorem 1.

**Theorem 5** *If $P$ is a generic set of $n$ points in a rectangle $\mathcal{R}$ and $F$ is a generic floorplan with a prescribed subset $S$ of the segments of size $n$, then there is a cover map $F'$ from $(F,S)$ to $P$.*

The idea for the proof is to use Theorem 3 as a tool. To this end we first transform the point set $P$ into a suitable density distribution $\mu = \mu_P$ inside $\mathcal{R}$. This density is defined as the sum of a uniform distribution $\mu_1$ with $\mu_1(q) = 1/\mathsf{area}(\mathcal{R})$ for all $q \in \mathcal{R}$ and a distribution $\mu_2$ that represents the points of $P$. Choose some $\Delta > 0$ such that for all $p, p' \in P$ we have $|x_p - x_{p'}| > 3\Delta$ and $|y_p - y_{p'}| > 3\Delta$, this is possible because $P$ is generic. Define $\mu_2 = \sum_{p\in P} \mu_p$ where $\mu_p(q)$ takes the value $(\Delta^2\pi)^{-1}$ on the disk $D_\Delta(p)$ of radius $\Delta$ around $p$ and value 0 for $q$ outside of this disk.

For a density $\nu$ over $\mathcal{R}$ and a rectangle $R \subseteq \mathcal{R}$ we let $\nu(R)$ be the integral of the density $\nu$ over $R$. Using this notation we can write $\mu_1(\mathcal{R}) = 1$ and $\mu_p(\mathcal{R}) = 1$ for all $p \in P$, hence the total mass of $\mathcal{R}$ is $\mu(\mathcal{R}) = 1 + n$.

Next we transform the floorplan $F$ into a floorplan $F_S$ depending on the set $S$ of segments that has to cover the points of $P$. To this end we replace every segment in $S$ by a thin rectangle, see Figure 5. In [2] this doubling of a segment is call inflation. Let $\mathcal{S}$ be the set of new rectangles obtained by infating segments from $S$.

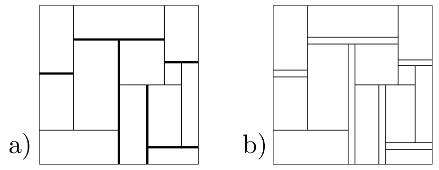


Figure 5: Floorplans $F$ with as prescribed subset $S$ of segments (bold and gray) and the floorplan $F_S$ obtained by doubling the segments of $S$.

Define weights for the rectangles of $F_S$ as follows. If $F_S$ has $r$ rectangles we define $w(R) = 1 + 1/r$ if $R \in \mathcal{S}$ and $w(R) = 1/r$ for all the rectangles of $F_S$ that came from rectangles of $F$. Note that the total weight, $\sum_R w(R) = 1 + n$, is in correspondence to the total mass $\mu(\mathcal{R})$.

The data $\mathcal{R}$ with $\mu$ and $F_S$ with $w$ constitute, up to scaling of $\mathcal{R}$ and $w$, a set of inputs for Theorem 3. From the conclusion of the theorem we obtain a floorplan $F'_S$ weakly equivalent to $F_S$ such that

$m(R) = \iint_R \mu(x,y)dxdy = w(R)$ for all rectangles $R$ of $F'_S$.

The definition of the weight function $w$ and the density $\mu$ is so that $F'_S$ should be close to a cover map from $(F,S)$ to $P$: Only the rectangles $R \in \mathcal{S}$ that have been constructed by inflating segments may contain a disk $D_\Delta(p)$ and each of these rectangles may contain at most one of the disks. This suggests a correspondence $\mathcal{S} \leftrightarrow P$. However, a rectangle $R \in \mathcal{S}$ may use parts of several discs to accumulate mass. To find a correspondence between $\mathcal{S}$ and $P$ we define a bipartite graph $G$ whose vertices are the points in $P$ and the rectangles in $\mathcal{S}$:

- A pair $(p,R)$ is an edge of $G$ iff $R \cap D_\Delta(p) \neq \emptyset$ in $F'_S$.

The proof of the theorem is completed by proving two claims:

- $G$ admits a perfect matching.
- From $F'_S$ and a perfect matching $M$ in $G$ we can produce a floorplan $F'$ that realizes the cover map from $(F,S)$ to $P$.

For the first of the claims we check Hall's matching condition: Consider a subset $A$ of $\mathcal{S}$. Since $F_S$ is realizing the prescribed weights we have $m(A) = \mu(A) = \sum_{R\in A}\mu(R) = \sum_{R\in A} w(R) = |A|(1+1/r)$. Since $\mu_1(A) < 1$ and $\mu_p(A) \leq 1$ for all $p \in P$ there must be at least $|A|$ points $p \in P$ with $\mu_p(A) > 0$, these are the points that have an edge to a rectangle from $A$ in $G$. We have thus shown that every set $A$ of inflated segments is incident to at least $|A|$ points in $G$, hence, there is an injective mapping $\alpha : \mathcal{S} \to P$ such that $R \cap D_\Delta(\alpha(R)) \neq \emptyset$ in $F'_S$ for all $R \in \mathcal{S}$.

The construction of the floorplan $F'$ that realizes the cover map from $(F,S)$ to $P$ is done in four steps indicated in Figure 6.

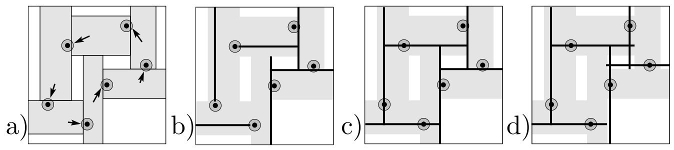


Figure 6: a) A solution $F'_S$ for the instance from Fig. 1. The arrows indicate a matching $\alpha$. b) Segments $s \in S$ shifted to their optimal position in $R_s$. c) Enlarged segments recover the contacts. d) Some segments $s$ are moved outside $R_s$ to cover the corresponding points $\alpha(R_s)$. Small final adjustments (clipping and enlarging) yield $F'$.

The topic of [1] was the study of the number $Z(P)$ of rectangulations of a generic point set $P$. This is the total number of cover maps from floorplans with $n$ segments to a generic point set $P$ with $n$ points. Theorem 5 implies that this number is at least as large as the number of weak equivalence classes of floorplans. This is the Baxter number $B_{n+1}$ which is known to be of order $\Theta(8^{n+1}/(n+1)^4)$. In [1] an upper bound for $Z(P)$ of order $O(20^n/n^4)$ is shown. To improve this bound remains an intriguing problem.

## References

[1] E. Ackerman, G. Barequet, and R. Y. Pinter, *On the number of rectangulations of a planar point set*, J. Combin. Theory Ser. A, 113 (2006), pp. 1072–1091.

[2] A. Asinowski, G. Barequet, M. Bousquet-Mélou, T. Mansour, and R. Pinter, *Orders induced by segments in floorplan partitions and (2-14-3,3-41-2)-avoiding permutations*, 2010.

[3] H. de Fraysseix and P. Ossona de Mendez, *On topological aspects of orientation*, Discr. Math., 229 (2001), pp. 57–72.

[4] D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek, *Area-universal and constrained rectangular layouts*, SIAM J. Computing, 41 (2012), pp. 537–564.

[5] S. Felsner, *Exploiting air-pressure to map floorplans on point sets*, 2012. `http://page.math.tu-berlin.de/~felsner/Paper/flop.pdf`.

[6] S. Felsner, É. Fusy, M. Noy, and D. Orden, *Bijections for Baxter families and related objects*, Journal of Comb. Theory A, 18 (2011), pp. 993–1020.

[7] S. Felsner, C. Huemer, S. Kappes, and D. Orden, *Binary labelings for plane quadrangulations and their relatives*, Discr. Math. and Theor. Comp. Sci., 12:3 (2010), pp. 115–138.

[8] É. Fusy, *Combinatoire des cartes planaires et applications algorithmiques*, PhD thesis, LIX Ecole Polytechnique, 2007. `http://www.lix.polytechnique.fr/~fusy/Articles/these_eric_fusy.pdf`.

[9] É. Fusy, *Transversal structures on triangulations: A combinatorial study and straight-line drawings*, Discr. Math., 309 (2009), pp. 1870–1894.

[10] T. Izumi, A. Takahashi, and Y. Kajitani, *Air-pressure model and fast algorithms for zero-wasted-area layout of general floorplan*, IEICE Trans. Fundam. Electron., Commun. and Comp. Sci., E81–A (1998), pp. 857–865.

[11] G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, Theor. Comput. Sci., 172 (1997), pp. 175–193.

[12] S. Wimer, I. Koren, and I. Cederbaum, *Floorplans, planar graphs, and layouts*, IEEE Transactions on Circuits and Systems, 35 (1988), pp. 267 –278.

# Book-Embeddings of Iterated Subdivided-Line Graphs

Toru Hasunuma *

## Abstract

We introduce a new operation "Γ" on graphs called the subdivided-line graph operation and define the $n$-iterated subdivided-line graph $\Gamma^n(G)$. We then present upper and lower bounds on the pagenumber of $\Gamma^n(G)$. Our upper bound depends only on $G$ and is independent of the number $n$ of iterations.

Using the subdivided-line graph operation, we can construct Sierpiński-like graphs. Applying our results on iterated subdivided-line graphs, we obtain upper bounds on the pagenumbers of the Sierpiński graphs which are at most two times the optimums.

## 1 Introduction

A *book* consists of a line called the *spine*, and half planes called *pages* sharing the spine as a common boundary. A *k-page book-embedding* of a graph $G = (V, E)$ is defined by an assignment of the vertices to distinct points on the spine, i.e., a vertex-ordering $\sigma : V(G) \mapsto \{1, 2, \ldots, |V(G)|\}$, and an assignment of the edges to one of $k$ pages so that no two edges assigned to the same page cross, i.e., an edge-assignment $\rho : E(G) \mapsto \{1, 2, \ldots, k\}$ such that for each $i \in \{1, 2, \ldots, k\}$ and any two edges $\{u, v\}, \{x, y\} \in \rho^{-1}(i)$, it does not hold that $\sigma(u) < \sigma(x) < \sigma(v) < \sigma(y)$. The minimum number of pages for a book-embedding of $G$ is the *pagenumber* $\mathsf{pn}(G)$ of $G$. Figure 1 shows an example of a 3-page book-embedding of a graph, where the spine is drawn as the horizontal line, normal lines above (resp., below) the spine indicate the edges assigned to the first (resp., second) page, and the edge drawn as a thick line is assigned to the third page.

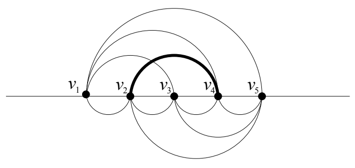


Figure 1: A 3-page book-embedding of a graph.

Book-embeddings are motivated by several areas of computer science [3, 9, 11]. In particular, book-embeddings of interconnection networks have applications to the Diogenes approach proposed by Rosenberg [9] to fault-tolerant processor arrays. Also, Wood [11] showed that book-embeddings can be applied to three-dimensional graph drawings. Until now, book-embeddings have been studied for many graph classes: complete (bipartite) graphs, butterfly networks, trees, grids, X-trees, (incomplete) hypercubes, de Bruijn and Kautz graphs, shuffle-exchange graphs, planar graphs, genus-$g$ graphs, bandwidth-$k$ graphs, $k$-trees, and iterated line digraphs (e.g., see [4] for the references of these results).

In this paper, we newly introduce an operation "Γ" called the subdivided-line graph operation and define the $n$-iterated subdivided-line graph $\Gamma^n(G)$ as the graph obtained from $G$ by iteratively applying the operation $n$ times. We then study book-embeddings of iterated subdivided-line graphs and present upper and lower bounds on $\mathsf{pn}(\Gamma^n(G))$. In particular, our upper bound depends only on $G$ and is independent of the number $n$ of iterations.

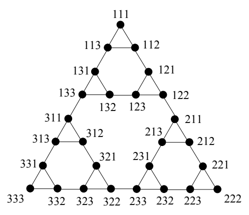


Figure 2: The Sierpiński graph $S(3, 3)$.

The Sierpiński graphs were introduced by Klavžar and Milutinović [7]. Figure 2 shows the Sierpiński graph $S(3, 3)$ (see Section 2 for the precise definition). The Sierpiński graph $S(n, k)$ has a recursive structure, i.e., $S(n, k)$ can be constructed from $k$ copies of $S(n - 1, k)$ by joining vertices with degree $k - 1$ in a fashion of the complete graph with $k$ vertices. Since $S(n, k)$ is not regular, several regular variations called the extended Sierpiński graphs $S^{+}(n, k)$, $S^{++}(n, k)$ were also introduced by Klavžar and Mohar [8]. Because of their interesting self-similar structures with relations to the problem of the Tower of Hanoi, various properties on the Sierpiński-like graphs have been investigated (e.g., see [6, 8]). On the other hand, the WK-recursive networks have independently been proposed by Vecchia and Sanges [10] as interconnection networks, and their topological properties have been investigated in [2]. The WK-recursive network and the Sierpiński graph have very similar structures, and

*Institute of Socio-Arts and Sciences, The University of Tokushima, `hasunuma@ias.tokushima-u.ac.jp`

the difference between them is the existence of "open edges" incident to each extreme vertex in the WK-recursive network. By deleting such open edges, they become isomorphic.

Using the subdivided-line graph operation, we can construct Sierpiński-like graphs. Namely, the class of iterated subdivided-line graphs generalizes the class of Sierpiński graphs. Applying our results on iterated subdivided-line graphs, upper and lower bounds on the pagenumbers of the Sierpiński graphs are obtained. Our upper bounds for the Sierpiński graphs are at most two times the optimums.

## 2 Preliminaries

Throughout the paper, a graph may have self-loops but not multiple edges, unless otherwise stated. Let $G = (V, E)$ be a graph. The number of self-loops in $G$ is denoted by $\ell(G)$. For $v \in V(G)$, let $N_G(v)$ and $E_G(v)$ be the set of vertices adjacent to $v$ and the set of edges incident to $v$ in $G$, respectively. The degree $d(v)$ of $v$ in $G$ is $|E_G(v)|$, i.e., $d(v) = |E_G(v)|$. Note that in this paper, if $v$ has a self-loop, then we count it one in $d(v)$ instead of two. The complete graph with $k$ vertices is denoted by $K_k$. The Sierpiński graph $S(n, k)$ is the graph with the vertex set consisting of all $n$-tuples of $k$ numbers $1, 2, \ldots, k$ and in which two vertices $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are adjacent if and only if there exists an integer $j$, where $1 \leq j \leq n$, such that $u_i = v_i$ for $1 \leq i < j$, $u_j \neq v_j$, and $u_i = v_j$, $v_i = u_j$ for $j + 1 \leq i \leq n$. For $1 \leq i \leq k$, a vertex $(i, i, \ldots, i)$ is called an *extreme vertex.* An extended Sierpiński graph $S^+(n, k)$ is obtained from $S(n, k)$ by adding a new vertex and joining it to every extreme vertex, while another extended Sierpiński graph $S^{++}(n, k)$ is constructed from $k + 1$ copies of $S(n-1, k)$ by joining their extreme vertices in a $K_{k+1}$-fashion.

## 3 The Subdivided-Line Graph Operation

The *line graph* $L(G)$ of $G$ is the graph whose vertex set is $E(G)$ and in which two distinct vertices $\{u, v\}$ and $\{x, y\}$ are adjacent if and only if they are adjacent in $G$, i.e., $\{u, v\} \cap \{x, y\} \neq \emptyset$. Besides, a vertex $\{w, w\}$ corresponding to a self-loop in $G$ also has a self-loop in $L(G)$. Let $e = \{x, y\} \in E(G)$. Then, let $G_e$ be the graph with $V(G_e) = V(G) \cup \{v_e\}$, where $v_e \notin V(G)$, and $E(G_e) = (E(G) \setminus \{\{x, y\}\}) \cup \{\{x, v_e\}, \{v_e, y\}\}$. We say that $G_e$ is obtained from $G$ by *elementary subdividing* the edge $e$. The *barycentric subdivision* $B(G)$ of $G$ is the graph obtained from $G$ by elementary subdividing every edge of $G$ except for self-loops.

**Definition 1** *Let $G$ be a graph. The subdivided-line graph $\Gamma(G)$ of $G$ is defined to be the line graph of the barycentric subdivision of $G$. i.e., $\Gamma(G) = L(B(G))$.*

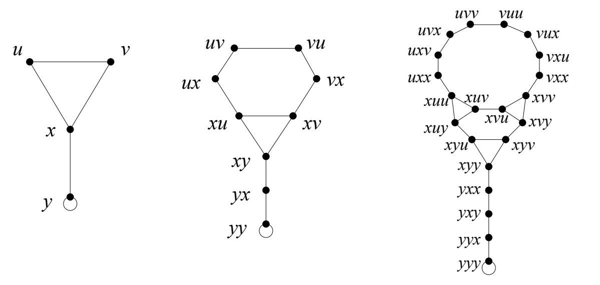


Figure 3: $G$, $\Gamma(G)$, and $\Gamma^2(G)$, where the vertex-labeling of $\Gamma^2(G)$ follows those for iterated subdivided-line graphs.

When we consider "$\Gamma$" as an operation on graphs, we call it the *subdivided-line graph operation.* Each vertex in $\Gamma(G)$ can be denoted by the ordered pair of vertices. Namely, for each non-loop edge $\{u, v\}$ of $G$, there exist two corresponding vertices $uv, vu$ in $\Gamma(G)$. For a self-loop $\{w, w\}$ of $G$, the only corresponding vertex in $\Gamma(G)$ is $ww$. Note that a vertex of the form $ww$ also has a self-loop in $\Gamma(G)$. Two distinct vertices $uv, xy$ in $\Gamma(G)$ are adjacent if and only if either $u = x$, or $u = y$ and $v = x$ (see the middle graph in Figure 3). The edge set of $\Gamma(G)$ is naturally divided into two categories. An edge joining vertices of the forms $uv$ and $vu$ is corresponding to the original edge $\{u, v\}$ in $G$, while an edge joining vertices $uv$ and $ux$, where $v \neq x$, is newly generated in $\Gamma(G)$. Then, we call edges of the former type *original edges* and edges of the latter type *generated edges.* Note that a self-loop at a vertex $ww$ is an original edge. The subgraph of $\Gamma(G)$ induced by the set of generated edges is the disjoint union of complete graphs, i.e., $\cup_{v \in V(G)} K_{d(v)}$. We denote by $K(v)$ the complete graph induced by a set $\{\{vw, vw'\} \mid w, w' \in N_G(v), w \neq w'\}$ of generated edges. Original edges of the form $\{vw, wv\}$, where $w \in N_G(v)$, are injectively incident to vertices of $K(v)$. Thus, every vertex of $K(v)$ has degree $d(v)$.

Using the subdivided-line graph operation, we define the iterated subdivided-line graphs.

**Definition 2** *The $n$-th iterated subdivided-line graph $\Gamma^n(G)$ of $G$ is the graph obtained from $G$ by iteratively applying the subdivided-line graph operation $n$ times.*

For $n \geq 1$, each vertex of $\Gamma^n(G)$ can be expressed by a sequence $v_0 v_1 \cdots v_n$ of vertices of $G$, where $v_1 \cdots v_n$ is any sequence on $N_G(v_0)$. Intuitively, we can say that each vertex $v$ in $G$ is expanded to $d(v)^n$ vertices in $\Gamma^n(G)$. Thus, $\Gamma^n(G)$ has $\sum_{v \in V(G)} d(v)^n$ vertices. Two vertices $u_0 u_1 \cdots u_n$ and $v_0 v_1 \cdots v_n$ are adjacent if and only if there exists an integer $0 \leq h \leq n$ such that $u_0 \cdots u_{h-1} = v_0 \cdots v_{h-1}$, $u_h \neq v_h$, and $u_j = v_h, v_j = u_h$ for $h < j \leq n$ (see the right graph in Figure 3). Such an $h$ for adjacent two vertices

$u_0u_1\cdots u_n$ and $v_0v_1\cdots v_n$ is the *level* of the edge joining the vertices. Note that any edge with level $n$ is a generated edge, while any edge with level $h < n$ is an original edge corresponding to an edge in $\Gamma^h(G)$. For any vertex $v_0v_1\cdots v_n$ in $\Gamma^n(G)$, the degree is equal to $d(v_0)$. Hence, $\Gamma^n(G)$ has $\frac{1}{2}(|\ell(G)|+\sum_{v\in V(G)} d(v)^{n+1})$ edges. Note that $\ell(G) = \ell(\Gamma^n(G))$. Besides, a vertex $v_0v_1\cdots v_n$ is incident to $d(v_0)-1$ edges with level $n$ and one edge with level $h < n$.

Let $\delta_G$ and $\Delta_G$ denote the minimum degree and the maximum degree of $G$, respectively. Also, let $\operatorname{diam}(G)$, $\kappa(G)$, and $\bar{\kappa}(G)$ denote the diameter, the vertex-connectivity, and the edge-connectivity of $G$, respectively. As fundamental properties of iterated subdivided-line graphs, it holds that $\delta_{\Gamma^n(G)} = \delta_G$, $\Delta_{\Gamma^n(G)} = \Delta_G$, $\operatorname{diam}(\Gamma^n(G)) \le 2^n(\operatorname{diam}(G)+1)-1$, and $\kappa(\Gamma^n(G)) = \bar{\kappa}(\Gamma^n(G)) = \bar{\kappa}(G)$ ([5]).

## 4 Upper and Lower Bounds on the Pagenumbers of Iterated Subdivided-Line Graphs

**Theorem 1** *Let $n \ge 1$. If $\Delta_G \le 2$, then $\mathsf{pn}(\Gamma^n(G)) = 1$. If $\Delta_G = 3$, then $\mathsf{pn}(\Gamma^n(G)) \le \mathsf{pn}(G)+1$. If $\Delta_G \ge 4$, then*

$$\left\lceil\frac{\Delta_G}{2}\right\rceil \le \mathsf{pn}(\Gamma^n(G)) \le \left\lceil\frac{\Delta_G}{2}\right\rceil + \max\left\{\mathsf{pn}(G), \left\lceil\frac{\Delta_G}{2}\right\rceil\right\}.$$

**Proof.** If $\Delta_G = 1$, i.e., $G \cong K_2$, then $\Gamma^n(G) \cong K_2$. When $\Delta_G = 2$, $\Gamma^n(G)$ is either a cycle or a path with a self-loop at each end-vertex and thus $\mathsf{pn}(\Gamma^n(G)) = 1$. For $\Delta_G \ge 3$, we show that $\mathsf{pn}(K_{\Delta_G}) \le \mathsf{pn}(\Gamma^n(G)) \le \mathsf{pn}(K_{\Delta_G})+\max\{\mathsf{pn}(G), \mathsf{pn}(K_{\Delta_G})\}$. Clearly, $\mathsf{pn}(K_3) = 1$. Besides, $\mathsf{pn}(K_k) = \lceil\frac{k}{2}\rceil$ for $k \ge 4$, which has been shown in [1]. In what follows, we consider the case that $\Delta_G \ge 4$, i.e., $\mathsf{pn}(K_{\Delta_G}) = \lceil\frac{\Delta_G}{2}\rceil$. The case that $\Delta_G = 3$ follows from the same discussion by simply replacing $\lceil\frac{\Delta_G}{2}\rceil$ by 1.

The lower bound of $\lceil\frac{\Delta_G}{2}\rceil$ follows from the fact that $\Gamma^n(G)$ contains $K_{\Delta_G}$. To show the upper bound, we inductively construct a $(\lceil\frac{\Delta_G}{2}\rceil+\max\{\mathsf{pn}(G), \lceil\frac{\Delta_G}{2}\rceil\})$-page book-embedding of $\Gamma^n(G)$. Let $\sigma$ and $\rho$ be the vertex-ordering and edge-assignment for a $\mathsf{pn}(G)$-page book-embedding of $G$, respectively. We define the vertex-ordering $\sigma_1$ of $\Gamma(G)$ as follows. Let $v \in V(G)$ and $N_G(v) = \{w_1, w_2, \ldots, w_{d(v)}\}$ such that $\sigma(w_1) < \cdots < \sigma(w_t) \le \sigma(v) < \sigma(w_{t+1}) < \cdots < \sigma(w_{d(v)})$. Define $\sigma_1(vw_i) = M(v)+t-i+1$ for $1 \le i \le t$ and $\sigma_1(vw_i) = M(v)+d(v)-i+t+1$ for $t < i \le d(v)$, where $M(v) = \sum_{1\le j<\sigma(v)} d(\sigma^{-1}(j))$. Under this vertex-ordering, we assign the edges of $\Gamma(G)$ to pages as follows. Each original edge is assigned to the same page according to the edge-assignment $\rho$. Any two original edges of $\Gamma(G)$ cross if and only if the corresponding edges of $G$ cross. Thus, all the original edges are correctly assigned to $\mathsf{pn}(G)$ pages. For each $K(v)$, the edges are assigned to pages with labels from $\mathsf{pn}(G)+1$ to $\mathsf{pn}(G)+\lceil\frac{d(v)}{2}\rceil$ according to the $\lceil\frac{d(v)}{2}\rceil$-page book-embedding of $K_{d(v)}$. For any two $K(v)$ and $K(v')$, any edge in $K(v)$ and any edge in $K(v')$ do not cross. Therefore, we totally need $\mathsf{pn}(G)+\lceil\frac{\Delta_G}{2}\rceil$ pages.

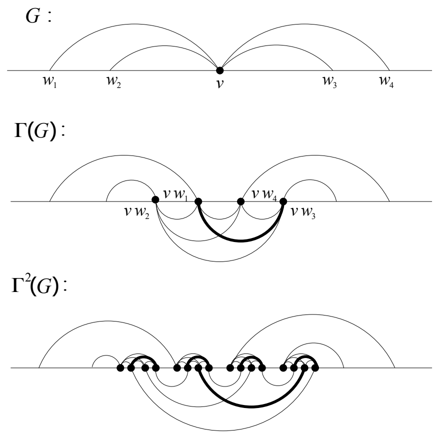


Figure 4: Parts of book-embeddings of $G$, $\Gamma(G)$, and $\Gamma^2(G)$ with $\Delta_G = 4$, where normal (resp., thick) edges above the spine are assigned to the first (resp., second) page, and normal (resp., thick) edges below the spine are assigned to the third (resp., fourth) page.

We divide the set of pages used for our book-embedding into two classes $A$ and $B$. Namely, let $A = \{1, 2, \ldots, \max\{\mathsf{pn}(G), \lceil\frac{\Delta_G}{2}\rceil\}\}$ and $B = \{\max\{\mathsf{pn}(G), \lceil\frac{\Delta_G}{2}\rceil\}+1, \ldots, \max\{\mathsf{pn}(G), \lceil\frac{\Delta_G}{2}\rceil\}+\lceil\frac{\Delta_G}{2}\rceil\}$. Now, let $n \ge 2$ and assume that $\Gamma^{n-1}(G)$ can be embedded in $\max\{\mathsf{pn}(G), \lceil\frac{\Delta_G}{2}\rceil\}+\lceil\frac{\Delta_G}{2}\rceil$ pages such that all the edge with level $n-1$ are assigned to pages in $B$ (resp., $A$) if $n$ is even (resp., odd). Let $\sigma_{n-1}$ and $\rho_{n-1}$ be the vertex-ordering and edge-assignment for such a book-embedding of $\Gamma^{n-1}(G)$, respectively. Define the vertex-ordering $\sigma_n$ for $\Gamma^n(G)$ as follows. Let $v = v_0v_1\cdots v_{n-1} \in V(\Gamma^{n-1}(G))$ and $N_G(v_0) = \{w_1, w_2, \ldots, w_{d(v_0)}\}$ such that $\sigma_{n-1}(v_0\cdots v_{n-2}w_1) < \cdots < \sigma_{n-1}(v_0\cdots v_{n-2}w_t) < \sigma(v) < \sigma_{n-1}(v_0\cdots v_{n-2}w_{t+1}) < \cdots < \sigma_{n-1}(v_0\cdots v_{n-2}w_{d(v_0)})$. Then we order the vertex $v_0\cdots v_{n-1}v_{n-1}$ which corresponds to the original edge incident to $v$ as the first vertex in the $d(v_0)$ vertices of $K(v)$. The remaining vertices are ordered similarly to the case $n = 1$. Let $M(v) = \sum_{1\le j<\sigma_{n-1}(v)} d(p(\sigma_{n-1}^{-1}(j)))$, where for each vertex $u = u_0u_1\cdots u_{n-1}$ of $\Gamma^{n-1}(G)$ $p(u)$ denotes the vertex $u_0$ of $G$. Then, we define $\sigma_n$ as follows. $\sigma_n(v_0\cdots v_{n-1}v_{n-1}) = M(v)+1$, $\sigma_n(v_0\cdots v_{n-1}w_i) = M(v)+t-i+2$ for $1 \le i \le t$, and $\sigma_n(v_0\cdots v_{n-1}w_i) = M(v)+d(v_0)-i+t+1$ for $t < i \le d(v_0)$. Then, we assign all the generated edges to pages in $A$ (resp., $B$) according to the $\lceil\frac{d(v_0)}{2}\rceil$-page book-embedding of $K(v_0v_1\cdots v_{n-1})$ for each vertex $v_0v_1\cdots v_{n-1}$ if $n$ is even (resp., odd). Assignment of all the original edges

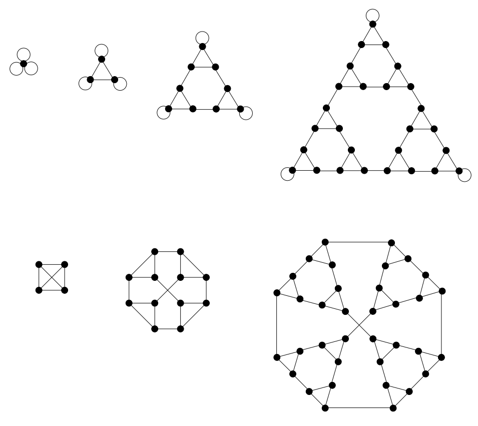


Figure 5: $\Gamma^i(K_1^3)$ for $i \leq 3$ and $\Gamma^i(K_4)$ for $i \leq 2$.

are the same as $\rho_{n-1}$. By the vertex-ordering $\sigma_n$, any two original edges of $\Gamma^n(G)$ cross if and only if the corresponding edges of $\Gamma^{n-1}(G)$ cross. Since each vertex of the form $v_0v_1 \cdots v_{n-1}v_{n-1}$ is ordered as the first in the $d(v_0)$ vertices of $K(v_0v_1 \cdots v_{n-1})$, any generated edge and any original edge do not cross. Besides, for any two vertices $v_0v_1 \cdots v_{n-1}$ and $v'_0v'_1 \cdots v'_{n-1}$ of $\Gamma^{n-1}(G)$, any edge in $K(v_0v_1 \cdots v_{n-1})$ and any edge in $K(v'_0v'_1 \cdots v'_{n-1})$ do not cross. Therefore, $\Gamma^n(G)$ can be embedded in a book without increasing the number of pages used in the book-embedding of $\Gamma^{n-1}(G)$ (see Figure 4). $\square$

## 5 Book-Embeddings of Sierpiński-Like Graphs

Let $S^\circ(n,k)$ be the graph obtained from the Sierpiński graph $S(n,k)$ by adding a self-loop to each extreme vertex. Then, we have the following lemma, where $K_1^k$ and $K_k^\circ$ denote the graph with one vertex and $k$ self-loops and the complete graph with a self-loop at each vertex, respectively. Figure 5 shows examples for $S^\circ(n,3)$ and $S^{++}(n,3)$.

**Lemma 2** *It holds that* $\Gamma^n(K_1^k) = \Gamma^{n-1}(K_k^\circ) \cong S^\circ(n,k)$ *and* $\Gamma^{n-1}(K_{k+1}) \cong S^{++}(n,k)$ *for* $n \geq 1$.

From Theorem 1 and Lemma 2, the following results are obtained. The lower bound of 2 on $\mathsf{pn}(S(n,3))$ follows from the fact that $S(n,3)$ is not outerplanar for $n \geq 3$.

**Theorem 3** *Let* $n \geq 1$. *It holds that* $\mathsf{pn}(S(n,3)) = 2$ *for* $n \geq 3$ *and* $\left\lceil \frac{k}{2} \right\rceil \leq \mathsf{pn}(S(n,k)) \leq 2\left\lceil \frac{k}{2} \right\rceil$ *for* $k \geq 4$.

By the constructions of book-embeddings shown in Theorem 1, each vertex $v_iv_i \cdots v_i$ of $\Gamma^{n-1}(K_k^\circ)$ is placed as the first in the $k^{n-1}$ vertices of the form $v_iw_1w_2 \cdots w_{n-1}$, $w_j \in V(K_k^\circ)$, $1 \leq j < n$. Besides, $S^{++}(n,k)$ can be obtained from $S(n,k)$ and $S(n-1,k)$ by joining their extreme vertices. From these observations, we can decrease the upper bound in Theorem 1 by one for $S^{++}(n,k)$ if $k$ is even and $k \geq 4$. Since $S^+(n,k)$ can be obtained from $S^{++}(n,k)$ by contracting one copy of $S(n-1,k)$ into a single vertex, we have the similar upper bound for $S^+(n,k)$.

**Theorem 4** *Let* $n \geq 1$. *It holds that* $\mathsf{pn}(S^+(n,3)) = \mathsf{pn}(S^{++}(n,3)) = 2$, $\left\lceil \frac{k}{2} \right\rceil \leq \mathsf{pn}(S^+(n,k)) \leq 2\left\lceil \frac{k}{2} \right\rceil$ *for* $k \geq 4$, *and* $\left\lceil \frac{k}{2} \right\rceil \leq \mathsf{pn}(S^{++}(n,k)) \leq 2\left\lceil \frac{k}{2} \right\rceil$ *for* $k \geq 4$.

## Acknowledgments

The author is grateful to the reviewers for their helpful comments. One of them kindly suggests using the concept of a barycentric subdivision of a graph.

## References

[1] F. Bernhart and P.C. Kainen, The book thickness of a graph, *J. Combin. Theory* Ser. B 27 (1979) 320–331.

[2] G-H. Chen and D-R. Duh, Topological properties, communication, and computation on WK-recursive networks, *Networks* 24 (1994) 303–317.

[3] F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg, Embedding graphs in books: a layout problem with application to VLSI design, *SIAM J. Algebraic Discrete Methods* 8 (1987) 33–58.

[4] T. Hasunuma, Improved book-embeddings of incomplete hypercubes, *Discrete Applied Math.* 157 (2009) 1423–1431.

[5] T. Hasunuma, Structural properties of iterated subdivided-line graphs, in preparation.

[6] A.M. Hinz and P. Parisse, Coloring Hanoi and Sierpiński graphs, *Discrete Math.* 312 (2012) 1521–1535.

[7] S. Klavžar and U. Milutinović, Graphs $S(n,k)$ and a variant of the Tower of Hanoi problem, *Czechoslovak Math. J.* 47 (122) (1997) 95–104.

[8] S. Klavžar and B. Mohar, Crossing numbers of Sierpiński-like graphs, *J. Graph Theory* 50 (2005) 186–198.

[9] A.L. Rosenberg, The Diogenes approach to testable fault-tolerant arrays of processors, *IEEE Trans. Comput.* C-32 (1983) 902–910.

[10] G.D. Vecchia and C. Sanges, A recursively scalable network VLSI implementation, *Future Generat. Comput. Syst.* 4 (1988) 235–243.

[11] D.R. Wood, Bounded degree book embeddings and three-dimensional orthogonal graph drawing, Proc. GD2001, LNCS vol. 2265, pp.312–327, 2002, Springer.

# Area Requirement of Graph Drawings with Few Crossings per Edge

Emilio Di Giacomo* Walter Didimo* Giuseppe Liotta* Fabrizio Montecchiani*

**Abstract**

In this paper we study how to compute compact straight-line drawings of planar graphs with a limited number of crossings per edge. We prove that every outerplanar graph can be drawn in $O(n \log n)$ area using a sub-linear number of crossings per edge, and that for any given number $\varepsilon > 0$, every outerplanar graph admits an $O(n^{1+\varepsilon})$ area drawing with $O(n^{1-\varepsilon})$ crossing per edge. The drawing algorithms run in linear time and can be also generalized to another meaningful sub-family of series-parallel graphs with bounded vertex-degree.

## 1 Introduction

Many papers in the literature are devoted to the study of area requirement of planar straight-line drawings of graphs. In 1990, de Fraysseix *et al.* [2] and Schnyder [14], independently discovered two alternative techniques for computing a planar straight-line drawing of an $n$-vertex planar graph in $O(n^2)$ area. In [2] it is also proved that this bound is worst-case optimal for the family of planar graphs, as there are infinitely many planar graphs that require quadratic area to be drawn in the plane without edge crossings. Since then, several attempts have been done to prove the existence of straight-line planar drawings with $o(n^2)$ area for specific sub-families of planar graphs. Shiloach [15] and Crescenzi *et al.* [1] established $O(n \log n)$ area bounds for $n$-vertex trees. Garg and Rusu [11] proved that any $n$-vertex tree with vertex-degree $O(\sqrt{n})$ admits a planar straight-line drawing in $O(n)$ area. Other than trees, two meaningful sub-families of planar graphs that have been widely studied in terms of area requirement are *outerplanar graphs* and *series-parallel graphs*. An outerplanar graph is a planar graph that admits a planar drawing such that all vertices are on the external face. Series-parallel graphs form a super-class of the outerplanar graphs; they are recursively obtained by applying two kinds of operations, called "series" and "parallel" compositions (see [5] for a formal definition). Frati and Di Battista [4] proved that every $n$-vertex outerplanar graph admits a planar straight-line drawing in $O(n^{1.48})$ area. Later on, Frati [10] proved that every $n$-vertex outerplanar graph with vertex-degree at most $d$ has a planar straight-line drawing in $O(dn \log n)$; this second bound improves the previous one only if $d = O(n^{0.48}/\log n)$. Despite these upper bounds, it is still unknown whether an outerplanar graph can be always drawn in linear area. Concerning the area requirement of planar straight-line drawings of $n$-vertex series-parallel graphs, only a super-linear lower bound is known, namely $\Omega(n2^{\sqrt{\log n}})$ [9] (note that the function $f(n) = n2^{\sqrt{\log n}}$ is such that $f(n) = \omega(n \log n)$ and $f(n) = o(n^2)$). A natural question that arises from the results mentioned above is whether allowing some edge crossings may help to reduce the area of a drawing of a planar graph. However, so far, only few papers have been devoted to the study of non-planar drawings of planar graphs with the aim of obtaining compact drawings. Namely, Wood [16] showed that, for any fixed positive integer $k > 0$, all $k$-colorable graphs have a straight-line drawing in linear area; this implies that planar graphs always admit linear-area straight-line drawings with crossing edges. However, the technique by Wood can give rise to edges that contain a linear number of crossings. More recently, algorithms for computing linear-area drawings of bounded treewidth graphs without $h$ mutually crossing edges (where $h$ is a given positive integer) have been described in [6], but these drawings may still contain edges with a linear number of crossings.

In this paper, we study compact straight-line drawings of planar graphs with "few" crossings per edge, proving that drawing area bounds better than $O(n^{1.48})$ and $O(dn \log n)$ are achievable for all outerplanar graphs and for another sub-family of series-parallel graphs with specific properties, if we allow a sub-linear number of crossings per edge. More precisely, we prove that every $n$-vertex outerplanar graph admits a straight-line drawing with $O(\frac{n}{\log n})$ crossings per edge in $O(n \log n)$ area. Also, we prove that for any given $\varepsilon > 0$, every $n$-vertex outerplanar graph admits a straight-line drawing with $O(n^{1-\varepsilon})$ crossings per edge in $O(n^{1+\varepsilon})$ area, which gives a clear trade-off scheme between area and edge crossings. Both these results are based on a more general drawing algorithm, which runs in linear time and which can be applied to other sub-families of planar graphs that admit a "level" drawing with specific properties. In particular, the drawing area bounds obtained for the outerplanar graphs apply for another meaning-

*Dipartimento di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia, Italy, {digiacomo,didimo,liotta,montecchiani}@diei.unipg.it

ful sub-family of series-parallel graphs with bounded degree, known as *flat series-parallel graphs* [5]. It is worth recalling that drawings of graphs with at most $k$ crossings per edge are usually called *$k$-planar drawings*, and their properties have been widely studied in the literature in terms of edge density (see, e.g., [7, 12, 13]). We finally recall that looking for compact drawings with non-constant number of crossings per edge is also motivated by the fact that $k$-planar straight-line drawings of series-parallel graphs may require $\Omega(n2^{\sqrt{\log n}})$ area, for any fixed integer constant $k > 0$, as proved in [6].

The basic ingredients and an outline of our drawing strategy are described in Section 2. The area requirement of planar drawings of outerplanar graphs with a sub-linear number of crossings per edge is studied in Section 3, and the extension of the results to flat series-parallel graphs is discussed in Section 4. Conclusions and open problems are in Section 5.

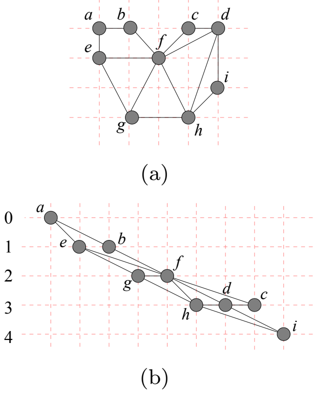


Figure 1: (a) A planar straight-line grid drawing Γ of an outerplanar graph $G$, where all vertices are on the external face. The area of Γ is $4 \times 5 = 20$. (b) A proper level drawing of $G$ with respect to the vertex leveling $L(G) = \{0, 1, 2, 3, 4\}$ such that: $\ell(0) = \{a\}$, $\ell(1) = \{e, b\}$, $\ell(2) = \{g, f\}$, $\ell(3) = \{h, d, c\}$, $\ell(4) = \{i\}$; in the drawing all levels are not folded.

## 2 The General Drawing Strategy: Leveling and Folding

We assume familiarity with graph drawing terminology [3]. A *leveling* of a graph $G$ is a labeling of all vertices of $G$ with non-negative integer numbers $L = \{0, \dots, r\}$ such that for each $j \in L$ there is at least one vertex labeled $j$. Each $j \in L$ is called the *level* $j$ of the leveling, and all vertices with label $j$ are the *vertices of level* $j$. A leveling of $G$ is denoted as $L(G)$ and the set of vertices of level $j$ is denoted by $\ell(j)$. Let Γ be a straight-line grid drawing of a graph $G$ with a leveling $L(G)$. For each vertex $v$, we denote by $x(v)$ and $y(v)$ the $x$-coordinate and the $y$-coordinate of $v$, respectively. Drawing Γ is called a *proper level drawing* of $G$ with respect to the leveling $L(G)$, if it has the following properties (see Figure 1(b) for an example of a proper level drawing): **L1:** All vertices of the same level $j$ have the same $y$-coordinate, which we simply denote by $y(j)$. **L2:** For each edge $(u, v)$, $|y(u) - y(v)| \leq 1$. **L3:** For any two levels $j < l$, all vertices of level $l$ are to the right of all vertices of level $j$. **L4:** If $u$ and $v$ are any two vertices of Γ that are consecutive in the left-to-right order, then $x(v) - x(u) = 1$ (i.e., the vertices occupy $n$ consecutive $x$-coordinates in the integer grid). Let $l$ be a level of a leveling $L(G) = \{0, \dots, r\}$ such that $0 < l < r$, and let Γ be a proper level drawing of $L(G)$. We say that $l$ is *folded* in Γ if $y(l-1) = y(l+1)$. The first and the last level of $L(G)$ are never folded in Γ.

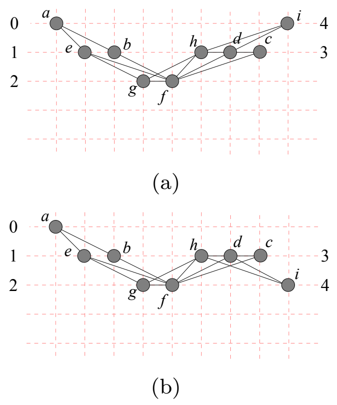


Figure 2: Two different proper level drawings of $G$ with respect to same vertex leveling as in Figure 1(b): (a) level 2 is folded while the others are not folded; (b) both levels 2 and 3 are folded, while the others are not folded.

Our general drawing strategy for a graph $G$ works into two main phases: Leveling: Find a leveling $L(G)$ and a suitable proper level drawing Γ of $G$ with respect to $L(G)$, without folded levels. Folding: Construct from Γ another proper level drawing Γ′, having a suitable subset of folded levels. Phase Folding constructs Γ′ from Γ by applying a sequence of *folding operations*. Each time a folding operation is applied, a new proper level drawing is obtained from the previous one, which contains one more folded level. More precisely, suppose that $\Gamma_0$ is a proper level drawing of a graph $G$ with respect to some leveling $L(G)$, and let $j$ be a level of $L(G)$ that is not folded in $\Gamma_0$. Applying a folding operation on level $j$ to the drawing $\Gamma_0$, we transform $\Gamma_0$ into a new proper level drawing $\Gamma_1$ in such a way that: ($i$) the left-to-right order of the vertices in $\Gamma_0$ and $\Gamma_1$ is the same; ($ii$) every level $l \neq j$ that is folded (resp. not folded) in $\Gamma_0$ is also folded (resp. not folded) in $\Gamma_1$; ($iii$) level $j$ is folded in $\Gamma_1$. Hence, $\Gamma_1$ has one folded level more than $\Gamma_0$, i.e., the level $j$. Notice that, by definition, the ordering of the folding operations in a sequence that transforms a drawing $\Gamma$ into another drawing $\Gamma'$ does not matter. We now prove the following lemma, one of the main ingredients of our final results.

**Lemma 1** *Let $G$ be an $n$-vertex graph with a given leveling $L(G)$ and let $d \geq 0$ be an integer constant. Let $\Gamma$ be a proper level drawing of $G$ with respect to $L(G)$, such that $\Gamma$ is $d$-planar and has no folded levels. Let $k(n) : \mathbb{N} \rightarrow \mathbb{N}$ be any function such that $k(n) \in O(n)$ and $k(n) > d$ for every $n \geq n_0$, for some $n_0 \geq 0$. Then, there exists a proper level drawing $\Gamma'$ of $G$ with respect to $L(G)$, such that: ($i$) $\Gamma'$ has $O(k(n))$ crossings per edge; ($ii$) $\Gamma'$ has area $O(\frac{n^2}{k(n)})$. Such a drawing can be computed in $O(n)$ time.*

**Sketch of Proof.** et $L = \{0, \dots, r\}$ be the levels of $L(G)$. For each level $0 < j < r$, denote by $M_j$ the maximum between the number of edges that connect vertices of level $j$ to vertices of level $j-1$ and the number of edges that connect vertices of level $j$ to vertices of level $j+1$. Drawing $\Gamma'$ is obtained from $\Gamma$ through a Folding phase that applies a suitable sequence of folding operations. Namely, starting from $\Gamma$ it suffices to iteratively apply a folding operation on every level $0 < j < r$ such that $M_j \leq \frac{k(n)-d}{2}$. If $M_j \leq \frac{k(n)-d}{2}$, we say that $j$ satisfies the *folding condition* (only when $k(n) > d$ for every $n \geq n_0$). First of all, observe that the folding operation does not change the number and the type of crossings in the subgraph induced by the vertices of two consecutive levels. Indeed, these crossings depend only on the left-to-right ordering of the vertices on the two levels, and this ordering is preserved by the folding operation. Hence, in $\Gamma'$ the subgraph induced by any two consecutive levels is still $d$-planar. We prove that in $\Gamma'$ each edge is crossed at most $k(n)$ times. Consider an edge $e = (u, v)$. By Property **L2** either $u, v$ belong to the same level (in which case edge $e$ does not cross edges in $\Gamma'$, as otherwise it would overlap some vertices and $\Gamma$ would not be a valid drawing) or they belong to consecutive levels. Assume that $u$ is a vertex of level $j$ and that $v$ is a vertex of level $j+1$. We distinguish between two cases: `Case 1:` $j$ is not folded in $\Gamma'$. In this case two further sub-cases are possible: ($a$) If also $j+1$ is not folded, then $e$ can only be crossed by edges going from level $j$ to level $j+1$, hence it crosses at most $d < k(n)$ other edges by hypothesis. ($b$) If $j+1$ is folded, then $(u, v)$ can also cross edges going from level $j+1$ to level $j+2$; however, since $j+1$ satisfies the folding condition, the number of these edges is at most $\frac{k(n)-d}{2}$, and hence $(u, v)$ contains at most $\frac{k(n)-d}{2} + d = \frac{k(n)+d}{2} \leq k(n)$ crossings. `Case 2:` $j$ is folded in $\Gamma'$. Also in this case we have two further sub-cases: ($a$) If $j+1$ is not folded, then $e$ can only cross (at most $d$) edges going from level $j$ to level $j+1$ and edges going from level $j-1$ to level $j$, which are at most $\frac{k(n)-d}{2}$, because $j$ satisfies the folding condition. Hence $e$ contains at most $k(n)$ crossings. ($b$) If $j+1$ is folded, then $e$ can cross at most $d$ edges going from level $j$ to level $j+1$, at most $\frac{k(n)-d}{2}$ edges going from level $j-1$ to level $j$, and at most $\frac{k(n)-d}{2}$ edges going from level $j+1$ to level $j+2$. Hence, it contains at most $k(n)$ crossings. We now prove the bound on the area of $\Gamma'$. Let $A(n)$ and $A'(n)$ denote the areas of $\Gamma$ and $\Gamma'$, respectively and let $h(\Gamma)$ and $h(\Gamma')$ denote the height of $\Gamma$ and $\Gamma'$, respectively. By the properties of a proper level drawing, and since $\Gamma$ has no folded levels, we have $A(n) = (r+1)n$. In order to bound the area $A'(n)$ we observe that if $l$ $(0 < l < r)$ is a folded level of $\Gamma'$, then $y(l-1) = y(l+1)$, which means that at least two levels (i.e., $i-1$ and $i+1$) occupy the same horizontal line in $\Gamma'$, thus reducing by at least one the number of horizontal lines used in $\Gamma$. Hence, if $\Gamma'$ has $h^*$ folded levels, then $h(\Gamma') \leq h(\Gamma) - h^*$, i.e., the height of $\Gamma'$ is at most the number of levels that are not folded in $\Gamma'$. We have our first inequality: $A'(n) \leq (h^* + 2)n$. Let $m$ be the number of edges of $G$, and let $m^*$ be the number of edges of $G$ that are incident to the $h^*$ non-folded levels, distinct from 0 and $r$. By construction, for each of the $h^*$ levels there are more than $\frac{k(n)-d}{2}$ edges incident to vertices of this level. Also, since each edge is shared by at most two of the $h^*$ levels, we have a second inequality: $m^* > \frac{h^*}{2}\frac{k(n)-d}{2} = \frac{h^*(k(n)-d)}{4}$. It is known [13] that, for a $d$-planar graph, $m \leq \sqrt{16.875\, d}\; n$, thus, $m^* \leq m \leq \sqrt{16.875\, d}\; n$. Hence, we have: $h^* < \frac{4\sqrt{16.875\, d}\; n}{k(n)-d}$. Plugging this third inequality into the first one, we get $A'(n) < \frac{4\sqrt{16.875\, d}\; n^2}{k(n)-d} + 2n$. Since $k(n) \in O(n)$ and $d$ is a constant, we have $A'(n) = O(\frac{n^2}{k(n)})$. □

## 3 Outerplanar graphs

Outerplanar graphs are drawn using the general strategy described in Section 2. We prove the following.

**Theorem 2** *Let $G$ be an outerplanar graph with $n$ vertices: ($i$) $G$ admits a straight-line grid drawing with $O(\frac{n}{\log n})$ crossings per edge in $O(n \log n)$ area; ($ii$) For any $\varepsilon > 0$, $G$ admits a straight-line grid drawing with $O(n^{1-\varepsilon})$ crossings per edge in $O(n^{1+\varepsilon})$ area; ($iii$) These drawings can be computed in linear time.*

**Sketch of Proof.** According to the general strategy of Section 2, a leveling $L(G)$ and a proper level drawing $\Gamma$ of $G$ with respect to $L(G)$ are first computed. From a result by Felsner *et al.* [8], it is known that a level drawing $\Gamma$ can be constructed in linear time such that $d = 0$ (the details of this construction are omitted for space reasons).

Assume now that $k(n) : \mathbb{N} \to \mathbb{N}$ is a function such that $k(n) = \Theta(\frac{n}{\log n})$. According to Lemma 1, we can compute in linear time from $\Gamma$ a $k(n)$-planar straight-line grid drawing $\Gamma'$ whose area is $O(n \log n)$. Analogously, given any function $k(n) : \mathbb{N} \to \mathbb{N}$ such that $k(n) = \Theta(n^{1-\varepsilon})$, by Lemma 1 we can transform in linear time $\Gamma$ into a $k(n)$-planar straight-line grid drawing $\Gamma'$ with $O(n^{1+\varepsilon})$ area. □

## 4 Flat Series-parallel graphs

It is known that outerplanar graphs are series-parallel graphs. Another meaningful sub-family of series-parallel graphs, is the class of the so-called *flat series-parallel graphs*. They are those series-parallel graphs without two parallel components that share a pole and that are not in a series composition. A formal definition of flat series-parallel graphs is provided in [5]. We prove the following.

**Theorem 3** *Let $d > 0$ be a given integer constant and let $G$ be a flat series-parallel graph with $n$ vertices and vertex-degree at most $d$: ($i$) $G$ admits a straight-line grid drawing with $O(\frac{n}{\log n})$ crossings per edge in $O(n \log n)$ area; ($ii$) For any $\varepsilon > 0$, $G$ admits a straight-line grid drawing with $O(n^{1-\varepsilon})$ crossings per edge in $O(n^{1+\varepsilon})$ area; ($iii$) These drawings can be computed in linear time.*

**Sketch of Proof.** As for the outerplanar graphs, we first define a leveling $L(G)$ and a proper level drawing $\Gamma$ of $G$ with respect to $L(G)$. In [6] it is shown how to compute $\Gamma$ of $G$ so that, if the degree of each vertex is at most $d$, an edge can cross at most $d$ other edges. The details of this technique, based on a BFS visit of the minimal decomposition tree of $G$, are omitted for space reasons.

Assume now that $k(n) : \mathbb{N} \to \mathbb{N}$ is a function such that $k(n) = \Theta(\frac{n}{\log n})$. According to Lemma 1, we can compute in linear time from $\Gamma$ a $k(n)$-planar straight-line grid drawing $\Gamma'$ whose area is $O(n \log n)$. Analogously, given $k(n) : \mathbb{N} \to \mathbb{N}$ such that $k(n) = \Theta(n^{1-\varepsilon})$, by Lemma 1 we can transform in linear time $\Gamma$ into a $k(n)$-planar straight-line grid drawing $\Gamma'$ with $O(n^{1+\varepsilon})$ area. □

## 5 Conclusions and Open Problems

Studying the trade-off between crossings and area requirement is a new promising research direction, which poses many interesting and challenging problems. The following open questions naturally arise from our results: Theorem 2 shows that straight-line drawings of outerplanar graphs in $O(n \log n)$ area always exist, which contain $O(\frac{n}{\log n})$ crossings per edge. Is it possible to achieve the same area bound using a logarithmic or even a constant number of crossings per edge? Theorem 3 proves the same bounds as Theorem 2 for a meaningful sub-family of series-parallel graphs with bounded degree. Does every series-parallel graph always admit a straight-line drawing with a sub-linear number of crossings per edge in $O(n \log n)$ area? We recall there are infinitely many series-parallel graphs that require $\omega(n \log n)$ area if drawn with constant number of crossings per edge, even using bent edges [6].

## References

[1] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *CGTA*, 2:187–200, 1992.

[2] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[3] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

[4] G. Di Battista and F. Frati. Small area drawings of outerplanar graphs. *Algorithmica*, 54:25–53, 2009.

[5] E. Di Giacomo. Drawing series-parallel graphs on restricted integer 3D grids. In *GD*, volume 2912 of *LNCS*, pages 238–246. Springer, 2004.

[6] E. Di Giacomo, W. Didimo, G. Liotta, and F. Montecchiani. $h$-quasi planar drawings of bounded treewidth graphs in linear area. In *WG*, volume 7551 of *LNCS*, pages 91–102. Springer, 2012.

[7] P. Eades and G. Liotta. Right angle crossing graphs and 1-planarity. In *GD*, volume 7034 of *LNCS*, pages 148–153. Springer, 2011.

[8] S. Felsner, G. Liotta, and S. K. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. *JGAA*, 7(4):363–398, 2003.

[9] F. Frati. Lower bounds on the area requirements of series-parallel graphs. *DMTCS*, 12(5):139–174, 2010.

[10] F. Frati. Straight-line drawings of outerplanar graphs in $O(dn \log n)$ area. *CGTA*, 45(9):524–533, 2012.

[11] A. Garg and A. Rusu. Straight-line drawings of general trees with linear area and arbitrary aspect ratio. In *ICCSA (3)*, volume 2669 of *LNCS*, pages 876–885. Springer, 2003.

[12] S.-H. Hong, P. Eades, G. Liotta, and S.-H. Poon. Fáry's theorem for 1-planar graphs. In *COCOON*, volume 7434 of *LNCS*, pages 335–346. Springer, 2012.

[13] J. Pach and G. Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.

[14] W. Schnyder. Embedding planar graphs on the grid. In *SODA*, pages 138–148. SIAM, 1990.

[15] Y. Shiloach. *Arrangements of Planar Graphs on the Planar Lattice.* PhD thesis, Weizmann Institute of Science, 1976.

[16] D. R. Wood. Grid drawings of $k$-colourable graphs. *CGTA*, 30(1):25–28, 2005.

# Distributed universal reconfiguration of 2D lattice-based modular robots

Ferran Hurtado[*] Enrique Molina[*] Suneeta Ramaswami[†] Vera Sacristán[*]

**Abstract**

We prove universal reconfiguration (i.e., reconfiguration between any two robotic systems with the same number of modules) of 2-dimensional lattice-based modular robots by means of a distributed algorithm. To the best of our knowledge, this is the first known reconfiguration algorithm that applies in a general setting to a wide variety of particular modular robotic systems, and holds for both square and hexagonal lattice-based 2-dimensional systems. All modules apply the same set of local rules (in a manner similar to cellular automata), and move relative to each other. Reconfiguration is carried out while keeping the robot connected at all times. The total number of time steps, moves and communication required for the reconfiguration is linear in the number of modules.

## 1 Introduction

### 1.1 Goal

We solve the following problem for 2-dimensional lattice-based modular robotic systems: Given two connected configurations with the same number of modules, reconfigure one into the other by means of a distributed algorithm. As far as we know, this is the first general reconfiguration algorithm encompassing both square and hexagonal regular lattices, and using a general framework that does not exploit specific characteristics of any particular robotic system. A large set of robotic prototypes fit this framework.

In our framework, a robot is a connected configuration of homogeneous modules that are located in a 2-dimensional lattice. Each module can attach to and detach from a neighboring module, and can change its position to a neighboring empty grid position in the lattice by attaching to a neighboring module and moving with respect to it. Each module has constant size memory, can perform constant size computations, and can send or receive constant size messages to or from its neighboring modules. One designated module needs linear memory to store the information of the goal shape and to perform computations required for the reconfiguration algorithm.

Within this framework, our algorithm is distributed and local. It consists of a set of rules, each one having a priority, a precondition, and an action or postcondition. Rules are identical for all modules, and are simultaneously executed by all of them. The term "local" here means that each module communicates with modules lying within a small neighborhood in order to execute the algorithm. In the procedure we propose, all modules know when they have reached their final destination.

### 1.2 Related work

Our approach builds on the seminal work of Beni [2], who proposed the conceptual model of cellular robotic systems, inspired by cellular automata. Since then, several authors have developed distributed algorithms for reconfiguring specific square lattice-based modular robot designs and shapes [8], as well as generic strategies for locomotion, reconfiguration and self-repairing for particular shapes [4, 10]. Simultaneously, locomotion and reconfiguration have been proved for some class of shapes within the hexagonal setting [6, 7, 16, 15, 9, 1], as well as for 3-dimensional lattices [3]. Local rules have also been used in the framework of a general metamodules' theory [5]. Recently, specific sets of rules have been proposed to produce reconfigurations between particular shapes of M-TRAN which are lattice-based [11]. To the best of our knowledge, this last work presents the first execution of a distributed local rules strategy on real robot units, hence proving its realizability beyond experimental simulation.

## 2 The model

In the square lattice setting, a module is any robotic unit located in a 2-dimensional square grid. We represent modules by squares occupying one grid cell, although their actual shape need not be a square. A module can independently attach to and detach from each of its 4 direct grid neighboring modules, if present. A robot is a connected set of identical modules. By "connected" we mean that the adjacency

[*]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain. {ferran.hurtado,vera.sacristan}@upc.edu, Enrique-Molina@hotmail.com. F.H. and V.S. were partially supported by projects MTM2012-30951, MTM2009-07242, Gen. Cat. DGR 2009SGR1040, and ESF EUROCORES programme EuroGIGA, CRP ComPoSe: MICINN Project EUI-EURC-2011-4306, for Spain.

[†]Department of Computer Science, Rutgers University, Camden, U.S.A. rsuneeta@camden.rutgers.edu. Partially supported by NSF grant CCF-0830589.

graph of the robot configuration (a node in the center of each module and a straight line edge for each attachment among modules) is connected.

Modules cannot move on their own, but they can move relative to each other. To be more precise, a module may perform four relative motions, illustrated in Figure 1, where the dark colored module is performing the move. The first two moves are of the *change position* type: a module performing *slide* or *convex transition* translates itself from its current lattice position to a neighboring one. The last two moves are of the *change attachments* type: a module performing *concave transition* or *opposite transition* changes its attachment from one neighbor to another without modifying its lattice position.

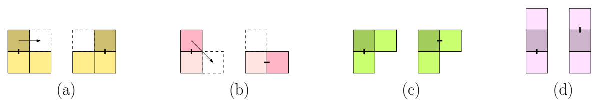


Figure 1: Change position moves: (a) slide (b) convex transition. Change attachments moves: (c) concave transition (d) opposite transition. All moves may apply in any of the 4 directions (N, S, E, W) relative to the moving module.

In our framework, the modules of the robot are indistinguishable, and each module is given and applies the same set of rules. In order to do so, we assume each module has a (simple) processor and some (small) memory, knows its own orientation (N, S, E, W) and state (active or passive), can detect whether it is attached to a neighbor, can send and receive (short) messages to and from neighbors, and is able to perform (elementary) operations with a few counters and text strings. For our reconfiguration algorithm, only one module needs to store the final configuration, which is a linear amount of information. This module, called the leader, can be either determined in advance or chosen by the set of modules [12].

As stated above, all modules run the same predefined set of rules. Each rule has the following structure: a priority, a precondition, and an action or postcondition. Priorities, represented as small integers, are used by the module to decide which of possibly several rules that apply to its situation is executed. A precondition is any constant size boolean combination of the following: compare priorities, check neighboring empty/filled positions, check own connections, match states/text or counters/integers, and compare calculation results with counters, messages and integers. A postcondition can be any *and* combination of the following: change position (slide, convex transition), change attachments (concave transition, opposite transition), modify state, compute and update counters, and send messages.

In our model, changing position only requires the goal lattice position to be free. This assumption could be a potential limitation because the atomic robot units of several current prototypes need some extra empty space to produce slide and convex transition. Nevertheless, by appropriately grouping atomic robot units into meta-modules, we have been able to ensure that our moves can be safely made without extra free space requirements in three general models of reconfigurable robots: the expand/contract model, the sliding model, and the rotating model.

## 3 Overview of Reconfiguration Strategy

The solution we present is distributed because each module acts on its own without the need of a central controller, other than to get the reconfiguration process started. Our solution is parallel as all modules act in parallel. Our solution is local because each module only needs to communicate with modules within a small neighborhood when checking rule preconditions. In this context, the neighborhood of a module consists of all modules lying in grid positions within the second annulus around it. Finally, we should mention that our reconfiguration strategy (and our simulator) is intended to run in a synchronized framework. An asynchronous version can be obtained by means of a shaking hands strategy, at a cost of increased communication among the modules.

The overall strategy behind our algorithm is to move modules along the boundary of the robot to reconfigure in two stages. We first reconfigure the robot from its initial shape into a canonical shape (the strip configuration) and then from the canonical to the final shape. The modules do not need to know the robot's complete initial shape. However, the goal shape needs to be known at least by the leader. In particular, our solution to reconfigure from the canonical to goal shape requires the leader to assign a final destination location for each module in the canonical configuration. Our solution is based on the following general operating principles:

1. A particular spanning tree of the robot's adjacency graph, called the *scan tree*, is built so that all leaves of the tree lie on the boundary of the robot (see Figure 2). At the beginning, all modules are considered to be static. At any given instant, only leaf modules can start moving, i.e., go from static to active. Once a module is active, its node is cut from the spanning tree.

2. The movement of the active modules along the boundary of the static robot always follows the right hand rule (turn right along the robot boundary) when reconfiguring from the initial to canonical shape, and the left hand rule when reconfiguring from the canonical to goal shape.

3. Moving modules are not allowed to climb (move relative to) other moving modules. This is a reasonable assumption in order to avoid unbounded acceleration and unpredictable collisions.

4. Every module is assigned a number when constructing the above stated spanning tree. Generally speaking, this number corresponds to the DFS (depth first search) order numbering of the nodes of the scan tree of the initial shape for the initial to canonical reconfiguration, or the goal shape for the canonical to goal reconfiguration. This number is used to guide the moves of the modules and also to prove the correctness of our solution.

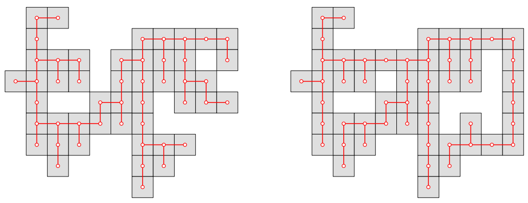


Figure 2: Left: the scan tree of a configuration without holes. Right: the scan tree of a configuration with holes.

Rules for advancing modules during reconfiguration must take care of various types of conflicts:

**Activation conflicts** occur when an active module tries to move to a position where it would attach to a static leaf which simultaneously becomes active. In this case, priority needs to be given either to the activation of the leaf or to the moving module. Any of the two choices is appropriate, as long as it stays consistent during the reconfiguration.

**Collision conflicts** occur when two active modules intend to move to the same lattice position. Priority is given to the module with lower DFS order number.

**Obstruction conflicts** occur when an active module would like to move into a lattice position which is already occupied by an active module. In this case, deadlocked situations could be created if the boundary of the static shape forms bottlenecks. See Figure 3 for an illustration. We avoid deadlocks by means of specific "jumping rules" that have higher priority than the regular advancing ones. Jumping at a bottleneck allows a module to advance more than one position along the boundary of the static shape in one step to avoid entrance into the "cul-de-sac" region. Deciding which modules should jump and at which bottlenecks is crucial in our strategy. These decisions are made by comparing the DFS number of the static modules that form the bottleneck. The active module uses the DFS numbers to determine whether it is about to enter or exit the corresponding cul-de-sac. The reconfiguration from the strip to the goal shape essentially reverses the forward (initial shape to strip) reconfiguration procedure. Modules march from the rightmost end of the strip following the left hand rule and the goal shape is constructed in a clockwise depth-first manner. However, two additional issues need to be addressed in the reverse reconfiguration. The main difference between the forward and backward procedures is that modules in the strip must be sent to their final goal destination in the order given by the depth-first traversal of the final shape. Therefore, the jumping rules need to be modified to make sure that no jump changes the order of the active modules when they march along the boundaries of the static structure. In addition, it is also necessary to ensure that for robot shapes with holes, no active module gets trapped within the wrong hole or outside its destination hole because of premature closing of the hole during the reconfiguration procedure.

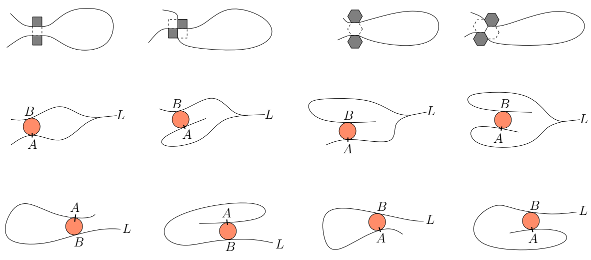


Figure 3: *First row:* The two possible bottleneck types in square lattices (left) and the hexagonal lattices (right). The continuous line schematizes the boundary of the robot shape. *Second and third rows:* The active module $x$ (depicted as a red/dark disc) is attached to the static module $A$ and decides whether to jump, i.e., attach to $B$ and detach from $A$. The decision is based on whether $B$ belongs to a branch in the tree previous, in DFS order, to that of $A$ (second row) or the same branch as that of $A$ (third row). This allows $x$ to determine if it is about to enter or exit the cul-de-sac and consequently, whether or not to jump at the bottleneck. $L$ is the position of the root of the tree, i.e., the leader. The continuous lines schematize the branches of the tree involved in the bottleneck $A$, $B$.

## 4 Main result

We state, without proof, some lemmas used to show the final result stated in Theorem 4.

**Lemma 1** *At all times along the reconfiguration, the static tree, although pruned, stays a scan-tree, and the numbering of the modules along its external boundary increases counterclockwise from the leader up to the first leaf.*

A deadlock loop is a sequence of active modules $a_1, \dots, a_k$ such that each $a_i$ intends to occupy the lattice position of module $a_{i-1}$ (indices are mod $k$).

**Lemma 2** *The algorithm outlined in Section 3 cannot create deadlock loops.*

The above lemma, together with the invariant established in Lemma 1, is used to prove the following:

**Lemma 3** *The algorithm outlined in Section 3 makes every module move past the leader when reconfiguring to the strip.*

**Theorem 4** *Given two robotic systems with the same number of modules, the algorithm outlined in Section 3 reconfigures one shape into the other.*

If executed synchronously, any reconfiguration of a robotic system of $n$ modules is done in $O(n)$ time steps with $O(n)$ basic moves per module, using $O(1)$ force per module, $O(1)$ size memory and computation per module (except for one module, which needs $O(n)$ size memory to store the information of the goal shape), and $O(n)$ communication per module.

## 5 Simulations

We have implemented our rules [13] for square lattices in a synchronized simulator [14], and have applied them to a large set of reconfigurations (Figure 4 shows a screen shot). We are currently working on the hexagonal lattice simulations.

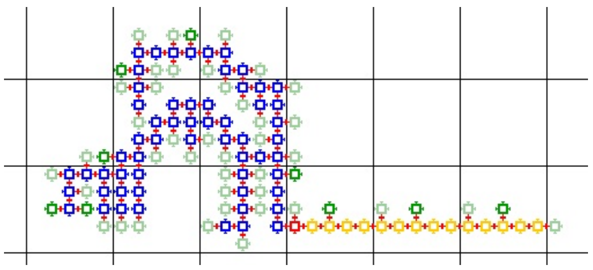


Figure 4: A screen shot of the simulation. Dark blue modules are static, green modules are active, and the yellow horizontal strip on the right is being formed.

## References

[1] J. Bateau, A. Clark, K. McEachern, E. Schutze, and J. Walter. Increasing the efficiency of distributed goal-filling algorithms for self-reconfigurable hexagonal metamorphic robots. In *Proc. of the International Conference on Parallel and Distributed Techniques and Applications*, 2012.

[2] G. Beni. The concept of cellular robotic system. In *Proc. of the IEEE International Symposium on Intelligent Control*, pages 57–62, 1988.

[3] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1734–1741, 2000.

[4] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *Int. J. Robot. Res.*, 23:919–937, 2004.

[5] D. J. Dewey, M. P. Ashley-Rollman, M. De Rosa, S. C. Goldstein, T. C. Mowry, S. S. Srinivasa, P. Pillai, and J. Campbell. Generalizing metamodules to simplify planning in modular robotic systems. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1338–1345, 2008.

[6] A. Dumitrescu, I. Suzuki, and M. Yamashita. Formations for fast locomotion of metamorphic robotic systems. *Int. J. Robot. Res.*, 23(6):583–593, 2004.

[7] A. Dumitrescu, I. Suzuki, and M. Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Trans. Robot. Autom.*, 20(3), 2004.

[8] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2858–2863, 1998.

[9] P. Ivanov and J. Walter. Layering algorithm for collision-free traversal using hexagonal self-reconfigurable metamorphic robots. In *Proc. of IEEE/RSJ International Conference on Robots and Systems (IROS)*, pages 521–528, 2010.

[10] K. Kotay and D. Rus. Generic distributed assembly and repair algorithms for self-reconfiguring robots. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2362–2369, 2004.

[11] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *Int. J. Robot. Res.*, 27:373–386, 2008.

[12] C. Nichitiu, J. Mazoyer, and E. Rémila. Algorithms for leader election by cellular automata. *J. Algorithms*, 41(2):302–329, 2001.

[13] O. Rodríguez. Simulating distributed action of modular robots. Degree thesis, Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain, 2013 (in Catalan).

[14] R. Wallner. A system of autonomously self-reconfigurable agents. Diploma thesis, Institute for Software Technology, Graz University of Technology, Graz, Austria, 2009.

[15] J. E. Walter, J. L. Welch, and N. M. Amato. Concurrent metamorphosis of hexagonal robot chains into simple connected configurations. *IEEE Trans. Robot. Autom.*, 18(6):945–956, 2002.

[16] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17:171–189, 2004.

# The Number of Different Unfoldings of Polyhedra

Takashi Horiyama[*] Wataru Shoji[†]

## 1 Introduction

An unfolding (also called an edge unfolding, a net or a development) of a polyhedron is a simple polygon obtained by cutting along the edges of the polyhedron and unfolding it into a plane. The cut edges of an edge unfolding of a polyhedron form a spanning tree of the 1-skeleton (i.e., the graph formed by the vertices and the edges) of the polyhedron (See, e.g., [8, Lemma 22.1.1]). Since Kirchhoff's matrix-tree theorem gives the number of spanning trees for any graph, we can obtain the number of unfoldings for any polyhedron. For example, a cube has 384 unfoldings (i.e., 384 ways of cut edges).

Different cut edges, however, may have isomorphic unfoldings. In Figure 1, (a) and (b) have different cut edges (depicted in bold lines), while their unfoldings have the same shape depicted in (c). In actual, 24 unfoldings of a cube are isomorphic to Figure 1(c). Later in this paper, we consider two cases for counting the number of unfoldings. (1) The number of *labeled unfoldings:* edges have labels and we distinguish unfoldings according to their cut edges. (2) The number of *nonisomorphic unfoldings:* we identify isomorphic unfoldings even if they have different edge labels. The 384 labeled unfoldings of a cube are classified into 11 essentially different (i.e., nonisomorphic) unfoldings.

As mentioned later in related work, the number of unfoldings are of great interest for their wide area of applications. As for the counting for concrete polyhedra, most of the results are on the numbers of labeled unfoldings, since they are obtained by the matrix-tree theorem. On the other hand, few are on the numbers of nonisomorphic unfoldings (e.g., those of Platonic solids [3, 9, 11], and Archimedean $n$-gonal prism with $n = 3$ to 14 [14]), while they also have rich store of mathematical knowledge.

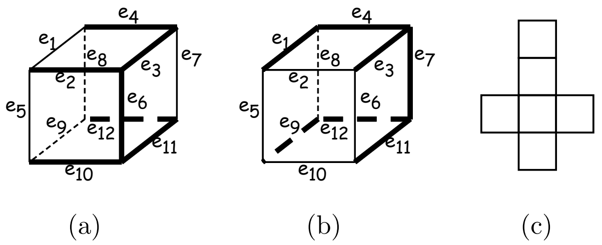


Figure 1: Different cut edges (a) and (b) have isomorphic unfoldings.

[*]Information Technology Center, Saitama University, Japan, horiyama@al.ics.saitama-u.ac.jp

[†]Graduate School of Science and Engineering, Saitama University, Japan, shoji@al.ics.saitama-u.ac.jp

**Our Contribution.** In this paper, we address how to count the number of nonisomorphic unfoldings for any polyhedron. The naive way for counting nonisomorphic unfoldings is to enumerate all labeled unfoldings and to omit isomorphic unfoldings. Unfortunately, a dodecahedron and an icosahedron have 5,184,000 labeled unfoldings, respectively, and the test for the isomorphism is tough. (The test may be required $\binom{5{,}184{,}000}{2}$ times.) We here note that unfoldings in this paper may have overlaps. (In [10], some overlapping unfoldings of some Archimedean solids are given.) Similar story happened on Platonic solids: When the numbers of thier nonisomorphic unfoldings were first obtained in about 40 years ago [3, 9, 11], we could not distinguish whether they are overlapping or not. In quite recent years, by enumerating all unfoldings, it is proved that any edge unfolding of Platonic solids is a flat nonoverlapping simple polygon [10, 13].

For counting the number of nonisomorphic unfoldings, we follow the basic idea in [9, 11], that is, to use Burnside's lemma [7]: given a polyhedron $P$, the number of nonisomorphic unfoldings is obtained by $u(\Gamma) = \frac{1}{|\mathrm{Aut}\,\Gamma|}\sum_{g\in \mathrm{Aut}\,\Gamma} |\{T \in \mathcal{T} \mid T = gT\}|$, where $\Gamma$ is the 1-skeleton of $P$, $u(\Gamma)$ is the number of nonisomorphic spanning trees of $\Gamma$, $\mathrm{Aut}\,\Gamma$ is the automorphism group of $\Gamma$, $g$ is a permutation in $\mathrm{Aut}\,\Gamma$, and $\mathcal{T}$ is the set of spanning trees of $\Gamma$. Although we can compute this equation by checking $T = gT$ (i.e., $T$ has the same structure with the permuted tree $gT$) for every $g \in \mathrm{Aut}\,\Gamma$ and $T \in \mathcal{T}$, it is impractical for large $\mathcal{T}$ (e.g., $|\mathcal{T}| =$ 21,789,262,703,685,125,511,464,767,107,171,876,864,000 for a truncated icosidodecahedron).

To overcome this situation, we follow the second idea in [4], that is, to use a quotient graph. A quotient graph $\mathcal{Q}(\Gamma, g)$ is intuitively obtained by iterative contraction of two vertices $u$ and $v$ in $\Gamma$ satisfying $u = g(v)$ (the edges are also contracted in the similar manner). By analyzing the structure of $\mathcal{Q}(\Gamma, g)$, we can obtain $|\{T \in \mathcal{T} \mid T = gT\}|$ without checking $T = gT$ for each $T \in \mathcal{T}$. In [4], they analyzed three cases which is necessary to obtain the number of nonisomorphic unfoldings of the five Platonic solids (in 3-dimensions) and the six regular convex polytopes in 4-dimensions. In this paper, we extend the technique so that we can apply it to any polyhedron.

Another contribution of this paper is the numbers of nonisomorphic unfoldings of all regular-faced convex polyhedra (i.e., Platonic solids, Archimedean solids, Johnson-Zalgaller solids, Archimedean prisms, and antiprisms). Furthermore, the numbers of nonisomorphic unfoldings of Catalan solids, bipyramids and trapezohedra are obtained, since they are the duals of Archimedean solids, prisms, antiprisms, and the dual of a solid has the same number of unfoldings with the original one.

For example, while a truncated icosahedron (and also a pentakis dodecahedron) has 375,291, 866,372,898,816,000 (approximately 375 exa) labeled unfoldings, it has 3,127,432,220,939,473,920 (approximately 3 exa) nonisomorphic unfoldings. A truncated icosidodecahedron (and also a disdyakis triacontahedron) has 21,789,262,703,685,125,511,464, 767,107,171,876,864,000 (approximately 21,789,262, 703,685,125 yotta) labeled unfolding, and has 181, 577,189,197,376,045,928,994,520,239,942,164,480 (approximately 181,577,189,197,376 yotta) nonisomorphic unfoldings.

**Related work.** In computational chemistry, fullerenes are of interest. Buckminsterfullerene (also known as icosahedral $C_{60}$, soccerballene, or truncated icosahedron) has 375,291,866,372,898,816,000 labeled unfoldings [5]. Handballene (also know as truncated dodecahedral $C_{60}$, or truncated dodecahedron) and Archimedean (also known as truncated icosidodecahedral $C_{120}$, or truncated icosidodecahedron) have 4,982,259,375,000,000,000 and 21,789,262,703,685,125,511,464,767,107,171,876,864, 000 labeled unfoldings, respectively [6]. In [13], the number of nonisomorphic unfoldings of a truncated octahedron is estimated to be approximately 2,300,000.

Akiyama et al. [1] are interested in the tessellation by unfoldings of polyhedra with regular polygonal faces, and the number of labeled unfoldings are investigated in their first step. [1] gives the number of labeled unfoldings of a cubotahedron, a truncated tetrahedron, and 17 out of 92 Johnson-Zalgaller solids.

Archimedean prisms and Archimedean antiprisms are also of interest. $n$-gonal prism and $n$-gonal antiprism has $\frac{n}{2}\{(2+\sqrt{3})^n+(2-\sqrt{3})^n-2\}$ [2] and $\frac{2n}{5}\{(2+\sqrt{3})^n+(2-\sqrt{3})^n-2\}$ [12] labeled unfoldings, respectively. As for the number of their nonisomorphic unfoldings, only the cases for $n$-gonal prism with $n=3$ to 14 are known [14].

## 2 Preliminaries

Let $\Gamma$, $V(\Gamma)$, $E(\Gamma)$, $\mathrm{Aut}\,\Gamma$, $\mathcal{T}(\Gamma)$ (or $\mathcal{T}$ in short) and $\mathcal{T}_g$ for $g \in \mathrm{Aut}\,\Gamma$, respectively, denote the 1-skeleton of a given polyhedron $P$, the set of vertices of $\Gamma$, the set of edges of $\Gamma$, the (symmetry) automorphism group of $\Gamma$,

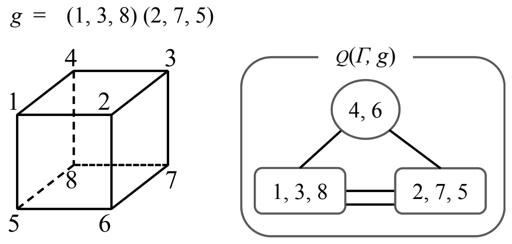


Figure 2: Quotient graph $\mathcal{Q}(\Gamma, g)$ for a rotation $g = (1,3,8)(2,7,5)$.

the set of spanning trees of $\Gamma$, and $\{T \in \mathcal{T} \mid T = gT\}$. Given a permutation $g \in \mathrm{Aut}\,\Gamma$, Fix $g$ is the subgraph of $\Gamma$ on which $g$ acts as the identity. In other words, The edges and vertices of Fix $g$ is defined as follows: $V(\text{Fix } g) = \{v \in V(\Gamma) \mid g(v) = v\}$, $E(\text{Fix } g) = \{(u,v) \in E(\Gamma) \mid u, v \in V(\text{Fix } g)\}$. If $g$ has no fixed point, i.e., $V(\text{Fix } g) = \emptyset$, we use notation Fix $g = \emptyset$. Let $\alpha(g)$ denote the number of $g$-invariant edges in $\Gamma$, i.e., $|\{(u,v) \in E(\Gamma) \mid (u,v) = (g(u), g(v)) \text{ or } (u,v) = (g(v), g(u))\}|$.

The quotient graph $\mathcal{Q}(\Gamma, g)$ is defined as follows. The orbit $\theta_v$ of vertex $v \in V(\Gamma)$ is the set of vertices $\{u \in V(\Gamma) \mid u = g^n(v), n \in \mathbb{Z}\}$. Let $\Omega$ be the set of orbits of length $> 1$. Then, the set $U$ of vertices of $\mathcal{Q}(\Gamma, g)$ is defined as follows: $U = \Omega$ if Fix $g = \emptyset$ holds. Otherwise, $U = \Omega \cup \{V(\text{Fix } g)\}$. Let $\pi$ be the natural projection

$$\pi : V \to U \begin{cases} v \to V(\text{Fix } g) & (\text{if } v \in V(\text{Fix } g)), \\ v \to \theta_v & (\text{otherwise}). \end{cases}$$

Let $E'$ denote the set $\{(u,v) \in E(\Gamma) \mid \pi(u) \neq \pi(v)\}$. The projection $\pi$ induces a map: $\tilde{\pi} : E' \to \mathcal{F} : (u,v) \to (\pi(u), \pi(v))$. The quotient graph $\mathcal{Q}(\Gamma, g)$ is defined as multigraph $(U, \mathcal{F})$, in which all edges in the same orbit of $E'$ corresponds to an edge in $\mathcal{F}$.

Figure 2 illustrates the quotient graph $\mathcal{Q}(\Gamma, g)$ for a rotation $g = (1,3,8)(2,7,5)$. The order of $g$ is 3. We can observe that $V(\text{Fix } g) = \{4,6\}$, $E(\text{Fix } g) = \emptyset$. Since we have two orbits $\{1,3,8\}$ and $\{2,7,5\}$, the set of vertices of $\mathcal{Q}(\Gamma, g)$ is $U = \{\{1,3,8\}, \{2,7,5\}, \{4,6\}\}$. Since we have two orbits of edges $\{(1,2),(3,7),(8,5)\}$ and $\{(1,5),(3,2),(8,7)\}$, $\mathcal{Q}(\Gamma, g)$ has two edges between $\{1,3,8\}$ and $\{2,7,5\}$.

In [4], in order to obtain $|\mathcal{T}_g|$ for any $g$ of regular convex polytopes in dimension $\leq 4$, the following three facts are used:

**Theorem 1 ([4])**

*(1) Let $g$ be of prime oder $p$ and let* Fix $g \neq \emptyset$. *Then,* $|\mathcal{T}_g| = |\mathcal{T}(\text{Fix } g)| \cdot |\mathcal{T}(\mathcal{Q}(\Gamma, g))|$.

*(2) Let $g$ be of order 2 with* Fix $g = \emptyset$. *Then,* $|\mathcal{T}_g| = |\mathcal{T}(\mathcal{Q}(\Gamma, g))| \cdot \alpha(g)$.

*(3) If* Fix $g \neq \emptyset$ *and* Fix $g$ *is not connected,* $|\mathcal{T}_g| = 0$ *holds.*

## 3 The number of nonisomorphic unfoldings

We extend Theorem 1 so as to obtain $|\mathcal{T}_g|$ for any $g$ of any polyhedron. Our extended theorem consists of the following four facts. The first one is a straightforward extension of Theorem 1(1), and the fourth one is new.

**Theorem 2**

*(1) Let* $\mathrm{Fix}\ g \neq \emptyset$ *and all vertices not in* $V(\mathrm{Fix}\ g)$ *have orbits of the same length. Then,* $|\mathcal{T}_g| = |\mathcal{T}(\mathrm{Fix}\ g)| \cdot |\mathcal{T}(\mathcal{Q}(\Gamma, g))|$.

*(2) Let $g$ be of order* 2 *with* $\mathrm{Fix}\ g = \emptyset$*. Then,* $|\mathcal{T}_g| = |\mathcal{T}(\mathcal{Q}(\Gamma, g))| \cdot \alpha(g)$.

*(3) If* $\mathrm{Fix}\ g \neq \emptyset$ *and* $\mathrm{Fix}\ g$ *is not connected,* $|\mathcal{T}_g| = 0$ *holds.*

*(4) If* $\mathrm{Fix}\ g = \emptyset$ *and* $\alpha(g) = 0$, $|\mathcal{T}_g| = 0$ *holds.*

In the rest of this section, we show that $|\mathcal{T}_g|$ can be obtained by any permutation in the symmetry group $\mathrm{Aut}\,\Gamma$ of any polyhedra.

**Theorem 3** *Let $P$ be a polyhedron, and* $\mathrm{Aut}\,\Gamma$ *be the symmetry group of $P$. Then, for any permutation* $g \in \mathrm{Aut}\,\Gamma$, *either of the four cases in Theorem 2 holds.*

## 4 Regular-faced convex polyhedra

By applying the proposed method to all regular-faced convex polyhedra, the numbers of nonisomorphic unfoldings are obtained. The results for the Archimedean solids and the Johnson-Zalgaller solids are listed in Tables 1 and 2. For Archimedean prisms and antiprisms, we have the following results.

**Theorem 4** *The number of nonisomorphic unfoldings of an $n$-gonal Archimedean prism is*

$$\begin{cases} \frac{1}{8\sqrt{3}}\{2\sqrt{3}\,n + \sqrt{3}(2+\sqrt{3})^n \\ \quad +(2+\sqrt{3})^{\lfloor \frac{n}{2} \rfloor}(4+2\sqrt{3}) \\ \quad +(2-\sqrt{3})^{\lfloor \frac{n}{2} \rfloor}(2\sqrt{3}-4) \\ \quad +\sqrt{3}((2-\sqrt{3})^n - 2)\} & (n \text{ is odd}), \\ \frac{1}{24}\{6n + 3(2+\sqrt{3})^n \\ \quad +4\sqrt{3}(2+\sqrt{3})^{\frac{n}{2}} \\ \quad -4\sqrt{3}(2-\sqrt{3})^{\frac{n}{2}} \\ \quad +3(2-\sqrt{3})^n - 6\} & (n \text{ is even}). \end{cases}$$

**Theorem 5** *The number of nonisomorphic unfoldings of an $n$-gonal Archimedean antiprism is*

$$\frac{1}{10}\left\{(\tfrac{1+\sqrt{5}}{2})^{4n} + (\tfrac{1+\sqrt{5}}{2})^{-4n} - 2\right\} + \frac{(3+\sqrt{5})^n - (3-\sqrt{5})^n}{2^{n+1}\sqrt{5}}.$$

## References

[1] J. Akiyama, T. Kuwata, S. Langerman, K. Okawa, I. Sato, G. C. Shephard: Determination of All Tessellation Polyhedra with Regular Polygonal Faces, *Proc. of the China-Japan Joint Conference on Computational Geometry, Graphs and Applications*, LNCS 7033, pp. 1–11, 2011.

[2] G. F. T. Boesch, Z. R. Bogdanowicz: The Number of Spanning Trees in a Prism, *Inter. J. Comput. Math.*, vol. 21, pp. 229–243, 1987.

[3] S. Bouzette, F. Vandamme: The regular Dodecahedron and Icosahedron unfold in 43380 ways, Unpublished manuscript.

[4] F. Buekenhout, M. Parker: The Number of Nets of the Regular Convex Polytopes in Dimension $\leq 4$, *Disc. Math.*, vol. 186, pp. 69–94, 1998.

[5] T. J. N. Brown, R. B. Mallion, P. Pollak, B. R. M. de Castro, J. A. N. F. Gomes: The number of spanning trees in buckminsterfullerene, *Journal of Computational Chemistry*, vol. 12, pp. 1118–1124, 1991.

[6] T. J. N. Brown, R. B. Mallion, P. Pollak, A. Roth: Some Methods for Counting the Spanning Trees in Labelled Molecular Graphs, examined in Relation to Certain Fullerenes, *Discrete Applied Mathematics*, vol. 67, pp. 51–66, 1996.

[7] A. Burnside: *Theory of Groups of Finite Order*, Cambridge University Press, 1911.

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

[9] C. Hippenmeyer: Die Anzahl der inkongruenten ebenen Netze eines regulären Ikosaeders, *Elem. Math.*, 34, 61–63, 1979.

[10] T. Horiyama and W. Shoji: Edge unfoldings of Platonic solids never overlap, *Proc. of the 23rd Canadian Conference on Computational Geometry*, pp. 65–70, 2011.

[11] M. Jeger: Über die Anzahl der inkongruenten ebenen Netze des Würfels und des regulären Oktaeders, *Elemente der Mathematik*, vol. 30, pp. 73–83, 1975.

[12] D. J. Kleitman, B. Golden: Counting trees in a certain class of graphs, *Am. Math. Monthly*, vol. 82, pp. 40–44, 1975.

[13] S. Pandey, M. Ewing, A. Kunas, N. Nguyen, D. H. Gracias, G. Menon: Algorithmic design of self-folding polyhedra, *Proc. Natl. Acad. Sci. USA*, 108 (50), 19885–19890, 2011.

[14] N. J. A. Sloane: Sequence A103535, The On-Line Encyclopedia of Integer Sequences.

Table 1: The number of edge unfoldings of Archimedean solids, where the entries without citation are newly obtained in this paper.

| Name | #(Labeled unfoldings) | #(Nonisomorphic unfoldings) |
|---|---|---|
| Cuboctahedron | 331,776 [1] | 6,912 |
| Icosidodecahedron | 208,971,104,256,000 | 1,741,425,868,800 |
| Truncated tetrahedron | 6,000 [1] | 261 |
| Truncated octahedron | 101,154,816 | 2,108,512 |
| Truncated cube | 32,400,000 | 675,585 |
| Truncated icosahedron | 375,291,866,372,898,816,000 [5] | 3,127,432,220,939,473,920 |
| Truncated dodecahedron | 4,982,259,375,000,000,000 [6] | 41,518,828,261,687,500 |
| Rhombicuboctahedron | 301,056,000,000 | 6,272,012,000 |
| Rhombicosidodecahedron | 201,550,864,919,150,779,950,956,544,000 | 1,679,590,540,992,923,166,257,971,200 |
| Truncated cuboctahedron | 12,418,325,780,889,600 | 258,715,122,137,472 |
| Truncated icosidodecahedron | 21,789,262,703,685,125,511,464,767,107,171,876,864,000 [6] | 181,577,189,197,376,045,928,994,520,239,942,164,480 |
| Snub cube | 89,904,012,853,248 | 3,746,001,752,064 |
| Snub dodecahedoron | 438,201,295,386,966,498,858,139,607,040,000,000 | 7,303,354,923,116,108,380,042,995,304,896,000 |

Table 2: The number of edge unfoldings of Johnson-Zalgaller solids, where the entries without citation are newly obtained in this paper.

| | #(Labeled unfoldings) | #(Nonisomorphic unfoldings) |
|---|---|---|
| J01 | 45 | 8 |
| J02 | 121 | 15 |
| J03 | 1,815 [1] | 308 |
| J04 | 24,000 | 3,030 |
| J05 | 297,025 | 29,757 |
| J06 | 78,250,050 | 7,825,005 |
| J07 | 361 [1] | 63 |
| J08 | 3,509 | 448 |
| J09 | 30,976 | 3,116 |
| J10 | 27,216 | 3,421 |
| J11 | 403,202 | 40,321 |
| J12 | 75 | 9 |
| J13 | 1,805 | 99 |
| J14 | 1,728 | 156 |
| J15 | 31,500 | 2,010 |
| J16 | 508,805 | 25,574 |
| J17 | 207,368 | 13,041 |
| J18 | 1,609,152 [1] | 268,260 |
| J19 | 227,402,340 | 28,427,091 |
| J20 | 29,821,320,745 | 2,982,139,245 |
| J21 | 8,223,103,375,490 | 822,310,337,549 |
| J22 | 37,158,912 [1] | 6,193,152 |
| J23 | 15,482,880,000 | 1,935,360,000 |
| J24 | 5,996,600,870,820 | 599,660,087,082 |
| J25 | 1,702,422,879,696,000 | 170,242,287,969,600 |
| J26 | 1,176 [1] | 152 |
| J27 | 324,900 [1] | 27,195 |
| J28 | 29,859,840 [1] | 1,867,560 |
| J29 | 30,950,832 [1] | 1,934,427 |
| J30 | 2,518,646,460 | 125,939,163 |
| J31 | 2,652,552,060 | 132,627,603 |
| J32 | 699,537,024,120 | 69,953,702,412 |
| J33 | 745,208,449,920 | 74,520,844,992 |
| J34 | 193,003,269,869,040 | 9,650,165,403,136 |
| J35 | 301,896,210 [1] | 25,158,925 |
| J36 | 302,400,000 [1] | 25,203,000 |
| J37 | 301,988,758,680 | 18,874,379,520 |
| J38 | 270,745,016,304,350 | 13,537,250,963,730 |
| J39 | 272,026,496,000,000 | 13,601,327,004,000 |
| J40 | 75,378,202,163,880,700 | 7,537,820,216,388,070 |
| J41 | 75,804,411,381,317,500 | 7,580,441,138,131,750 |
| J42 | 20,969,865,292,417,385,400 | 1,048,493,264,659,994,295 |
| J43 | 21,115,350,368,078,435,000 | 1,055,767,519,017,973,725 |
| J44 | 5,295,528,588 [1] | 882,609,105 |
| J45 | 13,769,880,349,680 | 1,721,235,971,518 |
| J46 | 32,543,644,773,848,180 | 3,254,364,517,723,165 |
| J47 | 9,324,488,558,669,593,960 | 1,864,897,711,733,918,792 |
| J48 | 2,670,159,599,304,760,178,000 | 267,015,959,942,030,583,130 |
| J49 | 672 | 173 |
| J50 | 5,544 | 1,401 |
| J51 | 42,336 | 3,549 |
| J52 | 16,744 | 4,201 |
| J53 | 153,816 | 38,526 |
| J54 | 75,973 [1] | 19,035 |
| J55 | 709,632 [1] | 88,776 |
| J56 | 707,232 [1] | 176,967 |
| J57 | 6,531,840 [1] | 544,680 |
| J58 | 92,724,962 | 9,272,497 |
| J59 | 1,651,482,010 | 82,580,526 |
| J60 | 1,641,317,568 | 410,335,964 |
| J61 | 28,745,798,400 | 4,790,966,400 |
| J62 | 28,080 | 7,050 |
| J63 | 1,734 | 289 |
| J64 | 8,450 | 1,409 |
| J65 | 1,245,456 [1] | 207,576 |
| J66 | 54,921,311,280 | 6,865,163,910 |
| J67 | 90,974,647,120,896 | 5,685,916,514,256 |
| J68 | 68,495,843,558,495,480,625,000 | 6,849,584,355,849,548,062,500 |
| J69 | 936,988,158,859,771,579,003,317,600 | 46,849,407,942,992,327,926,343,838 |
| J70 | 930,303,529,996,712,062,599,302,400 | 232,575,882,499,181,854,544,317,560 |
| J71 | 12,479,653,904,364,665,921,377,091,740,032 | 2,079,942,317,394,110,986,896,181,956,672 |
| J72 | 206,686,735,580,507,426,149,463,308,960 | 20,668,673,558,050,742,614,946,330,896 |
| J73 | 211,950,222,127,067,401,293,093,928,960 | 10,597,511,106,353,370,064,654,696,448 |
| J74 | 211,595,653,377,414,999,219,839,524,608 | 52,898,913,344,353,749,804,959,881,152 |
| J75 | 216,255,817,875,464,148,759,178,607,616 | 36,042,636,312,577,358,126,529,767,936 |
| J76 | 21,081,520,904,394,872,104,529,280 | 2,108,152,090,439,487,210,452,928 |
| J77 | 21,635,458,027,234,604,842,992,000 | 2,163,545,802,723,460,484,299,200 |
| J78 | 21,638,184,348,166,814,636,938,752 | 10,819,092,174,083,407,318,469,376 |
| J79 | 22,171,247,351,297,062,278,807,776 | 11,085,623,675,648,531,139,403,888 |
| J80 | 2,163,645,669,729,922,583,040 | 108,182,283,486,496,129,152 |
| J81 | 2,094,253,294,125,015,611,392 | 523,563,323,531,253,902,848 |
| J82 | 2,151,245,812,763,713,106,752 | 1,075,622,906,381,856,553,376 |
| J83 | 197,148,908,795,401,104 | 32,858,151,465,900,184 |
| J84 | 8,640 | 1,109 |
| J85 | 1,291,795,320 [1] | 80,742,129 |
| J86 | 84,480 | 21,204 |
| J87 | 652,846 | 326,423 |
| J88 | 2,002,440 | 500,959 |
| J89 | 32,373,600 | 8,094,150 |
| J90 | 519,556,800 | 64,950,268 |
| J91 | 870,912 | 108,936 |
| J92 | 235,726,848 | 39,287,808 |

# Computational Complexity of Piano-Hinged Dissections

Zachary Abel[*] Erik D. Demaine[†] Martin L. Demaine[†] Takashi Horiyama[‡] Ryuhei Uehara[§]

## Abstract

We prove NP-completeness of deciding whether a given loop of colored right isosceles triangles, hinged together at edges, can be folded into a specified rectangular three-color pattern. By contrast, the same problem becomes polynomially solvable with one color or when the target shape is a tree-shaped polyomino.

## 1 Introduction

One of the simplest and most practical physical folding structures is that of a *hinge*, as in most doors or attaching the lid to a grand piano. Frederickson [4] introduced a way to make folding structures out of such hinges that can change their shape between "nearly 2D" shapes. The basic idea is to thicken a (doubly covered) 2D polygon by extruding it orthogonally into a height-$2\varepsilon$ 3D prism, divide that prism into two height-$\varepsilon$ layers, further divide those layers into $\varepsilon$-thickened polygonal pieces, and hinge the pieces together with hinges along shared edges. The goal in a *piano-hinged dissection* is to find a connected hinging of $\varepsilon$-thickened polygonal pieces that can fold into two (or more) different $2\varepsilon$-thickened polygons.

Piano-hinged dissections are meant to be a more practical form of *hinged dissections*, which typically use point hinges and thus are more difficult to build [4]. Although hinged dissections have recently been shown to exist for any finite set of polygons of equal area [1], no such result is known for piano-hinged dissections.

Here we study a family of simple piano-hinged dissections, which we call a *piano-hinged loop*: $4n$ identical $\varepsilon$-thickened right isosceles triangles, alternating in orientation, and connected into a loop by hinges on the bottoms of their isosceles sides; see Figure 1. Frederickson [4, chapter 11] mentions without proof that this piano-hinged dissection can fold into any ($2\varepsilon$-thickened) $n$-omino, that is, any connected edge-to-edge joining of $n$ unit squares.

Three commercial puzzles, shown in Figure 2, consist of piano-hinged loops. *GeoLoop* is a piano-hinged loop with $n = 6$ that was patented by Kenneth Stevens in 1993/1994 [6] and sold by Binary Arts[1] in 1996. The pieces alternate between two colors, and by a checkerboard property of the piano-hinged loop, the resulting squares of any polyomino will alternate in color (on either side), so this puzzle is effectively *uncolored*. *Ivan's Hinge* is a piano-hinged loop with $n = 4$ that was patented by Jan Essebaggers and Ivan Moscovich in 1993/1994 [3] and sold by Paradigm Games in the mid-to-late 1990s [4] and recently by Fat Brain Toys[2]. Each piece is colored irregularly with one of two colors, and the goal in this puzzle is to make not only the specified tetromino shape but also the specified two-color pattern. *Tony's Hinge* is a variation of Ivan's Hinge, sold by Kellogg Company in 1988 but also copyright by Ivan Moscovitch and made by Paradigm Games. It uses colored images and requires putting certain images in particular places, in addition to the color constraints.

**Our results.** In this paper, we investigate the computational complexity of folding colored and uncolored piano-hinged loop puzzles into $n$-ominoes.

First we consider the uncolored piano-hinged loop, as in GeoLoop. For completeness, we prove Frederickson's claim that this loop can realize any $2\varepsilon$-thickened $n$-omino, by mimicking a simple inductive argument for hinged dissections of polyominoes from [2]. For the special case of *tree-shaped* polyominoes, where the dual graph of edge-to-edge adjacencies among unit squares forms a tree, we prove further that the folding of the piano-hinged loop is unique up to cyclic shifts of the pieces in the loop.

Next we consider colored piano-hinged loops, as in Ivan's Hinge. For tree-shaped polyominoes, the previous uniqueness result implies that the problem can be solved in $O(n^2)$ time by trying all cyclic shifts. (In particular, this observation makes the $n = 4$ case of Ivan's Hinge easy to solve in practice, as each tetromino has either 1 or 4 spanning trees to try.) For general polyominoes, we prove that the problem is NP-complete even if the number of colors is 3, each piece is colored uniformly one color, and the target shape is a rectangle.

[*] MIT Department of Mathematics, 77 Massachusetts Ave., Cambridge, MA 02139, USA, zabel@math.mit.edu

[†] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine, mdemaine}@mit.edu

[‡] Information Technology Center, Saitama University, horiyama@al.ics.saitama-u.ac.jp

[§] School of Information Science, JAIST, uehara@jaist.ac.jp

[1] Binary Arts changed its name to ThinkFun (http://www.thinkfun.com) in 2003.

[2] http://www.fatbraintoys.com

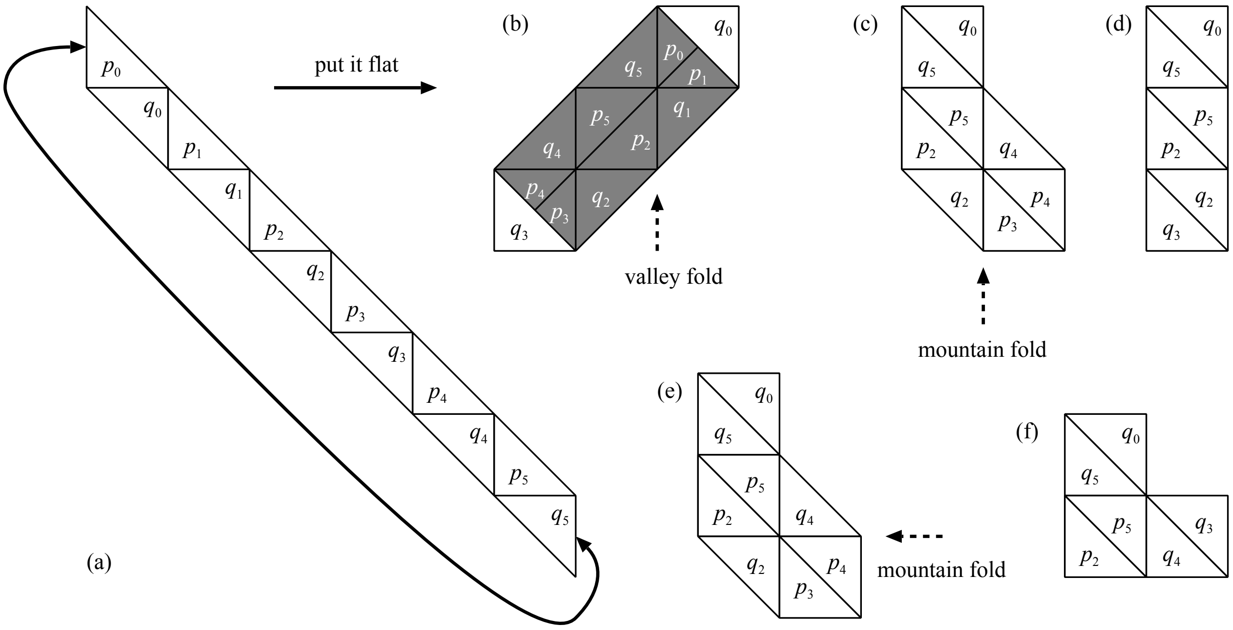

(a)
put it flat
(b)
valley fold
(c)
mountain fold
(d)
(e)
mountain fold
(f)
$p_0$ $q_0$ $p_1$ $q_1$ $p_2$ $q_2$ $p_3$ $q_3$ $p_4$ $q_4$ $p_5$ $q_5$

Figure 1: Piano-hinged loop with $n = 3$ (a), and two sequences of folding operations that result in a doubly covered straight tromino (b, c, d) or L tromino (b, e, f). Gray denotes the back side of the pieces.

Figure 2: GeoLoop, Ivan's Hinge, and Tony's Hinge.

## 2 Preliminaries

A *piano-hinged loop* consists of a loop of $4n$ consecutive isosceles right triangles $p_0, q_0, p_1, q_2, \ldots, p_{2n-1}, q_{2n-1}$, as shown in Figure 1. Every two consecutive triangular pieces share one of two isosceles edges. The $p_i$'s have a common orientation (collinear hypotenuses when unfolded), as do the $q_i$'s, and these two orientations differ from each other. Each shared edge is a *piano hinge* on the back side that permits bending inward (bringing the two back sides together).

In a *folded state* of the piano-hinged loop into a doubly covered polyomino, (1) each piano hinge is flat (180°) or folded inward (360°); and (2) each unit square of the polyomino consumes exactly four triangles, with two triangles on the front and two on the back side. Thus, in any folded state, the exposed surface consists of all front sides of the pieces, while the back sides of all pieces remain hidden on the inside. Therefore, we can ignore the color of the back side of each piece, so for simplicity we can assume that each piece has a uniform color (instead of a different color on each side). Let $c(p_i)$ and $c(q_i)$ denote the color of piece $p_i$ and $q_i$.

For the resultant polyomino $P$ of $n$ unit squares, we define the *connection graph* $G(P) = (V, E)$ as follows: $V$ consists of $n$ unit squares, and $E$ contains an edge $\{u, v\}$ if and only if squares $u$ and $v$ are adjacent (share an edge) in $P$. Having $\{u, v\} \in E$ is a necessary but not sufficient condition for there to be a hinge connecting the four pieces representing square $u$ to the four pieces representing square $v$; if there is such a hinge, we call $u$ and $v$ *joined*.

The *uncolored piano-hinged loop problem* asks whether a given polyomino can be constructed as (the silhouette of) a folded state of a given piano-hinged loop. The "silhouette" phrasing allows the folding to have unjoined squares, which are adjacent in the polyomino but not attached by a hinge in the folded state. The *colored piano-hinged loop problem* asks whether a given colored polyomino pattern can be similarly constructed from a given colored piano-hinged loop.

The piano-hinged loop has a simple checkerboarding property:

**Observation 1** Consider two adjacent squares $u$ and $v$ in a polyomino $P$, obtained as a folded state of a piano-hinged loop. Without loss of generality, assume that the top side of $u$ contains (the front side of) triangle $p_i$. Then (1) the other triangle of $u$ on front side is $p_j$ for some $j$, (2) the backside of $u$ contains two $q$s, (3) the front side of $v$ contains two $p$s, and (4) the backside of $v$ contains two $q$s.

Ivan's Hinge has a group of triangles that are monochromatic as assumed above, and a group of triangles with different colors on their front and back sides. However, these groups directly correspond to the parity classes in Observation 1. Hence, for each unit square, the front side consists of two triangles from the same group, and the back side consists of two triangles from the other group. Thus, from a theoretical point of view, we can again effectively assume that the pieces are monochromatic. (Practically, the differing colors can vary the color patterns, which can help visually.)

## 3 Uncolored Piano-Hinged Loop

We begin with the universality theorem of GeoLoop, claimed by Frederickson [4]:

**Theorem 1 ([4])** *Any polyomino $P$ of $n$ unit squares can be realized as a folded state of the piano-hinge loop of $4n$ pieces.*

Once we fix the spanning tree $T$ of $G(P)$, we claim that the folded state is uniquely determined up to cyclic shift of the pieces. Both this corollary and the previous theorem follow from a simple argument of repeatedly pruning leaves in the graph of joinings.

**Corollary 2** *Let $P$ be any polyomino of $n$ unit squares such that $G(P)$ is a tree. Then it can be uniquely folded from the piano-hinge loop of $4n$ pieces, up to cyclic shift of the pieces.*

For a given tree-shaped polyomino, the piano-hinge loop traverses the tree in the same manner as a depth-first search without crossing. That is, if we imagine that we are in the maze in the form of the tree, and traverse the maze by the right-hand rule, then we traverse each edge twice, and this is the order followed by the piano-hinge loop. This intuition will be useful in some proofs in this paper.

## 4 General Piano-Hinged Loop

Consider a polyomino $P$ in which pieces $p_i$ and $q_i$ have colors $c(p_i)$ and $c(q_i)$, respectively. When the connection graph $G(P)$ is a tree (or the spanning tree of $G(P)$ is explicitly given), we still have a polynomial-time algorithm to solve the problem:

**Theorem 3** *Let $P$ be any polyomino of $n$ unit squares such that $G(P)$ is a tree $T$. Then the general piano-hinge loop problem can be solved in $O(n^2)$ time.*

Next we turn to the case that $P$ is a general polyomino, where the problem is NP-complete:

**Theorem 4** *The colored piano-hinge loop problem is NP-complete, even if the number of colors is 3 and the target polyomino is a rectangle.*

**Proof outline:** We prove NP-hardness by reducing from 3-PARTITION, defined as follows.

**3-PARTITION** (cf. [5])
INSTANCE: A finite set $A = \{a_1, a_2, \ldots, a_{3m}\}$ of $3m$ weighted elements with $w(a_j) \in \mathbb{Z}_+$, where $w(a_j)$ gives the weight of $a_j$, and a bound $B \in \mathbb{Z}_+$ such that each $a_j$ satisfies $B/4 < w(a_j) < B/2$ and $\sum_{j=1}^{3m} w(a_j) = mB$.
QUESTION: Can $A$ be partitioned into $m$ disjoint sets $A^{(1)}, A^{(2)}, \ldots, A^{(m)}$ such that $\sum_{a_j \in A^{(i)}} w(a_j) = B$ for $1 \le i \le m$?

If $A$ has a solution, then each $A^{(i)}$ must contain exactly three items, because $B/4 < w(a_j) < B/2$, for all $i$ and $j$.

Figure 3 outlines the construction. Our piano-hinge loop $L$ consists of two parts (Figure 3(a)). The first part is a series of black triangles that form $m$ empty *bins*, such that each bin should be filled by $B$ gray unit squares. The second part is a sequence of gray and white triangles. The $i$th run of $4a_i$ consecutive grey triangles in the sequence represents the weight of an element $a_i$ for each $i$. White triangles will be used to place the grey items arbitrary into bins. The key point is that each run of gray triangles must be put into exactly one bin, and the grey triangles cannot be split into two or more different bins. Using this property, we simulate 3-PARTITION. Figure 4 shows one of the gadgets from the full proof.

Adding $3m \times (b+3) - 1$ black squares in the lower right of Figure 3(b), we can make the desired shape a rectangle of size $15m \times (12m + b + 3)$. $\square$

## References

[1] T. G. Abbott, Z. Abel, D. Charlton, E. D. Demaine, M. L. Demaine, and S. D. Kominers.

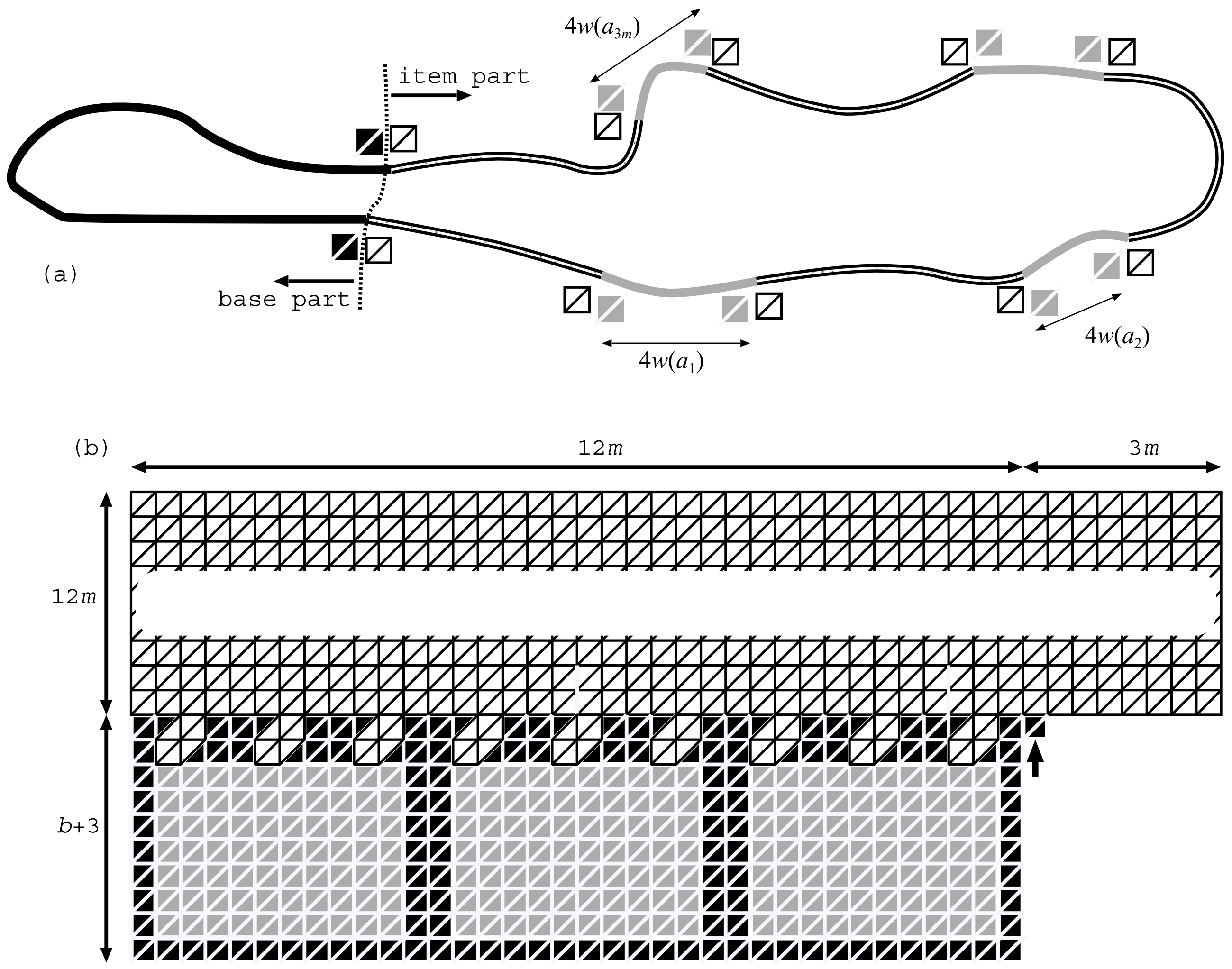


Figure 3: Outline of the construction

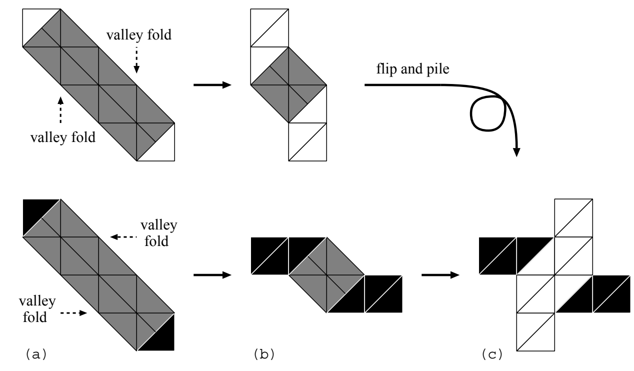


Figure 4: Crossover gadget

Hinged dissections exist. *Discrete & Computational Geometry*, 47(1):150–186, 2012.

[2] E. D. Demaine, M. L. Demaine, D. Eppstein, G. N. Frederickson, and E. Friedman. Hinged dissection of polyominoes and polyforms. *Computational Geometry: Theory and Applications*, 31(3):237–262, June 2005.

[3] J. Essebaggers and I. Moscovich. Triangle hinged puzzle. European Patent EP0584883. Filed August 25, 1993. Awarded March 2, 1994.

[4] G. N. Frederickson. *Piano-hinged Dissections: Time to Fold!* A K Peters, 2006.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[6] K. V. Stevens. Folding puzzle using triangular blocks. United States Patent 5,299,804. Filed January 19, 1993. Awarded April 5, 1994.

# Coding Ladder Lotteries[*]

Tomoki Aiuchi[†‡] Katsuhisa Yamanaka[†§] Takashi Hirayama[†¶] Yasuaki Nishitani[†‖]

## Abstract

A *ladder lottery*, known as the "Amidakuji" in Japan, is a common way to choose a permutation randomly. In this paper, we consider a problem of coding optimal ladder lotteries. We first propose two codes: a route-based code and a line-based code for an optimal ladder lottery with $n$ vertical lines and $b$ horizontal lines. The lengths of two codes are $n(n-1)$ bits and $n+2b-1$ bits, respectively. As a by-product of coding, we derive an upper bound on the number of ladder lotteries with $n$ vertical lines and $b$ horizontal lines. Furthermore, by improving the line-based code, we propose a more compact code. The length of the improved code is $n+2b-1$ bits in the worst case. However we experimentally show that the improved code is more compact than the two codes.

## 1 Introduction

In this paper we consider a problem of coding ladder lotteries. A *ladder lottery*, known as the "Amidakuji" in Japan, is a common way to choose a permutation randomly. Formally, a ladder lottery $L$ of a permutation $\pi = (p_1, \ldots, p_n)$ is a network with $n$ vertical lines (*lines* for short) and many horizontal lines (*bars* for short) each of which connects two consecutive vertical lines. See Figure 1 for an example. The $i$th line from the left is called *line $i$*. The top endpoints of lines correspond to $\pi$. The bottom endpoints of lines correspond to the identical permutation $(1, \ldots, n)$. Each element $p_i$ in $\pi$ starts the top endpoint of line $i$, and goes down along the line, then whenever $p_i$ comes to an endpoint of a bar, $p_i$ goes horizontally along the bar to the other endpoint, then goes down again. Finally $p_i$ reaches the bottom endpoint of line $p_i$. This path is called the *route* of element $p_i$.

A ladder lottery of a permutation $\pi = (p_1, \ldots, p_n)$ is *optimal* if it consists of the minimum number of bars among ladder lotteries of $\pi$. Let $L$ be an optimal ladder lottery of $\pi$ with $n$ lines and $b$ bars. Then we can observe that $b$ is equal to the number of *inversions*

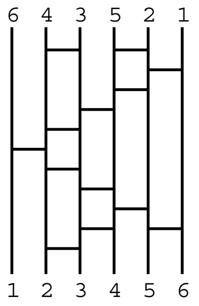


Figure 1: An optimal ladder lottery of (6,4,3,5,2,1).

of $\pi$, which is a pair $(p_i, p_j)$ in $\pi$ with $p_i > p_j$ and $i < j$. The ladder lottery in Figure 1 has thirteen bars, and the corresponding permutation (6,4,3,5,2,1) has thirteen inversions, so the ladder lottery is optimal.

Optimal ladder lotteries are related to pseudoline arrangements. There is one-to-one correspondence between optimal ladder lotteries of $\pi = (n, \ldots, 1)$, namely a reverse permutation, and arrangements of $n$ pseudolines [5].

Coding pseudoline arrangements has been investigated to obtain upper bounds of the number of pseudoline arrangements [2, 3]. Knuth [4] showed that the number of arrangements of $n$ pseudolines is bounded by $2^{0.792n^2}$. Based on coding pseudoline arrangements, Felsner [2] improved the upper bound to $2^{0.697n^2}$. Recently, Felsner and Valtr [3] improved to $2^{0.657n^2}$ based on coding. To the best of our knowledge, there is no previous work that addresses coding ladder lotteries of any permutation.

In this paper we design several codes for optimal ladder lotteries with $n$ lines and $b$ bars. A class of optimal ladder lotteries with $n$ lines is a generalization of a class of arrangements of $n$ pseudolines. For an optimal ladder lottery, $b \le \binom{n}{2}$ holds, and $b = \binom{n}{2}$ holds if and only if the optimal ladder lottery corresponds to a reverse permutation, that is the optimal ladder lottery correspond to an arrangement of $n$ pseudolines.

We first propose two codes: a route-based code and a line-based code. The lengths of two codes are $n(n-1)$ bits and $n+2b-1$ bits, respectively. Then, by improving the line-based code, we propose an improved code. We compare them in the sense that (1) the length of a code is compact and (2) time complexities of coding and decoding are efficient. Table 1 is a comparison of proposed codes. The length of the improved code is $n+2b-1$ bits in the worst case.

[*]This work was supported by JSPS KAKENHI Grant Number 23700012.

[†]Department of Electrical Engineering and Computer Science, Iwate University
[‡]Currently he is a member of Leadkonan
[§]`yamanaka@cis.iwate-u.ac.jp`
[¶]`hirayama@cis.iwate-u.ac.jp`
[‖]`nisitani@cis.iwate-u.ac.jp`

Table 1: The lengths, coding time and decoding time of our codes for a ladder lottery with $n$ lines and $b$ bars.

| | Length (bits) | Coding time | Decoding time |
|---|---|---|---|
| Route-based | $n(n-1)$ | $\mathrm{O}(n^2)$ | $\mathrm{O}(n^2)$ |
| Line-based | $n+2b-1$ | $\mathrm{O}(n+b)$ | $\mathrm{O}(n+b)$ |
| Improved | $n+2b-1$ | $\mathrm{O}(n+b)$ | $\mathrm{O}(n+b)$ |

However we experimentally show that the improved code is more compact than the other codes. On the other hand, as a by-product of coding, we derive an upper bound on the number of ladder lotteries with $n$ lines and $b$ bars.

Throughout of this paper, we assume that the end-of-file is prepared as a special character. Hence the codes proposed in this paper contain no end-of-file.

Due to space limitations, all proofs in this extended abstract are omitted. They are contained in a full version of the paper [1].

## 2 A Route-based Code

The coding idea for pseudoline arrangements by Felsner [2] can be applied for optimal ladder lotteries. In this section we propose a code which is straightforwardly derived from his coding [2].

The coding idea is to represent each route as a binary code. The route turns left or right zero or more times. We encode the directions (left or right) into a binary code.

Let $L$ be an optimal ladder lottery of a permutation $\pi = (p_1, \dots, p_n)$. Suppose a route of $p_i$ turns left or right $t(i)$ times. We note that $t(i) \leq n-1$ holds. Define $d^i_j =$ '0' if the $j$th direction of the route of $p_i$ is left, and $d^i_j =$ '1' if the $j$th direction of the route of $p_i$ is right. We construct a binary code $d^i_1 \dots d^i_{t(i)}$ by arranging $d^i_j$s, and then append '0's so that the length of the code becomes $n-1$ bits. We call the code a *route-code* of $p_i$.

We define a code for $L$. A *route-based code*, denoted by $RC$, for $L$ is the code obtained by arranging route-codes of $p_1, \dots, p_n$ in this order. The length of $RC$ is $n(n-1)$ bits. For instance, a route-based code for the optimal ladder lottery in Figure 1 is "111111101001010110000100000000".

Now we explain how to reconstruct the original ladder lottery from a route-based code.

Let $RC$ be a route-based code for an optimal ladder lottery $L$ with $n$ lines and $b$ bars. Since we know the length of $RC$ is $n(n-1)$ bits, we can calculate the value of $n$ by computing the length of $RC$ and hence recognize the boundary between any two consecutive route-codes. Let focus on the first bit in each route-code. It represents the first direction of each route. By arranging the first bits in route-codes, we define a *direction vector* $DV_1 = (d^1_1, \dots, d^n_1)$.

Now, we show that a bar can be reconstructed from $DV_1$. In $DV_1$, $j$ is *flip* if $d^j_1 =$ '1' and $d^{j+1}_1 =$ '0' holds. This represents an intersection, namely a bar, of the two routes $p_j$ and $p_{j+1}$. $DV_1$ may contain multiple flips. Among them we call the minimum flip the *first flip*. Intuitively, the first flip in $DV_1$ corresponds to the upper-left bar in $L$.

Let $j$ be the first flip. We reconstruct a bar in which two routes of $p_j$ and $p_{j+1}$ intersect. Then we construct $DV_2$ from $DV_1$ by passing through the bar. That is, we replace $d^j_1$ with $d^j_2$ and $d^{j+1}_1$ with $d^{j+1}_2$, and swap $d^j_2$ with $d^{j+1}_2$. $DV_2$ represents the next directions of routes after passing the bar corresponding to the first flip in $DV_1$. By repeatedly applying this process for $DV_2, \dots, DV_b$, we can reconstruct all bars. Then, we obtain $DV_{b+1}$ consisting of only '0's. A direction vector may contain $d^j_n$ which is not defined in route-code of $p_j$. For convenience, we assume that $d^j_n =$ '0'. The last direction vector $DV_{b+1}$ contains only '0's.

By maintaining a list of flips for each $DV_i$, $(1 \leq i \leq b)$, we can decode in $\mathrm{O}(n^2)$ time.

Since a flip always corresponds to a bar and each $DV_i$ $(1 \leq i \leq b)$ contains at least one flip (See [1] for details), we have the following theorem.

**Theorem 1** *We can encode an optimal ladder lottery into a code of length $n(n-1)$ bits. Coding and decoding can be done in $O(n^2)$ time.*

## 3 A Line-based Code

We propose a new code for an optimal ladder lottery in this section. This code focuses on lines in an optimal ladder lottery. We represent each line as a sequence of endpoints of bars.

A line-based code can be applied to a (non-optimal) ladder lottery. However, for convenience, we deal with only an optimal ladder lottery in this section. In the next section we improve the line-based code by using properties of an optimal ladder lottery.

Let $L$ be an optimal ladder lottery with $n$ lines and $b$ bars. We denote by $V(i) = (v_1, \dots, v_{|V(i)|})$ a sequence of endpoints of bars on a line $i$ from the top to the bottom. We encode $V(i)$ with $C(i) = (c_1 \dots c_{|V(i)|})$ where $c_j =$ '0' if $v_j$ is a right endpoint on a line $i$ and $c_j =$ '1' if $v_j$ is a left endpoint on a line $i$. We call $C(i)$ a *line-code* for a line $i$. Finally we concatenate line-codes $C(i)$ for $i = 1, \dots, n$ in this order, and insert '0' between any consecutive two line-codes to represent a boundary. We call the obtained code a *line-based code*, and denote by $LC$. For example, a line-based code for the ladder lottery in Figure 1 is "10110110010011001100100010010000".

Now we explain how to reconstruct the original optimal ladder lottery from a line-based code. Let $LC$ be a line-based code for an optimal ladder lottery $L$.

In $LC$, a '0' represents either a right endpoint of a bar or a boundary of consecutive two line-codes. A key of reconstruction is to recognize boundaries in a line-based codes. If the boundaries in $LC$ are recognized, then it is easy to reconstruct each line $i$ for $i = 1, \ldots, n$ from the corresponding line-code.

We explain how to recognize boundaries. Since the line 1 contains only left endpoints, the first consecutive '1's correspond to $C(1)$ and the first '0' is the boundary between $C(1)$ and $C(2)$. Now we assume that the line $i-1$ is reconstructed and the boundary between line-codes $C(i-1)$ and $C(i)$ is recognized . Then, we know the number of left endpoints of bars on a line $i-1$. Since it equals to the number of right endpoints of bars on a line $i$, we obtain the number of '0's in $C(i)$. Hence, the boundary between line-codes $C(i)$ and $C(i+1)$ can be recognized in $LC$, and a line $i$ is reconstructed from $C(i)$.

We estimate the length of $LC$ for an optimal ladder lottery with $n$ lines and $b$ bars. Since $LC$ contains 2 bits for each bar and 1 bit for each boundary, its length is $n + 2b - 1$ bits.

**Theorem 2** *Let $L$ be an optimal ladder lottery with $n$ lines and $b$ bars. There exists a code for $L$ of length $n + 2b - 1$ bits. Coding and decoding can be done in $O(n + b)$ time.*

Since a line-based code includes exactly $b$ '1's, the number of every possible line-code for an optimal ladder lottery with $n$ lines and $b$ bars is at most $\binom{n+2b-1}{b}$. We obtain the following corollary.

**Corollary 3** *The number of optimal ladder lotteries with $n$ lines and $b$ bars is at most $\binom{n+2b-1}{b}$.*

## 4 Improvements

In this section we design a code by improving the line-based code. The length of the proposed code is at most $n + 2b - 1$ bits. On the other hand, we show that the code is more compact than the original line-based code by an experiment in the next section.

First, we introduce three improvement ideas. By combining the ideas, we propose a code.

Let $L$ be an optimal ladder lottery with $n$ lines and $b$ bars, and $LC$ its line-based code.

### Improvement 1

The first improvement idea is to save the line-code for the line $n$. There are only right endpoints of bars on $n$, and the number of right endpoints of bars on $n$ is equal to the number of left endpoints of bars on $n-1$. Hence, even if the line-code for $n$ and its preceding '0' corresponding to a boundary are omitted, we can reconstruct the original optimal ladder lottery.

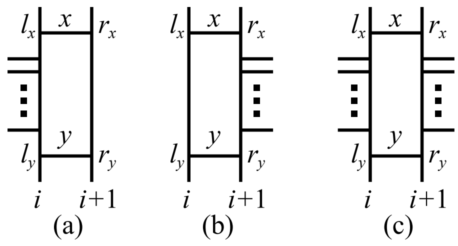


Figure 2: Possible situations for consecutive two bars.

### Improvement 2

We show a useful property of an optimal ladder lottery for saving bits.

Let $x, y$ be two consecutive bars with left endpoints on a line $i$ and right endpoints on a line $i+1$ in an optimal ladder lottery (See Figure 2). We denote by $l_x, r_x$ the left and the right endpoints of $x$. Similarly, we denote by $l_y, r_y$ the left and the right endpoints of $y$. Then we have the following lemma.

**Lemma 4** *A configuration between the two bars is either of three cases: (1) there is at least one right endpoint between $l_x$ and $l_y$ and no endpoint between $r_x$ and $r_y$ (Figure 2(a)), (2) there is no endpoint between $l_x$ and $l_y$ and at least one left endpoint between $r_x$ and $r_y$ (Figure 2(b)), or (3) there is at least one right endpoint between $l_x$ and $l_y$ and at least one left endpoint between $r_x$ and $r_y$ (Figure 2(c)).*

Lemma 4 gives an idea for saving bits. If there is no endpoint between $l_x$ and $l_y$, then there always exists at least one endpoint between $r_x$ and $r_y$ (Figure 2(b)). Hence we can save one '1' representing the endpoint immediately under $r_x$.

For example, we obtain "1011011000010011000000001000" by applying this idea to $LC$ for the ladder lottery in Figure 1.

Reconstruction of $LC$ with this idea can be done in a straightforward way. The line 1 is easily reconstructed from the corresponding line-code. Suppose a line $i$ is reconstructed. Then we can reconstruct the saved '1's in a line-code for a line $i+1$ by checking the reconstructed line $i$.

### Improvement 3

Along with improvement 2, we can save some bits for right endpoints of bars on the line $n-1$. Since the line $n$ has no left endpoint of any bar, any pair of consecutive two bars between lines $n-1$ and $n$ always forms the configuration illustrated in Figure 2(a). Let $x, y$ be any consecutive two bars with left endpoints on the line $n-1$ and right endpoints on the line $n$.

We denote by $l_x, r_x$ the left and the right endpoints of $x$. Similarly, we denote by $l_y, r_y$ the left and the right endpoints of $y$. Since there is no endpoint between $r_x$ and $r_y$, there is at least one right endpoint of a bar between $l_x$ and $l_y$. Therefore we can save one '0' representing the right endpoint immediately under $l_x$. By this improvement zero or more '0's in the line-code for $n-1$ are saved.

For example, $LC$ with improvement 3 for the ladder lottery in Figure 1 is "101101100100110011001000101000". In this example, one '0' for the line 5 is saved.

We briefly explain how to reconstruct. See [1] for further details. First we apply the decoding of a line-based code without improvement. If there are saved '0's in the line-code for $n-1$, the reconstruction fails, since the number of '0's is insufficient. Otherwise, $L$ is reconstructed correctly.

In the case of the failure, we retry the decoding of two line-codes for $n-1$ and $n$ more carefully. Saved '0's are recognized easily, since such '0's appear immediately after '1's except the last '1' in the line-code for $n-1$.

### Combining improvements 1-3

Now we show that the three improvements can be applied to a line-based code simultaneously.

The three improvements have no duplicate of saved '0's or '1's. Hence, we can apply improvements 1-3 to a line-based code directly. We call the obtained code an *improved code* for an optimal ladder lottery. We denote by $IC$ the improved code and by $C_I(i)$ a line-code for a line $i$ in $IC$. For example, $IC$ for the optimal ladder lottery in Figure 1 is "10110110000100110000001".

Now we explain how to decode $IC$. The decoding is a similar way for a line-based code with improvement 3. First, we apply a decoding of a line-based code with improvements 1 and 2 to $IC$(See [1] for a detail). If the decoding is a failure, the number of '0's is insufficient, we retry to decode $C_I(n-1)$ carefully. Otherwise, $L$ can be reconstruct correctly.

We have the following theorem.

**Theorem 5** *Let $L$ be an optimal ladder lottery with $n$ lines and $b$ bars. We can compute an improved code $IC$ in $O(n+b)$ time, and reconstruct $L$ from $IC$ in $O(n+b)$ time. The length of $IC$ is $n+2b-1$ bits in the worst case.*

## 5 Experimental Result

We compare the average lengths of the codes proposed in previous sections.

An environment for an experiment is as follows. (1) OS: FreeBSD 8.2-RELEASE, (2) CPU: AMD Phenom(tm) II X6 1065T Processor (2909.62-MHz K8-class CPU), (3) Main memory: 4GB and (4) Programming language: Common lisp (5) Compiler: SBCL 1.0.57.

Processes in our experiment are as follows. First we enumerate all permutations of $n$ elements. Second all optimal ladder lotteries of each permutation are enumerated. Third three codes $RC$, $LC$ and $IC$ are computed for every optimal ladder lottery. Finally we calculate the average length of each code per an optimal ladder lottery.

Table 2 shows average lengths of $RC$, $LC$ and $IC$.

Table 2: The average length of each code per an optimal ladder lottery.

| $n$ | $RC$ (bits) | $LC$ (bits) | $IC$ (bits) |
|---|---|---|---|
| 2 | 2.0 | 2.0 | 1.0 |
| 3 | 6.0 | 5.4 | 3.3 |
| 4 | 12.0 | 10.7 | 7.7 |
| 5 | 20.0 | 18.1 | 14.1 |
| 6 | 30.0 | 27.8 | 22.4 |
| 7 | 42.0 | 39.5 | 32.7 |
| 8 | 56.0 | 53.3 | 44.8 |
| 9 | 72.0 | 69.2 | 58.8 |

Table 2 shows that $IC$ is the most compact among the three codes. Although the length of $IC$ is $n+2b-1$ bits in the worst case, the experimental result shows that the average length of $IC$ is quite compact. This implies that our improvements are valid on saving bits.

## References

[1] T. Aiuchi, K. Yamanaka, T. Hirayama, and Y. Nishitani. Coding ladder lotteries. *Technical Report IWATE-CIS-13-01, http://www.kono.cis.iwate-u.ac.jp/~yamanaka/Tech/IWATE-CIS-13-01.pdf*, 2013.

[2] S. Felsner. On the number of arrangements of pseudolines. In *Proc. The 12th annual Symposium on Computational Geometry, (SCG 1996)*, pages 30–37, 1996.

[3] S. Felsner and P. Valtr. Coding and counting arrangements of pseudolines. *Discrete & Computational Geometry*, 46:405–416, 2011.

[4] D.E. Knuth. *Axioms and hulls.* LNCS 606, Springer-Verlag, 1992.

[5] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada. Efficient enumeration of all ladder lotteries and its applications. *Theoretical Computer Science*, 411:1714–1722, 2010.

# On Counting Triangles, Quadrilaterals and Pentagons in a Point Set

Sergey Bereg* Kira Vyatkina†

**Abstract**

We propose a novel approach to counting empty triangles, quadrilaterals and pentagons with vertices in a planar set $P$ of points, which extends to the case of those polygons containing precisely or at most $k$ points from $P$. For the former tasks, our methods require $O(n^3)$ time in case of triangles and quadrilaterals, and $O(n^4)$ time in case of pentagons; the latter task is accomplished in $O(n^3 \log k)$, $O(k \cdot n^3 \log k)$, and $O(k \cdot n^4 \log k)$ time in case of triangles, quadrilaterals, and pentagons, respectively. The space requirements are always linear. In addition, our algorithms can output the number of non-convex (and thus, of convex) quadrilaterals and pentagons for each problem statement.

## 1 Introduction

In the last two decades, the problem of counting $k$-gons with vertices in a set $P$ of $n$ points, which would possess certain properties, has been addressed several times in the literature. In particular, the task of counting convex $k$-gons in a point set was first considered by Rote et al. [4], who gave an algorithm for solving it in time $O(n^{k-2})$. Later on, this time bound was improved to $O(n^{\lceil k/2 \rceil})$ by Rote and Woeginger [5]. In 1995, Mitchell et al. [3] proposed algorithms for computing the total number of convex polygons in $P$ in $O(n^3)$ time, and tabulating the number of convex $k$-gons in $O(m \cdot n^3)$ time, for all $k = 3, \ldots, m$. If the polygons are required to be empty, the respective tasks can be accomplished within the same time bounds. Mitchell et al. [3] also mentioned that the number of $t$-subsets $T \subseteq P$, the convex hull of which is a $k$-gon, can be computed in $O(k(t-k+1)n^4)$ time. Very recently, a method for counting empty pseudo-triangles in a point set has been proposed by Kopeliovich and Vyatkina [2], which runs in $O(n^5)$ time and linear space.

In this work, we propose novel algorithms for counting empty triangles, quadrilaterals and pentagons in $P$, as well as those containing precisely or at most $k$ point from $P$, thereby imposing no restrictions on the polygons. For triangles and quadrilaterals, our methods accomplish the former task in $O(n^3)$ time and $O(n)$ space, while for the latter task, the running time amounts to $O(n^3 \log k)$ and $O(k \cdot n^3 \log k)$ for triangles and quadrilaterals, respectively. For pentagons, the time bounds increase to $O(n^4)$ and $O(k \cdot n^4 \log k)$ for the former and latter task, respectively. Our approach can also distinguish between convex and non-convex polygons, and count them separately within the same complexity bounds.

To simplify the exposition, throughout this paper, we assume that no three points from $P$ are collinear.

## 2 Triangles

First, we show how to count empty triangles defined by a set $P$ of $n$ points. Next, we demonstrate that our approach can be easily extended to count the triangles with vertices in $P$, which contain exactly or at most $k$ points from $P$ in their interior.

For any point $a \in P$, let $P_a = P \setminus \{a\}$; for any two distinct points $a, b \in P$, let $P_{ab} = P \setminus \{a, b\}$.

Consider an ordered pair $(a, b)$ of points from $P$. We start by computing the number of points $c \in P_{ab}$, such that the triangle $\triangle abc$ is empty, and its vertices $a, b, c$ are in ccw order. To this end, we suggest the following approach.

Without loss of generality, let us assume that $ab$ is horizontal, and $b$ lies strictly to the right of $a$. Thus, $c$ must reside strictly above $ab$.

Let $p_1, p_2, \ldots, p_{n-2}$ denote the points from $P_{ab}$ sorted in angular order around $a$. Based on the above observation, we restrict our attention to the (sorted) subset $P' = \{p_1, p_2, \ldots, p_m\}$ of points from $P$ lying strictly above $ab$, where $0 \leq m \leq n-2$. For a point $p_i \in P'$, let $s_i$ denote the slope of the line $(bp_i)$, where $1 \leq i \leq m$. Throughout the computations, we maintain the maximum slope $s_{max}$ over those of the lines through $b$ and the points from $P'$ processed so far.

We initialize a counter $N_{ab}$ and $s_{max}$ to one and the slope of the line $(bp_1)$, respectively, and scan the points $p_2, \ldots, p_m$ one by one. When processing a point $p_j$, where $2 \leq j \leq m$, we compare $s_j$ with $s_{\max}$. If the former is greater, $N_{ab}$ is incremented by one, and $s_{max}$ is updated; otherwise, we proceed immediately to the next point from $P'$ (Fig. 1). After the last point $p_m$ is handled, $N_{ab}$ will store the number of empty triangles $\triangle abc$ with $a, b, c$ in ccw order.

*Department of Computer Science, University of Texas at Dallas, Box 830688, Richardson, TX 75083, USA. E-mail: besp@utdallas.edu

†Algorithmic Biology Laboratory, Saint Petersburg Academic University, the Russian Academy of Sciences, 8/3 Khlopina str., Saint Petersburg 194021, Russia E-mail: kira@math.spbu.ru

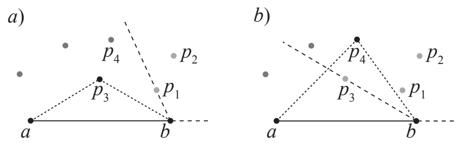


Figure 1: Empty triangles $\triangle abc$ with $a, b, c$ in ccw order are being counted. The points from $P'$, which have already been processed, the one being handled, and those unprocessed are shown light gray, black, and dark gray, respectively. a) When $p_3$ is handled, the maximum slope over those of the lines through $b$ and the points from $P'$ processed earlier is $s_1$; $s_3 > s_1$, and thus, $\triangle abp_3$ is empty. b) When $p_4$ is handled, the maximum slope over those of the lines through $b$ and the points from $P'$ processed earlier is $s_3$; $s_4 < s_3$, and thus, $\triangle abp_4$ is non-empty.

For a single ordered pair $(a, b)$ of points from $P$, the above algorithm takes linear time, assuming the points from $P_a$ are pre-sorted with respect to $a$. Otherwise, the running time increases to $O(n \log n)$. Thus, a naive approach to counting empty triangles in $P$, which would examine each ordered pair $(a, b)$ independently, would take $O(n^3 \log n)$ time. However, all the sorted orders of points can be easily computed in $O(n^2 \log n)$ in total, if the sorted order of points for each $a \in P$ is retrieved in $O(n \log n)$ time and then used for all the $n - 1$ pairs $(a, b)$, where $b \in P_a$. Alternatively, an asymptotically more efficient approach can be undertaken, which uses the standard method of computing the arrangement of dual lines [1], and makes available all the sorted orders of points in $O(n^2)$ total time.

This reasoning leads to the following theorem.

**Theorem 1** *Let $P$ be a set of $n$ points in general position in the plane. The number of empty triangles in $P$ can be computed in $O(n^3)$ time using $O(n)$ space.*

**Proof.** First, for each point $a \in P$, retrieve the sorted orders of the points from $P_a$ in total $O(n^2)$ time. Next, for each ordered pair $(a, b)$ of points from $P$, compute the number of empty triangles $\triangle abc$, where $c \in P_{ab}$ and $a, b, c$ are in ccw order. Then sum up the obtained numbers. Since in this way, each empty triangle will be counted three times (once for each side), the resulting value needs to be divided by three. This gives the total number of empty triangles in $P$. □

Now suppose we want to count the triangles with vertices in $P$, which contain inside precisely one point from $P$. It is easy to see that to this end, the same approach can be applied, but when processing a pair $(a, b)$ and counting the triangles $\triangle abc$ with $a, b, c$ in ccw order, inside which lies exactly one point from $P$, we shall need to maintain the *two* largest slopes $s_{max}$, $s_{sec}$ of the lines through $b$ and the points from $P'$ processed so far. When processing the next point $p_j \in P'$, we first compare $s_j$ with $s_{sec}$. Is the former is smaller, we proceed to the next point. Otherwise, if $s_j$ is greater than $s_{max}$, both $s_{max}$ and $s_{sec}$ are updated: the latter is set to the current $s_{max}$, and the former – to $s_j$. Finally, if neither is the case, and thus, $s_j$ falls between $s_{max}$ and $s_{sec}$, the counter $N_{ab}$ is incremented by one, and $s_{sec}$ is set to $s_j$. Obviously, these modifications affect neither the running time nor the memory requirements of the algorithm. Note that if we were interested in the number of triangles with vertices in $P$ containing *at most* one point from $P$, the only extra action would be to increment $N_{ab}$ by one in the second case as well.

Our proposed method readily generalizes further to the cases when we aim to compute the number of triangles in $P$, which contain precisely or at most $k$ points from $P$, where $0 \leq k \leq n - 3$. In either case, we shall need to maintain the $k + 1$ largest slopes $s_1^*$, $\ldots$, $s_k^*$, $s_{k+1}^*$ of the lines through $b$ and the points from $P'$ processed so far. When handling the next point $p_j$, its corresponding slope $s_j$ is located in the set of the slopes stored, and those smaller than $s_j$ (if any) are appropriately updated. The counter $N_{ab}$ is incremented by one if $s_k^* > s_j > s_{k+1}^*$ in the former case, and if $s_j > s_{k+1}^*$ in the latter case. If the $k + 1$ slopes are stored in a binary balanced search tree (or any other data structure allowing for insertions and deletions in logarithmic time), then the time spent on examining each candidate for $c$, for a given ordered pair $(a, b)$, will increase by factor $\log k$.

We conclude this section with the below theorem.

**Theorem 2** *Let $P$ be a set of $n$ points in general position in the plane. The number of triangles in $P$, which contain inside precisely or at most $k$ points from $P$, can be computed in $O(n^3 \log k)$ time using $O(n)$ space.*

## 3 Quadrilaterals

In this section, we shall first discuss the problems of counting empty quadrilaterals with vertices in $P$, and then the task of counting those which contain exactly or at most $k$ points in their interior.

Consider an ordered pair $(a, b)$ of points from $P$; without loss of generality, assume that $ab$ is horizontal, and $b$ lies strictly to the right of $a$. We start by counting empty quadrilaterals in $P$ with a diagonal $ab$ (Fig. 2). To this end, we apply the algorithm from Section 2 to determine the number $U_{ab}$ of empty triangles $\triangle abc$ with $c$ lying above $ab$ (i.e. with $a, b, c$ in ccw order), and the number $L_{ab}$ of those with $c$ lying below $ab$ (i.e. with $b, a, c$ in ccw order). The number $D_{ab}$ of empty quadrilaterals with a diagonal $ab$ can then be calculated as $U_{ab} \cdot L_{ab}$.

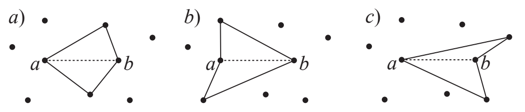


Figure 2: Empty quadrilaterals with a diagonal $ab$: a) convex; b) non-convex with a concave vertex at $a$; c) non-convex with a concave vertex at $b$.

However, if we process all the ordered pairs $(a, b)$ of points from $P$ in the above way and sum up the obtained numbers $D_{ab}$, then each convex empty quadrilateral in $P$ will be counted twice (once for either diagonal). Therefore, when processing a pair $(a, b)$, we shall also compute the number $NC_{ab}$ of non-convex quadrilaterals with a concave vertex at either $a$ or $b$ (Fig. 2b,c). Then the number of convex quadrilaterals with a diagonal $ab$ encountered when handling $(a, b)$ will be given by $C_{ab} = D_{ab} - NC_{ab}$, and we shall calculate the total number of empty quadrilaterals in $P$ as $T_{ab} = \sum_{(a,b)\in P} D_{ab} - \frac{1}{2} \cdot \sum_{(a,b)\in P} C_{ab}$.

In order to calculate $NC_{ab}$, we separately compute the numbers $NC^a_{ab}$ and $NC^b_{ab}$ of non-convex quadrilaterals with a diagonal $ab$ and a concave vertex at $a$ and $b$, respectively. To retrieve $NC^a_{ab}$, we first run a slightly modified algorithm from Section 2 for the pairs $(a, b)$ and $(b, a)$, and label all the points $c$ and $c'$ from $P$ lying above and below $ab$, respectively, such that the triangle $\triangle abc$ and $\triangle abc'$ is empty, respectively. Next, we sweep the plane with a directed line $\bar{l}$ rotating around $a$, which is initially horizontal and directed to the right. Thereby we maintain the number $N_{\bar{l}}$ of labeled points below $ab$ that lie to the left of $\bar{l}$. Initially, $NC^a_{ab}$ is set to zero, and each time $\bar{l}$ passes through a labeled point lying above $ab$, it is incremented by the current value of $N_{\bar{l}}$. The process terminates when $\bar{l}$ becomes again horizontal, but directed to the left. Then $NC^b_{ab}$ is retrieved analogously, and finally, $NC_{ab}$ is obtained as $NC^a_{ab} + NC^b_{ab}$.

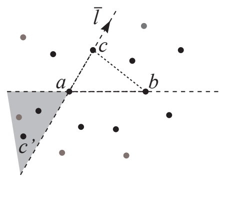


Figure 3: The labeled points (which together with $a$ and $b$ define an empty triangle) are shown black; other points from $P_{ab}$ are shown gray. The directed line $\bar{l}$ rotates ccw around $a$. For the current point $c$, the black points lying in the gray area, and only those, can represent the fourth vertex $c'$ of an empty quadrilateral $ac'bc$ with a concave vertex at $a$.

The above reasoning leads to the following theorem.

**Theorem 3** *Let $P$ be a set of $n$ points in general position in the plane. The number of empty quadrilaterals in $P$ can be computed in $O(n^3)$ time using $O(n)$ space. The number of convex and non-convex empty quadrilaterals in $P$ can be retrieved within the same complexity bounds.*

In order to count the quadrilaterals that contain precisely or at most $k$ points from $P$, the same framework can be applied. Since each empty quadrilateral is detected by the above approach as a pair of non-overlapping empty triangles $\triangle abc$ and $\triangle abc'$ sharing a common edge $ab$, our proposed modification will comprise detecting such pairs of triangles $\triangle abc$ and $\triangle abc'$ containing precisely $m$ and precisely or at most $k - m$ points from $P$ in their interior, respectively, where $0 \leq m \leq k$, using the modified algorithm outlined at the end of Section 2. In particular, each time a single run of the algorithm counting empty triangles was required, we shall now need $k+1$ calls to a modified algorithm with an input parameter $m = 0, 1, \ldots, k$.

**Theorem 4** *Let $P$ be a set of $n$ points in general position in the plane. The number of quadrilaterals in $P$, which contain inside precisely or at most $k$ points from $P$, can be computed in $O(k\cdot n^3 \log k)$ time using $O(n)$ space. The number of convex and non-convex such quadrilaterals can be retrieved within the same complexity bounds.*

## 4 Pentagons

We start by making the following simple observation in respect of the structure of pentagons: any triangulation of a pentagon comprises two diagonals incident to a common vertex. Consequently, we suggest the following approach to counting pentagons in a point set $P$: consider all the possible pairs of segments defined by $P$, which share a common endpoint, and count the number of pentagons in $P$, for which they represent two diagonals.

For any three non-collinear points $x, y, z$ in the plane, let $h^z_{xy}$ denote the open half-plane bounded by the line $(xy)$, which does not contain $z$.

Consider an ordered triple $(a, b, b')$ of points from $P$, such that $a, b, b'$ are in ccw order in the boundary of $\triangle abb'$. We start by counting empty pentagons in $P$ with diagonals $ab$ and $ab'$. Any such pentagon can be obtained as a union of the triangle $\triangle abb'$ and two other triangles $\triangle abc \in h^{b'}_{ab}$ and $\triangle ab'c' \in h^{b}_{ab'}$. In particular, if $\triangle abb'$ is non-empty, we conclude immediately that the answer is zero. Otherwise, we need to compute the number of pairs of non-overlapping empty triangles $\triangle abc$ and $\triangle ab'c'$ (Fig. 4a,b).

We shall first select $c$, and then $c'$. Let us say that $c \in P \cap h^{b'}_{ab}$ is valid if $\triangle abc$ is empty, and $c' \in P \cap h^{b}_{ab'}$ is valid if $\triangle ab'c'$ is empty and $\triangle abc \cap \triangle ab'c' = a$.

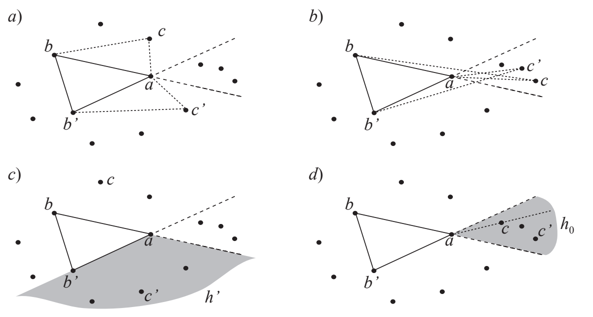


Figure 4: The segments $ab$ and $ab'$ are supposed to be diagonals of an empty pentagon $acbb'c'$. The triangle $\triangle abb'$ is empty. a) A valid choice of $c$ and $c'$. b) An invalid choice of $c'$, for the given $c$. c) For any valid $c$, any $c' \in P \cap h'$ is valid. d) For any valid $c \in P \cap h_0$, any $c' \in P \cap h_0$ lying below the line $(ac)$ is valid.

Let $h' = h^b_{ab'} \cap h^c_{ab}$, and let $P' = P \cap h'$. Observe that for any valid $c$, any point $c' \in P'$, such that $\triangle ab'c'$ is empty, represents a valid choice (Fig. 4c). Thus, to obtain the number $N'$ of valid choices for both $c$ and $c'$, such that $c' \in P'$, it is sufficient to compute the number $N'_c$ of empty triangles $\triangle abc$ with $a, b, c$ in cw order, and $N'_{c'}$ – that of empty triangles $\triangle ab'c'$ with $a, b', c'$ in ccw order and $c' \in P'$. The former task is accomplished by reflecting the coordinate system and applying the algorithm from Section 2, and for the latter task, this algorithm is slightly modified (namely, the counter is incremented only when the points lying in the area of interest are processed). Then we calculate $N' = N'_c \cdot N'_{c'}$.

Similarly, for any valid $c \in (P \cap h^{b'}_{ab}) \setminus (P \cap h^b_{ab'})$, any point $c' \in P \cap h^b_{ab'}$ such that $\triangle ab'c'$ is empty, represents a valid choice. Thus, the entire number of valid choices for both $c$ and $c'$ in this case can be computed analogously. Yet to avoid duplicate counting, we shall need to restrict our attention to the candidate points $c'$ lying inside the region $h_0 = h^{b'}_{ab} \cap h^b_{ab'}$.

If neither case holds, both $c$ and $c'$ fall inside the region $h_0$ (Fig. 4d). Let $P_0 = P \cap h_0$; then for any valid $c \in P_0$, any point $c' \in P_0$ falling in $h^b_{ac}$, and such that $\triangle ab'c'$ is empty, represents a valid choice. The number of valid choices for both $c$ and $c'$ can be retrieved in a way similar to the one used to compute $NC^a_{ab}$ in Section 3, again applying a slightly modified algorithm from Section 2 and plane sweep.

To summarize, for each ordered triple $(a, b, b')$ of points from $P$, such that $a, b, b'$ are in ccw order in the boundary of the triangle $\triangle abb'$, we first test whether $\triangle abb'$ is empty, and if so, apply the above algorithm. Either task is accomplished in $O(n)$ time using $O(n)$ space, for each of $O(n^3)$ such triples, assuming all the sorted orders of points were retrieved at the preprocessing stage.

Note that if all the ordered triples $(a, b, b')$ of points from $P$ are processed in a straightforward way according to the above scheme, then some of the empty pentagons may be encountered more than once. However, a more thorough analysis of mutual location of $c$ and $c'$ allows to properly handle such situations, and in addition, to count separately convex and non-convex pentagons within the same time and space bounds.

**Theorem 5** *Let $P$ be a set of $n$ points in general position in the plane. The number of empty pentagons in $P$ can be computed in $O(n^4)$ time using $O(n)$ space. The number of convex and non-convex empty pentagons in $P$ can be retrieved within the same complexity bounds.*

To count the pentagons that contain precisely or at most $k$ points from $P$, we need to modify the proposed framework in a similar way as in Section 3, and to count the exact number of points lying inside $\triangle abb'$ instead of testing whether it is empty.

**Theorem 6** *Let $P$ be a set of $n$ points in general position in the plane. The number of pentagons in $P$, which contain inside precisely or at most $k$ points from $P$, can be computed in $O(k \cdot n^4 \log k)$ time using $O(n)$ space. The number of convex and non-convex such pentagons can be retrieved within the same complexity bounds.*

## 5 Conclusion

In this work, we have proposed novel and simple algorithms for counting empty triangles, quadrilaterals and pentagons in a set $P$ of $n$ points, as well as those containing precisely or at most $k$ points from $P$. For quadrilaterals and pentagons, our algorithms allow to determine the number of convex and non-convex polygons in either case. Analogous problems about $k$-gons for $k \geq 6$ remain for future research.

## References

[1] H. Edelsbrunner and L. Guibas, Topologically sweeping an arrangement. *J. Comput. and System Sci.*, 38(1):165–194, 1989.

[2] S. Kopeliovich and K. Vyatkina, On counting and analyzing empty pseudo-triangles in a point set. *Proc. 12th Int. Conf. on Comput. Sci. and its Appl., LNCS* 7333:280–291, Springer, 2012.

[3] J.S.B. Mitchell, G. Rote, G. Sundaram, and G. Woeginger, Counting convex polygons in planar point sets. *Inf. Proc. Lett.*, 56(1):45–49, 1995.

[4] G. Rote, G. Woeginger, B. Zhu, and Zh. Wang, Counting convex $k$-gons in planar point sets. *Inf. Proc. Lett.*, 38(3):149–151, 1991.

[5] G. Rote and G. Woeginger, Counting convex $k$-gons in planar point sets. *Inf. Proc. Lett.*, 41(4):191–194, 1992.

# Randomized incremental construction of the Hausdorff Voronoi diagram of non-crossing clusters[*]

Panagiotis Cheilaris[†] Elena Khramtcova[†] Evanthia Papadopoulou[†]

## Abstract

We give a randomized incremental construction for the Hausdorff Voronoi diagram of non-crossing clusters of points. Our best complexity algorithm takes expected time $O(n \log^2 n (\log \log n)^2)$ and worst-case space $O(n)$, improving upon previous results. A simpler-to-implement algorithm, based on the Voronoi hierarchy, is also given, which takes expected time $O(n \log^3 n)$ and expected space $O(n)$. To achieve this, we augment the Voronoi hierarchy with the ability to efficiently handle non-standard characteristics of generalized Voronoi diagrams, such as sites of non-constant complexity, sites that are not enclosed in their Voronoi regions, and empty Voronoi regions.

## 1 Introduction

Given a family $F$ of point clusters in $\mathbb{R}^2$, the *Hausdorff Voronoi diagram* of $F$ is a subdivision of the plane into regions such that the *Hausdorff Voronoi region* of a cluster $P \in F$ is the locus of points closer to $P$ than to any other cluster in $F$. The closeness of a point $t \in \mathbb{R}^2$ to a cluster $P$ is measured by the *farthest distance* $d_f(t, P) = \max_{p \in P} d(t, p)$, where $d(t, p)$ is the Euclidean distance between $t$ and $p$. The motivation for investigating Hausdorff Voronoi diagrams comes from its use in efficiently estimating the sensitivity of a Very Large Scale Integration (VLSI) design to random defects during manufacturing [11].

Clusters $P$ and $Q$ are called *non-crossing* if the convex hull of $P \cup Q$ admits at most two supporting segments with one endpoint in $P$ and one endpoint in $Q$ (see Figure 1). We assume that clusters are pairwise non-crossing, unless stated otherwise.

The Hausdorff Voronoi diagram of non-crossing clusters has size $O(n)$ [12], where $n$ is the total number of points in all clusters. It can be constructed in time $O(n^2)$ [8], or in time $O((n + K) \log n)$ [11,12], however, $K$ can be superlinear.[1] It is an instance of abstract Voronoi diagrams [10], thus it can also be constructed in expected time $O(bn \log n)$, where $b = O(n)$ is the time to construct the bisector between two clusters [1]. A more recent algorithm gives $O(n \log^4 n)$ time and $O(n \log^2 n)$ space complexity [6].

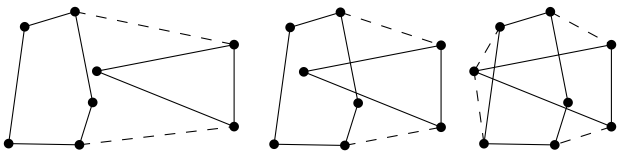


Figure 1: disjoint, non-crossing, crossing clusters.

In this work, we build the Hausdorff Voronoi diagram using a *randomized incremental* approach. That is, sites (clusters) are inserted one by one in random order and the diagram is updated at every insertion [5]. The bottleneck in this approach is to identify fast a point $t \in \mathbb{R}^2$ that will lie in the region of the new site. This is difficult for the Hausdorff Voronoi diagram because: (a) the region of the new cluster (site) might not contain any of its points, (b) the insertion of the new cluster can make an existing region empty, and (c) clusters have non-constant size, and thus the computation of a bisector or the answering of an *in-circle test* require non-constant time. To overcome these issues we exploit properties of Hausdorff Voronoi diagram and we maintain a *dynamic point location* data structure, which is also used to perform simple parametric search queries.[2] Our approach is modular as it can use any dynamic point location data structure. If we use the data structure by Baumgarten *et al.* [2], then we get an algorithm which takes expected time $O(n \log^2 n (\log \log n)^2)$ and uses linear space. Alternatively, if we augment the *Voronoi hierarchy* [9] with the ability to efficiently handle the difficulties (a) to (c), we obtain a more practical algorithm which takes expected $O(n \log^3 n)$ time and $O(n)$ space.

The augmentation of the Voronoi hierarchy may be of interest for incremental construction of other generalized Voronoi diagrams.

[*]Supported in part by the Swiss National Science Foundation grant 134355, under the auspices of the ESF EUROCORES program EuroGIGA/VORONOI.

[†]Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

[1]$K$ is the number of pairs of clusters such that one cluster is contained in a specially defined enclosing circle of the other, e.g., the minimum enclosing circle [12].

[2]Construction algorithms for the Hausdorff Voronoi diagram [6] and for the farthest-polygon Voronoi diagram [4] also resort to parametric search.

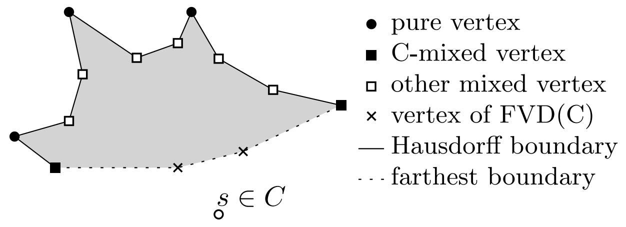


Figure 2: $\mathrm{hreg}(s), s \in C$.

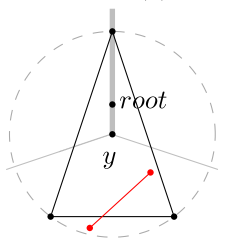


Figure 3: 3-point cluster $C$ (black), and 2-point cluster $P$ (red) limiting w.r.t. $y \in \mathrm{fskel}(P)$.

## 2 Definitions and Structural properties

Let $F = \{C_1, \ldots, C_m\}$ be a family of non-crossing clusters of points such that no two clusters have a common point. For simplicity we assume that no four points lie on the same circle. Let $\mathrm{conv}\, P$ denote the convex hull of cluster $P$ and $\mathrm{CH}(P)$ denote the sequence of points of $P$ on the boundary of $\mathrm{conv}\, P$, in counterclockwise order. For $s \in C$, the *farthest region* of $s$ in the *farthest Voronoi diagram* (FVD) of $C$ is:

$$\mathrm{freg}_C(s) = \{p \mid \forall s' \neq s \colon d(p,s) > d(p,s')\}.$$

The graph structure of the FVD of $C$, $|C| > 1$, forms a tree, called the *farthest skeleton* of $C$, $\mathrm{fskel}(C)$. If $|C| = 1$ then $\mathrm{fskel}(C)$ is $C$ itself.

The *Hausdorff region* of a cluster $C \in F$ and a point $s \in C$ are defined as

$$\mathrm{hreg}_F(C) = \{p \mid \forall C' \neq C \colon d_{\mathrm{f}}(p,C) < d_{\mathrm{f}}(p,C')\};$$
$$\mathrm{hreg}_F(s) = \mathrm{hreg}_F(C) \cap \mathrm{freg}_C(s).$$

The boundary of the Hausdorff region of a point $s \in C$ consists of two chains: (1) the *farthest boundary* of $s$, which is the portion of $\mathrm{fskel}(C)$ in $\mathrm{hreg}_F(C)$, i.e., $\mathrm{bd}\,\mathrm{hreg}_F(s) \cap \mathrm{bd}\,\mathrm{freg}_C(s)$; (2) the *Hausdorff boundary* of $s$, i.e., $\mathrm{bd}\,\mathrm{hreg}_F(s) \cap \mathrm{bd}\,\mathrm{hreg}_F(C)$. Neither chain can be empty if $\mathrm{hreg}_F(C) \neq \emptyset$ and $|C| > 1$.

As shown in [12], there are three types of vertices on the boundary of a Hausdorff Voronoi region $\mathrm{hreg}_F(s)$: (1) Standard Voronoi vertices, which are equidistant from $C$ and two other clusters, referred to as *pure* vertices (using the terminology of [4]). (2) *Mixed* vertices, which are equidistant from $C$ and one other cluster. The mixed vertices which are equidistant to two points of $C$ and one point of another cluster are called *C-mixed* vertices; there are exactly two of them on the boundary of $\mathrm{hreg}_F(s)$. (3) Vertices of $\mathrm{fskel}(C)$ on the farthest boundary of $s$. See Figure 2.

Useful properties of the Hausdorff Voronoi diagram are summarized in Proposition 1. We need some definitions.

Consider a cluster $C$. Line segment $\overline{ab}$ is a *chord* of $C$ if $a, b \in \mathrm{CH}(C)$. Assign a root in $\mathrm{fskel}(C)$ arbitrarily and denote this rooted tree by $T(C)$. Let $y$ be any point of $\mathrm{fskel}(C)$, and $\overline{cc^*}$ be a chord of $C$ such that $y$ lies on the bisector between $c$ and $c^*$. Let $D_y$ be the closed disk centered at $y$ with radius $d_{\mathrm{f}}(y,C)$. Then, $C \subset D_y$. Point $y$ subdivides $\mathrm{fskel}(C)$ into two parts. Denote the part containing the descendants of $y$ in $T(C)$ by $T(y)$, and its complement by $T^{\sim}(y)$. Chord $\overline{cc^*}$ subdivides $D_y$ into $D_y^{\mathrm{r}}$ and $D_y^{\mathrm{f}}$, where $D_y^{\mathrm{r}}$ (resp., $D_y^{\mathrm{f}}$) is the *rear (forward) part*, enclosing the portion of $\mathrm{conv}\, C$ inducing $T(y)$ ($T^{\sim}(y)$). See Figure 3.

**Definition 1** *Cluster $P$ is* limiting *with respect to point $y \in \mathrm{fskel}(C)$, if disk $D_y$ contains $P$. Cluster $P$ is called* forward limiting *if $P \subset D_y^{\mathrm{f}} \cup \mathrm{conv}\, C$ or* rear limiting *if $P \subset D_y^{\mathrm{r}} \cup \mathrm{conv}\, C$.*

The following properties are derived directly from [12].

**Proposition 1** *Let $F$ be a family of non-crossing clusters and $C, P, Q \in F$. Then:*

*(i) If $\mathrm{hreg}_F(C) \neq \emptyset$, then $\mathrm{hreg}_F(C) \cap \mathrm{fskel}(C)$ is non-empty and connected.*

*(ii) Let $y$ be a point of $\mathrm{fskel}(C)$ such that $y$ is closer to cluster $P$ than to $C$. If $P$ is forward (resp. rear) limiting with respect to $y$ then the entire $T(y)$ (resp. $T^{\sim}(y)$) is closer to $P$ than to $C$.*

*(iii) Let $\overline{uv}$ be an edge of $\mathrm{fskel}(C)$. If both $u$ and $v$ are closer to $P$ than to $C$ then $\mathrm{hreg}_F(C)$ does not intersect $\overline{uv}$.*

*(iv) Region $\mathrm{hreg}_F(C) = \emptyset$ if and only if either there is a cluster $P \subset \mathrm{conv}\, C$, or there exists a pair of clusters $\{P, Q\}$ such that $P$ is rear limiting and $Q$ is forward limiting with respect to the same point $y \in \mathrm{fskel}(P)$. Pair $\{P, Q\}$ is called a* killing pair *for $C$.*

## 3 General incremental construction algorithm

Let $C_1, C_2, \ldots, C_m$ be a fixed order of clusters. Let $F_i$ denote the family of the first $i$ clusters according to this order. The incremental approach constructs successively the Hausdorff Voronoi diagram of $F_1, F_2, \ldots, F_m = F$. For each cluster $C_i$, we have the farthest Voronoi diagram, $\mathrm{FVD}(C_i)$, and a (static) point location data structure on $\mathrm{FVD}(C_i)$.

We construct $\mathrm{HVD}(F_{i+1})$ from $\mathrm{HVD}(F_i)$ by inserting $C_{i+1}$. We first find a point $t$, which is closer to $C_{i+1}$ than to any cluster in $F_i$, or if there is no such point, we conclude that $\mathrm{hreg}_{F_{i+1}}(C_{i+1}) = \emptyset$ and stop. By Proposition 1(i), it is sufficient to search for

$t$ just in $\mathrm{fskel}(C_{i+1})$. Then, we trace the region of $C_{i+1}$ around $t$ and update the diagram.

We first consider vertices of $\mathrm{fskel}(C_{i+1})$. For each such vertex $w$:

- Find the nearest to $w$ cluster $C^w \in F_i$ using point location in $\mathrm{HVD}(F_i)$,
- if $d_{\mathrm{f}}(w, C_{i+1}) < d_{\mathrm{f}}(w, C^w)$, then $t = w$; exit the procedure. Else, if possible, eliminate from further consideration a subtree of $\mathrm{fskel}(C_{i+1})$ incident to $w$; see Proposition 1(ii).

If no vertex is found, then consider any remaining edges of $\mathrm{fskel}(C_{i+1})$ as $\mathrm{hreg}_{F_{i+1}}(C_{i+1})$ may intersect the interior of at most one edge of $\mathrm{fskel}(C_{i+1})$; see Proposition 1(i).

Edge $uv$ of $\mathrm{fskel}(C_{i+1})$ is called a *candidate* edge if $d_{\mathrm{f}}(u, C_{i+1}) < d_{\mathrm{f}}(u, C^v)$ and $d_{\mathrm{f}}(v, C_{i+1}) < d_{\mathrm{f}}(v, C^u)$. By Proposition 1(iii), it is sufficient to only check a candidate edge and there can be at most one such edge for $C_{i+1}$. Thus, for a candidate edge (if any) we perform parametric search to decide whether $\mathrm{hreg}_{F_{i+1}}(C_{i+1})$ is empty or not, and still to find a point $t$ in this region in the latter case.

**Lemma 2** *Suppose clusters are inserted in a uniformly random order. Then, the expected time complexity of the randomized algorithm is*

$$O(n \log n) + O(n)(t_{\mathrm{q}}(n) + t_{\mathrm{i}}(n) + t_{\mathrm{d}}(n)) + m \cdot t_{\mathrm{p}}(n),$$

*where $t_{\mathrm{q}}(n)$, $t_{\mathrm{p}}(n)$, $t_{\mathrm{i}}(n)$, $t_{\mathrm{d}}(n)$ are the times for a query, a parametric search, an insertion, and a deletion*[3] *in a point location data structure for a diagram of complexity $O(n)$, respectively.*

**Proof.** (Sketch) The expected total number of insertions and deletions is $O(n)$.[4] The total time for the construction of farthest Voronoi diagrams for all clusters is $O(n \log n)$. For each cluster $C_i$, we perform $O(|C_i|)$ point location queries and at most one parametric search. Thus, in total we perform $O(n)$ point locations and at most $m$ parametric searches. $\square$

We can use the dynamic point location data structure of Baumgarten *et al.* [2] with $t_{\mathrm{q}}(n) = O(\log n \log \log n)$, $t_{\mathrm{i}}(n) = O(\log n \log \log n)$, and $t_{\mathrm{d}}(n) = O(\log^2 n)$. Parametric search can be performed as a simulation of a point location query for the unknown point $t$ in time $t_{\mathrm{p}}(n) = (t_{\mathrm{q}}(n))^2$ (see also [4]). As a result:

**Theorem 1** *There is a randomized algorithm that constructs the Hausdorff Voronoi diagram of a family of non-crossing clusters in linear space and in expected time $O(n \log^2 n (\log \log n)^2)$.*

[3]The time for insertion and deletion can be amortized.

[4]It can be shown with the Clarkson-Shor technique [5].

## 4 Augmenting the Voronoi hierarchy

The Voronoi hierarchy [9] is a simple randomized point location data structure for Voronoi diagrams inspired from the Delaunay hierarchy [7]. For a family $F$ of general sites, each level $\ell$ of the hierarchy corresponds to a subset $F^{(\ell)}$ of $F$ and stores the Voronoi diagram of $F^{(\ell)}$. Level 0 corresponds to $F$. A *Voronoi hierarchy* of *height* $k$ is then: $F = F^{(0)} \supseteq F^{(1)} \supseteq \ldots \supseteq F^{(k)}$. For all $\ell \in \{1, \ldots, k\}$, $F^{(\ell)}$ is a random sample of $F^{(\ell-1)}$ according to a Bernoulli distribution with parameter $\beta \in (0, 1)$. The expected height of the hierarchy for a family of $m$ sites is $O(\log m)$. *Point location* in the Voronoi hierarchy for a query point $q$ works as follows. Starting from the topmost level $k$, for each level $\ell$, find the site $S^\ell$ in $F^{(\ell)}$ which is the nearest to the query point $q$, by performing a *walk*. Each step of the walk reduces the distance to $q$ from the current site $S$ by moving to a site, neighboring to $S$. Walk at level $\ell - 1$ starts from $S^\ell$. The answer to the query is $S^0$.

For the Hausdorff Voronoi diagram several complications arise: (a) Sites are of non-constant complexity. (b) We need to perform parametric search i.e., a walk for an unknown point along a candidate edge. (c) Voronoi regions might be empty.

To address (a) we need the concept of an *active* point. Consider a cluster $C$ at level $\ell$, such that $\mathrm{hreg}_{F^{(\ell)}}(C) \neq \emptyset$. For brevity, let this region be denoted as $\mathrm{hreg}_F^{(\ell)}(C)$, and similarly for the regions of individual points.

**Definition 2** *Point $c \in C$ is* active *at level $\ell$, if $\mathrm{hreg}_F^{(\ell)}(c) \neq \emptyset$. The set $\hat{C}^{(\ell)}$ of all active points of $C$ at level $\ell$ is called the* active set *of $C$ at level $\ell$. For brevity, $d_{\mathrm{f}}^{(\ell)}(t, C) = d_{\mathrm{f}}(t, \hat{C}^{(\ell)})$.*

**Performing one step of the walk.** Let $C$ be the current cluster visited during the walk at level $\ell$, and $q$ be the query point. The next cluster $C'$ in the walk is determined as follows. Let $c$ be a point of $\hat{C}^{(\ell)}$ such that $q \in \mathrm{freg}_{\hat{C}^{(\ell)}}(c)$. Let $v_1, \ldots, v_j$ be a list of pure vertices on the Hausdorff boundary of $\mathrm{hreg}_F^{(\ell)}(c)$, in counterclockwise order, and let $C^0, \ldots, C^j, C^{j+1}$ be respective adjacent clusters. The rays $\overrightarrow{cv_1}, \ldots, \overrightarrow{cv_j}$ partition $\mathrm{freg}_{\hat{C}^{(\ell)}}(c)$ into $j+1$ unbounded regions. If $\overrightarrow{cq}$ is just after the ray $\overrightarrow{cv_i}$ or just before the ray $\overrightarrow{cv_{i+1}}$, then set $C' = C^i$.

In order to find $C'$ in $O(\log n)$ time, for each cluster $C$ at level $\ell$ such that $C \in F^{(\ell)}$ we store the binary trees containing: (1) the active set $\hat{C}^{(\ell)}$; (2) for each $c \in \hat{C}^{(\ell)}$, the list of all pure vertices adjacent to $\mathrm{hreg}_F^{(\ell)}(c)$ (see Figure 2).

**The parametric search.** For $\ell \in \{0, \ldots, k\}$, let $I^\ell$ be the interval of points on $uv$ which are closer to $C_{i+1}$

than to any cluster in $F_i^{(\ell)}$. (By convention, $I^{k+1} = uv$.) Then, $uv = I^{k+1} \supseteq I^k \supseteq I^{k-1} \supseteq \cdots \supseteq I^1 \supseteq I^0$. If $I^\ell \neq \emptyset$, we compute the leftmost endpoint $u^\ell$ of interval $I^\ell$, i.e., the endpoint which is closer to $u$. If $I^0 \neq \emptyset$, then $u^0$ is the answer to the query.

From the point $u^{\ell+1}$ and the cluster of $F_i^{(\ell+1)}$ closest to $u^{\ell+1}$, point $u^\ell$ is computed entirely at level $\ell$. We find a sequence of points $u^{\ell+1} = a_0, a_1, \ldots, a_r = u^\ell$. For each point $a_j$, we keep track of the cluster $C^{a_j}$ in $F_i^\ell$ which is the closest to $a_j$. We compute $a_{j+1}$ from $a_j$ as follows.

- If $d_f(a_j, C_{i+1}) \leq d_f(a_j, C^{a_j})$, we set $u^\ell = a_j$ and continue to the next level.
- Else, if $d_f(v, C^{a_j}) \leq d_f(v, C_{i+1})$, we stop and report that $\text{hreg}_{F_{i+1}}(C_{i+1}) = \emptyset$.
- Otherwise, we determine $a_{j+1}$ by a (standard) parametric search in $\text{FVD}(C^{a_j})$ with segment $a_j v$. Then, we perform a walk (at level $\ell$) from cluster $C^{a_j}$ to find the cluster $C^{a_{j+1}}$ closest to $a_{j+1}$. If $C^{a_{j+1}} = C^{a_j}$, we set $u^\ell = a_j$ and continue to the next level.

**Empty Voronoi regions.** When $C_{i+1}$ is inserted, an existing non-empty region of a cluster $P$ may become empty. If $P$ has an empty region at level $\ell$, but a non-empty region at level $\ell + 1$, then the point location for a query point $q \in \text{hreg}_{F_{i+1}}^{(\ell+1)}(P)$ will give an error. To fix the problem, we link cluster $P$ at level $\ell + 1$ to at most two other clusters at level $\ell$ (see Proposition 1(iv)), so that every point $q \in \mathbb{R}^2$ is strictly closer to either one of them than to $P$, as follows.

While inserting $C_{i+1}$ at level $\ell$, we keep track of the list $V$ of all the (deleted) $P$-mixed vertices.

At level $\ell + 1$, for each $P$-mixed vertex $v$, we check if $d_f(v, C_{i+1}) \geq d_f(v, P)$. If yes, we store the point $c \in C_{i+1}$ for which $d_f(v, C_{i+1}) = d(v, c)$.

1. If all $P$-mixed vertices are closer to $C_{i+1}$ than to $P$, we link $P$ only to $C_{i+1}$.
2. Else, we link $P$ to its killing pair $\{K, C_{i+1}\}$, such that $K \in F^{(l)}$; see Proposition 1(iv).

We identify cluster $K$ using the list $V$ and the point $c$. Each vertex $u \in V$ is equidistant from points $p_u$, $p_u^* \in P$, and $q_u \in Q$, for some $Q \in F_i^{(\ell)}$. We check whether $c$ and $q_u$ are on different sides of the chord $\overline{p_u p_u^*}$. If yes, then set $K = Q$ and stop.

The complexity is analyzed in the following.

**Lemma 3** *The expected length of the walk at level $\ell$ is constant.*

**Lemma 4** *A point location query and a parametric search query are answered in expected $O(\log^2 n)$ and $O(\log^3 n)$ time respectively.*

**Lemma 5** *Let $n$ be the sum of the sizes of all sites in a family of sites $F$. Assuming that the underlying type of Voronoi diagram for $F$ is of size $O(n)$, the expected size of the Voronoi hierarchy for $F$ is also $O(n)$.*

**Theorem 2** *The Hausdorff Voronoi diagram of non-crossing clusters can be constructed in $O(n \log^3 n)$ expected time and $O(n)$ expected space, using the Voronoi hierarchy.*

For details, see [3].

## References

[1] M. Abellanas, G. Hernandez, R. Klein, V. Neumann-Lara, and J. Urrutia. A combinatorial property of convex sets. *Discrete Comput. Geom.*, 17(3):307–318, 1997.

[2] H. Baumgarten, H. Jung, and K. Mehlhorn. Dynamic point location in general subdivisions. *J. Algorithms.*, 17(3):342–380, 1994.

[3] P. Cheilaris, E. Khramtcova, and E. Papadopoulou. Randomized incremental construction of the Hausdorff Voronoi diagram of non-crossing clusters. Technical Report 2012/03, University of Lugano, 2012.

[4] O. Cheong, H. Everett, M. Glisse, J. Gudmundsson, S. Hornus, S. Lazard, M. Lee, and H.-S. Na. Farthest-polygon Voronoi diagrams. *Comput. Geom.*, 44(4):234–247, 2011.

[5] K. Clarkson and P. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[6] F. Dehne, A. Maheshwari, and R. Taylor. A coarse grained parallel algorithm for Hausdorff Voronoi diagrams. In *Proc. 35th Int. Conf. on Parallel Processing (ICPP)*, pages 497–504, 2006.

[7] O. Devillers. The Delaunay Hierarchy. *Int. J. Foundations Comput. Sci.*, 13:163–180, 2002.

[8] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, 4:311–336, 1989.

[9] M. Karavelas and M. Yvinec. The Voronoi diagram of convex objects in the plane. Technical report RR-5023, INRIA, 2003.

[10] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom.*, 3(3):157–184, 1993.

[11] E. Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica.*, 40(2):63–82, 2004.

[12] E. Papadopoulou and D. T. Lee. The Hausdorff Voronoi diagram of polygonal objects: a divide and conquer approach. *Int. J. Comput. Geom. Appl.*, 14(6):421–452, 2004.

# Geometry and Images

Marcus Magnor*

**Abstract**

Images and geometry are intimately interconnected. The appearance of objects is determined by their shape, and the geometry of things and scenes is what our visual brain has learned to deduce from images to make sense of the world. From artistic drawings to 3D movies, various aspects of geometry play a role in the formation, perception and comprehension of images. In my talk, I will present various examples of how images and geometry are interrelated.

*Department of Computer Science, TU Braunschweig, Germany. m.magnor@tu-bs.de

# Balanced partitions of 3-colored geometric sets in the plane

Sergey Bereg* Ferran Hurtado† Mikio Kano‡ Matias Korman† Dolores Lara† Carlos Seara† Rodrigo I. Silveira† Jorge Urrutia§ Kevin Verbeek¶

**Abstract**

In this work we study several problems regarding balanced bipartitions of 3-colored sets of points, and of 3-colored sets of lines. We say that a bipartition of a colored set is *balanced* if each of its two parts contains the same quantity of elements of each color. First we consider sets of lines and study the existence of segments that intersect the arrangement in a balanced set; we also study the dual problem, that is, we consider point sets in the plane and study bipartitions induced by double wedges. Then we consider point sets on a closed Jordan curve and study bipartitions induced by arcs. Last, we consider point sets in the plane lattice and study bipartitions induced by L-lines.

## 1 Introduction

Let $S$ be a finite set of geometric objects distributed into classes or *colors*. A subset $S_1 \subseteq S$ is said to be *balanced* if $S_1$ contains the same number of elements of $S$ from each of the colors.

Naturally, if $S$ is balanced, the complement of a balanced subset of $S$ is also balanced, hence we talk of a *balanced bipartition* of $S$. When the point set $S$ is on the plane, and the balanced partition is defined by a geometric object $\zeta$ splitting the plane into two regions, we say that $\zeta$ is *balanced*.

In this paper we study problems on balanced bipartitions of 3-colored sets of points and lines in the plane. It is easy to construct balanced 3-colored point sets for which the only balanced partitioning line is the trivial one containing the whole pointset on one side. For example, consider an equilateral triangle $p_1p_2p_3$ and replace every vertex $p_i$ by a very small disk $D_i$ so that no line can intersect the three disks; place $n$ red, $n$ green, and $n$ blue points inside the disks $D_1$, $D_2$ and $D_3$, respectively. It is obvious that for this configuration no balanced line exists. On the other hand, for every 3-colored set $S$ of points there is a conic that simultaneously bisects the three colors: take the plane to be $z = 0$ in $\mathbb{R}^3$, lift the points vertically to the unit paraboloid $P$, use the 3-dimensional ham-sandwich theorem for splitting evenly the lifted point set with a plane $\Pi$, and use the projection of $P \cap \Pi$ as halving conic in $z = 0$. Instead of changing the partitioning object, one may impose some constraints to the point set. For example, Bereg and Kano have recently proved [2] that if all vertices of the convex hull of $S$ have the same color, then there exists a line that determines a halfplane containing exactly $k$ points of each color, with $0 < k < n$. This result was recently extended to sets of points in higher dimensions, by Akopyan and Karasev [1].

In this work we present the following results: (a) In Section 2 we prove that for every 3-colored arrangement of lines there exists a segment that intersects exactly one line of each color. Moreover, if there are $2m$ lines of each color, there is a segment intersecting $m$ lines of each color. (b) In Section 3 we show that given $n$ red, $n$ green and $n$ blue points on any closed Jordan curve $\gamma$, for every integer $0 \leq k \leq n$ there is a pair of disjoint intervals on $\gamma$ whose union contains exactly $k$ points of each color. (c) Given a set $S$ of $n$ red, $n$ green and $n$ blue points in the integer lattice, whose orthogonal convex hull is monochromatic, there exist one vertical and one horizontal ray with common apex, whose union splits the plane into two balanced regions; this is presented in Section 4. Due to lack of space, we omit most proofs from this extended abstract.

## 2 Balancing line arrangements

### 2.1 Cells in colored arrangements

Let $L$ be a set of lines in the plane, partitioned into three sets $R$, $G$, and $B$. We refer to the elements of $R$, $G$, and $B$ as red, green, and blue, respectively. Let $\mathcal{A}(L)$ be the arrangement induced by the set $L$. We assume that $\mathcal{A}(L)$ is *simple*, that is, there are no parallel lines and no more than two lines intersect at one point. In this section we prove that there always exists a 3-chromatic face in $\mathcal{A}(L)$, that is, a face that has at least one side of each color. We also extend this result to higher dimensions.

*University of Texas at Dallas. besp@utdallas.edu.

†Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, {ferran.hurtado, matias.korman, maria.dolores.lara, carlos.seara, rodrigo.silveira}@upc.edu

‡Ibaraki University, kano@mx.ibaraki.ac.jp.

§Instituto de Matemáticas, UNAM, México. urrutia@matem.unam.mx.

¶University of California, Santa Barbara. kverbeek@cs.ucsb.edu.

Consider a face $f$ of the 2-dimensional arrangement $\mathcal{A}(L)$. Consider the dual graph of $f$; we obtain a dual face $\hat{f}$ that contains a vertex for every bounding line of $f$, and contains an edge between two vertices of $\hat{f}$ if the intersection of the corresponding lines is part of the boundary of $f$ (see Figure 1(a)). Let $C$ be a simple cycle of vertices where each vertex is colored either red, green, or blue. Let $n_r(C), n_g(C)$, and $n_b(C)$ be the number of red, green, and blue vertices of $C$, respectively. We simply write $n_r$, $n_g$, and $n_b$ if $C$ is clear from the context. The *type* of an edge of $C$ is the multiset of the colors of its vertices. Let $n_{rr}$, $n_{gg}$, $n_{bb}$, $n_{rg}$, $n_{rb}$, and $n_{gb}$ be the number of edges of the corresponding type. Note that, if $f$ is bounded, then $\hat{f}$ is a simple cycle, where each vertex is colored either red, green, or blue. We say a bounded face $f$ is *complete* if $n_{rg} \equiv n_{rb} \equiv n_{gb} \equiv 1 \pmod 2$ holds for $\hat{f}$.

**Lemma 1** *Consider a simple cycle $C$, where each vertex is colored either red, green, or blue. Then $n_{rg} \equiv n_{rb} \equiv n_{gb} \pmod 2$.*

**Proof.** The result follows from double counting. For $n_r$ we get the equation $2n_r = 2n_{rr}+n_{rg}+n_{rb}$. This directly implies that $n_{rg} \equiv n_{rb} \pmod 2$. We can do the same for $n_g$ and $n_b$ to obtain the claimed result. □

**Theorem 2** *Let $L$ be a set of lines in $\mathbb{R}^2$ colored with 3 colors so that each color appears at least once, and the arrangement $\mathcal{A}(L)$ induced by $L$ is simple. There always exists a complete face in $\mathcal{A}(L)$.*

**Proof.** The result clearly holds if $|R| = |G| = |B| = 1$. For the general case, we start with one line of each color, and then incrementally add the remaining lines, maintaining a complete face $f$ at all times. Without loss of generality, assume that a red line $\ell$ is inserted into $\mathcal{A}(L)$. If $\ell$ does not cross $f$, we keep $f$. Otherwise, $f$ is split into two faces $f_1$ and $f_2$ (see Figure 1(b)). Similarly, $\hat{f}$ is split into $\hat{f}_1$ and $\hat{f}_2$ (with the addition of one red vertex, see Figure 1(c)). Because $\ell$ is red, the number $n_{gb}$ of green-blue edges does not change, that is, $n_{gb}(\hat{f}) = n_{gb}(\hat{f}_1)+n_{gb}(\hat{f}_2)$. This implies that either $n_{gb}(\hat{f}_1)$ or $n_{gb}(\hat{f}_2)$ is odd. By Lemma 1 it follows that either $f_1$ or $f_2$ is complete. □

Using the standard point-line duality in the plane, we obtain the following result:

**Corollary 1** *Let $L$ be a set of lines in $\mathbb{R}^2$ colored with 3 colors so that each color appears at least once, and the arrangement $\mathcal{A}(L)$ induced by $L$ is simple. There always exists a segment intersecting exactly one line of each color of $\mathcal{A}(L)$.*

We can extend the previous result to higher dimensions: every $(d+1)$-chromatic arrangement of hyperplanes in the $d$-dimensional space, contains a $(d+1)$-chromatic face. The result is sharp with respect to the number of colors. To extend the proof, we use similar techniques: first consider the dual of a cell in the arrangement as a triangulation of the $(d-1)$-dimensional sphere; then define types of faces of that triangulation based on the colors of their incident vertices, and prove an analogue of Lemma 1. The definition of *complete* cells is generalized accordingly, maintaining the property that a complete cell is $(d+1)$-chromatic. Based on that, we prove the following.

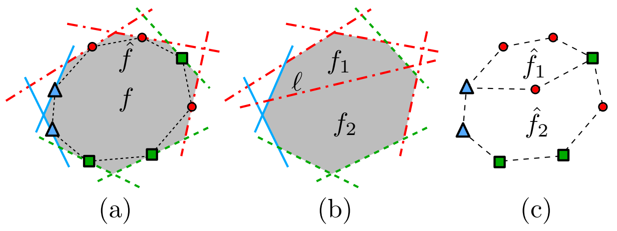


Figure 1: (a) A (complete) face $f$ and its dual $\hat{f}$ (dotted). (b) Face $f$ is split into $f_1$ and $f_2$. (c) The dual $\hat{f}$ is split into $\hat{f}_1$ and $\hat{f}_2$.

**Theorem 3** *Let $H$ be a set of hyperplanes colored with $d+1$ colors so that each color appears at least once, and the arrangement $\mathcal{A}(H)$ induced by $H$ is simple. There always exists a complete cell in $\mathcal{A}(H)$.*

## 2.2 3-colored point sets and balanced double wedges

A result equivalent to Corollary 1 is presented in the next theorem. This can be shown using standard duality transformation between points and lines.

**Theorem 4** *Let $S$ be a set of points in general position in $\mathbb{R}^2$ colored with 3 colors so that each color appears at least once. Then, There exists a double wedge that contains exactly one point of $S$ of each color.*

Next we turn our attention to balanced 3-colored point sets, and prove that a ham-sandwich-like theorem for double wedges exists.

**Theorem 5** *Let $S$ be a balanced set of 3-colored $6n$ points in general position the plane. There exists a double wedge containing exactly $n$ points of $S$ of each color.*

**Proof.** (Sketch) Without loss of generality we assume that the points of $S$ have distinct $x$-coordinates and distinct $y$-coordinates. For two distinct points $a$ and $b$ in the plane, let $\ell(a, b)$ denote the line passing through them. Consider the arrangement $\mathcal{A}$ of all the lines passing through two points from $S$.

Consider a horizontal line $\ell$ that does not contain any point from $S$. We walk on $\ell$ from left to right in a continuous form. For any point $p \in \ell$ we define an ordering $\sigma_p$ of $S$ as follows: consider the lines $\ell(p,q), q \in S$ and sort them by slope. Let $(p_1, \ldots, p_{6n})$ be the obtained sorting. By construction, any interval $\{p_i, p_{i+1}, \ldots, p_j\}$ of an ordering of $p$ corresponds to a set of points that can be covered by a double wedge whose apex is $p$ (and *vice versa*).

Given a sorting $\sigma_p = (p_1, \ldots, p_{6n})$ of $S$ we associate a polygonal curve as follows: for every $k \in \{1, 2, \ldots, 3n\}$ let $b_k$, and $g_k$ be the number of blue and green points in the set $S(p,k) = \{p_k, p_{k+1}, \ldots, p_{k+3n-1}\}$ of $3n$ points, respectively. We define the corresponding lattice point $q_k := (b_k - n, g_k - n)$, and the path $\phi(\sigma) = (q_1, \ldots, q_{3n}, -q_1, \ldots, -q_{3n}, q_1)$.

Observe that if $q_k = (0,0)$ for some $p \in \ell$ and some $k \leq 3n$, then there exists a balanced double wedge. Assume, for the sake of contradiction, that $q_k \neq (0,0)$ for all orderings $\sigma_p$ and all $k \leq 3n$. We observe several important properties of $\phi(\sigma)$:

- $\phi(\sigma)$ is centrally symmetric (w.r.t. the origin).
- $\phi(\sigma)$ is a closed path. Moreover, the interior of any edge of $\phi(\sigma)$ cannot contain the origin.
- If the orderings of two points $p$ and $q$ are equal, then their paths $\phi(\sigma_p)$ and $\phi(\sigma_q)$ are equal.
- Path $\phi(\sigma)$ has nonzero winding (with respect to the origin).

With these properties it can be seen that, when we walk from one cell to an adjacent one, not many changes can happen to the path $\phi(\sigma)$. In particular, the winding must have the same sign. However, after we have moved $p$ from $p_x = -\infty$ to $p_x = +\infty$, the corresponding orderings are reverse. In particular, their paths must have windings of different sign with respect to the origin. This contradicts the property that the motion of $p$ does not affect the winding number of $\phi(\sigma)$. □

**Corollary 2** *Let $L$ be a balanced set of $6n$ lines inducing a simple arrangement. Then, there exists a segment intersecting exactly $n$ lines of each color.*

## 3 Balanced partitions on closed Jordan curves

In this section we consider balanced 3-colored point sets on closed Jordan curves. It is easy to see that, by homeomorphism, it suffices to give proofs for the unit circle. We show that there is always a bipartition of the set which is balanced and that can be realized by at most two disjoint arcs on the circle. To prove the claim we use the following arithmetic lemma:

**Lemma 6** *For every integer $n \geq 2$, any integer $k \in \{1, 2, \ldots, n\}$ can be obtained from $n$ applying the functions $f(x) = \lfloor x/2 \rfloor$ and $g(x) = n - x$ to $n$ at most $2 \log n + O(1)$ times.*

Let $S^1$ be the unit circle in $\mathbb{R}^2$ with the usual parametrization $f(t) = (\cos(t), \sin(t)), t \in [0, 2\pi)$. Let $P$ be a balanced set of $3n$ points in $S^1$. The following theorem is a discrete version of the main result in [4]. There the methods are topological, while our approach is combinatorial, based on Lemma 6. The result can also be seen as a version of the *Necklace Theorem* [3] for closed curves. We say that a set $Q \subseteq S^1$ is a *2-arc* set if it is the union of at most two disjoint arcs of $S^1$.

**Theorem 7** *Let $P$ be a balanced set of 3-colored $3n$ points in $S^1$. For each $k \leq n$ there exists a 2-arc set $P_k \subseteq S^1$ containing exactly $k$ points of each color.*

**Proof.** (Sketch) Let $I$ be the set of numbers $k$ such that a subset $P_k$ as in the theorem exists. We prove that $I = \{1, \ldots, n\}$. By Lemma 6 it suffices to prove that: i) $n \in I$, ii) If $k \in I$ then $n - k \in I$, and iii) If $k \in I$ then $k/2 \in I$.

Claims i) and ii) follow from the fact that $S^1$ and the complementary of any 2-arc set are 2-arc sets, respectively. To prove iii), we lift $P$ to $\mathbb{R}^3$ using the moment curve. Abusing slightly the notation, we identify each point $f(t) = (\cos(t), \sin(t)), t \in [0, 2\pi)$ on $S^1$ with its corresponding parameter $t$. We assume that $0 \notin P_k$; otherwise we can change the parametrization of $S^1$ by rotating around the origin to assure this fact.

Then, for $t \in S^1$ we define $\gamma(t) = \{t, t^2, t^3\}$; also, if $S \subseteq S^1$, we define $\gamma(S) = \{\gamma(s) | s \in S\}$. As $0 \notin P_k$, any two disjoint arcs in $S^1$ become two disjoint intervals in $\gamma(S^1)$. Now we apply the ham-sandwich theorem to $\gamma(P_k)$ and obtain a plane that cuts each chromatic class in $\gamma(P_k)$ in half. In order to finish the proof, we must show that this plane induces the desired partition of $P$. Notice that any plane in $\mathbb{R}^3$ will intersect $\gamma(S^1)$ in at most 3 points, hence the projection will be a 2-arc set. □

The above proof can be generalized to $c$ colors: if $P$ contains $n$ points of each color on $S^1$, then for each $k \in \{1, \ldots, n\}$ there exists a $(c-1)$-arc set $P_k \subseteq S^1$ such that $P_k$ contains exactly $k$ points of each color (where the definition of a $(c-1)$-arc set is the natural extension of the 2-arc set). In the full version we also show that the bound in the number of intervals is tight.

## 4 L-lines in the plane lattice

We now consider a balanced partition problem for 3-colored point sets in the integer plane lattice. An *L-line with corner $q$* is the union of two different rays with common apex $q$, each of them being either vertical or horizontal. Figure 2 shows a balanced L-line with apex $q$.

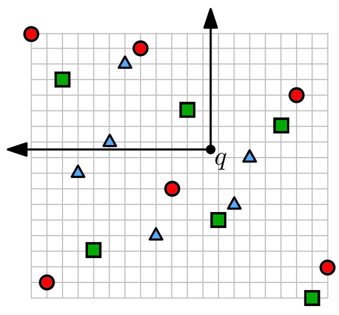


Figure 2: A balanced set of 18 points in the integer lattice with a nontrivial balanced L-line.

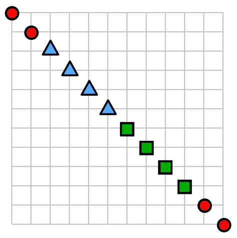


Figure 3: A set of 3-colored points in the plane lattice. Any L-line containing points of all three colors will fully contain a color class, hence this problem instance does not admit a balanced L-line.

L-lines in the lattice play a role comparable to that of ordinary lines in the real plane. For ordinary lines, Bereg and Kano [2] proved that if $S$ is a balanced 3-color point set whose convex hull is monochromatic, then there always exists a balanced line for $S$. It is easy to see that with no constraints on $S$, balanced L-lines (other than the trivial ones) may not exist. We extend the result in [2] to the the plane lattice. As in the Euclidean plane, there exist problem instances for which the only balanced L-line is the trivial one, see for example Figure 3. To prevent this, we impose a condition on $S$: all vertices in the orthogonal convex hull of $S$ have the same color. Observe that this condition is the natural translation to the one used for ordinary lines by Bereg and Kano. Moreover, an equivalent condition is the fact that for every L-line that splits $S$ into two non-empty subsets, each subset contains at least one red point.

**Theorem 8** *Let $S$ be a balanced set of 3-colored $3n$ points in the integer lattice. If the orthogonal hull is monochromatic, then there exists a nontrivial balanced L-line.*

**Proof.** (Sketch) We use a technique similar to that described in the proof of Theorem 5. Imagine sweeping the lattice with a horizontal line (from top to bottom). At any point of the sweep, we sort the points above the line from top to bottom, and the remaining points from left to right. By doing so, we obtain some orderings of the point set. Once we have swept all the points, we do a second sweep, this time vertical. As in the proof of Theorem 5, we associate a curve to each instant of the sweep. Moreover, the winding cannot change before and after sweeping a point. Once we have rotated the sweep line twice (i.e., we have rotated the line $\pi$ radians), we obtain a reverse ordering, obtaining a similar contradiction. Details will be given in a full version of this paper. $\square$

## 5 Concluding remarks

Observe that our results on double wedges can be viewed as partial answers to the following interesting open problem: Find all $k$ such that, for any set of $n$ red, $n$ green and $n$ blue points in general position in the plane, there exists a double wedge containing exactly $k$ points of each color. We have given here an affirmative answer for $k = 1, n/2$ and $n-1$ (Theorems 4 and 5). On the other hand, Theorem 7 can be viewed as an affirmative answer for all $k = 1, \dots, n$ if the points belong to a circle (or in general, if they are in convex position).

## Acknowledgments

Most of this research took place during the *5th International Workshop on Combinatorial Geometry* held at UPC-Barcelona from May 30 to June 22, 2012. We thank Inês Matos, Pablo Pérez Lantero, Vera Sacristán and all other participants for useful discussions.

F.H., M.K., C.S., and R.S. are partially supported by projects MINECO MTM2012-30951, Gen. Cat. DGR2009SGR1040, and ESF EUROCORES programme EuroGIGA, CRP ComPoSe: MICINN Project EUI-EURC-2011-4306. F.H. and C.S. are partially supported by project MICINN MTM2009-07242. M.K. also acknowledges the support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union. R.S. was partially supported by FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235.

## References

[1] A. Akopyan and R. Karasev. Cutting the same fraction of several measures. arXiv:1203.5591, 2012.

[2] S. Bereg and M. Kano. Balanced line for a 3-colored point set in the plane. *The Electronic Journal of Combinatorics*, 19:P33, 2012.

[3] J. Matousek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*, Chapter 3. Springer, 2007.

[4] W. Stromquist and D. Woodall. Sets on which several measures agree. *Journal of Mathematical Analysis and Applications*, 108(1):241–248, 1985.

# The Complexity of Separating Points in the Plane

Sergio Cabello*[†] Panos Giannopoulos[‡][§]

**Abstract**

We study the following separation problem: Given $n$ connected curves and two points $s$ and $t$ in the plane, compute the minimum number of curves one needs to retain so that any path connecting $s$ to $t$ intersects some of the retained curves. We give the first polynomial ($\mathcal{O}(n^3)$) time algorithm for the problem and show its generalized version, where several input points are to be separated, to be NP-hard for natural families of curves, like segments in two directions or unit circles.

## 1 Introduction

Let $C$ be a family of connected curves in the plane (typically, circles or segments, possibly of unit size), and let $s$ and $t$ be two points not incident to any curve $C$. In the 2-Point-Separation problem we want to compute a subset $C' \subseteq C$ of minimum cardinality that *separates* $s$ from $t$, i.e., any path connecting $s$ to $t$ intersects some curve of $C'$. Its generalization where several input points are to be separated will be referred to as Points-Separation. We actually solve a weighted version of 2-Point-Separation, where we have a weight function $w$ assigning weight $w(c) \geq 0$ to each curve $c \in C$ and a solution of minimum total weight is sought. This scenario is useful, for example, when we want to keep separated two points in a polygonal domain (weight 0) using a subset of disks (weight 1). See Fig. 1 for an example. Such problems naturally arise in so-called barrier problems when wireless sensors are modeled by disks [6] and [2].

We assume that some primitive operations involving the input curves (e.g., computing the intersection of two curves, deciding whether a curve is separating) can be carried out efficiently. These operations take constant time for semialgebraic curves of constant description complexity.

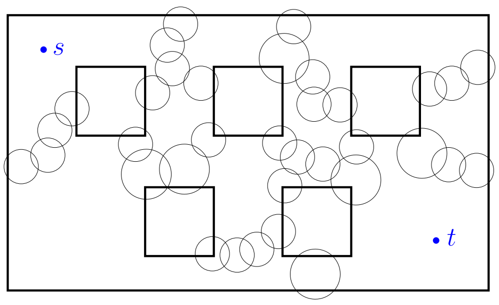


Figure 1: Weighted 2-Point-Separation instance: disks in a polygonal domain with rectangular holes.

**Results and methods.** We provide an algorithm that solves the weighted version of 2-Point-Separation in $\mathcal{O}(nk + n^2 \log n)$ time, where $k$ is the number of pairs of curves that intersect. The algorithm itself is simple, but its correctness is not obvious. We justify its correctness by considering an appropriate set of closed walks in the intersection graph of the curves and showing that it satisfies the so-called *3-path-condition* [9] (see also [8, Chapter 4]). This makes the connection of our problem to topology clear. This approach works when the optimal solution is given by at least 3 curves. The case where the optimal solution is attained by two curves is being taken care of separately by brute-force.

On the negative side, we use a reduction from Planar-3-SAT to show that Points-Separation is NP-hard for two natural families of curves: (i) horizontal and vertical segments and (ii) unit circles.

**Related work.** No polynomial-time algorithm that gives the exact optimum for 2-Point-Separation was previously known, even for unit disks. Gibson et al. [4] provide a polynomial-time $\mathcal{O}(1)$-approximation algorithm for disks. By considering several instances of this they also give an $\mathcal{O}(1)$-approximation for Points-Separation. Using our exact solution to 2-Point-Separation for the boundaries of the disks leads to a better approximation factor in the final outcome of their latter algorithm.

The ideas used here for 2-Point-Separation were already included in the unpublished manuscript with Alt and Knauer [1] for segments. This work replaces and extends that part of the manuscript. In the terminology used in Wireless Sensor Networks, we are computing a minimum-size 1-barrier [6].

*Department of Mathematics, IMFM and FMF, University of Ljubljana, Jadranska 19, SI-1000 Ljubljana, Slovenia, sergio.cabello@fmf.uni-lj.si.

[†]Research partially supported by the Slovenian Research Agency, program P1-0297, project J1-4106, and within the EUROCORES Programme EUROGIGA (project GReGAS) of the European Science Foundation.

[‡]Institut für Informatik, Universität Bayreuth, Universitätsstraße, 30, D-95447 Bayreuth, Germany, panos.giannopoulos@uni-bayreuth.de.

[§]Research supported by the German Science Foundation (DFG) under grant Kn 591/3-1.

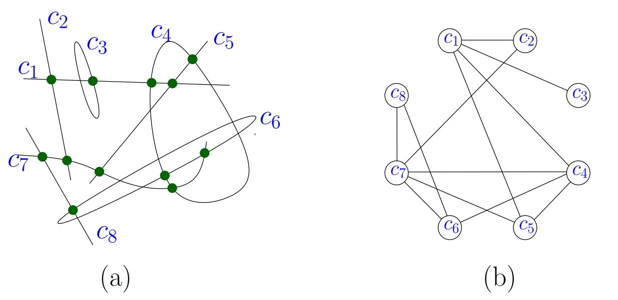


Figure 2: (a) A set of curves $C$ with the fixed intersection points $x_{c,c'}$. (b) The corresponding intersection graph $\mathbb{G}$.

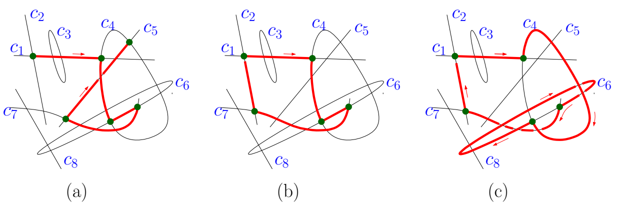


Figure 3: Some paths in the example of Fig. 2, using the fixed intersection points marked in Fig. 2. In (a) there is a $\pi$-path for the walk $\pi = c_2c_1c_4c_6c_7c_5c_4$. In (b) and (c) there are two different closed $\pi$-paths for the closed walk $\pi = c_2c_1c_4c_6c_7c_2$.

## 2 Algorithm for 2-Point-Separation

The use of the term *curve* will be restricted to elements of $C$. The use of the term *path* (or closed path) will be restricted to *parametric* paths constructed in our algorithm and proofs. The use of the term *walk* will be restricted to graphs. A *cycle* is a closed walk in a graph without repeated vertices.

We assume for simplicity that the segment $\overline{st}$ does not contain any intersection of two curves of $C$.

Let $\gamma$ be a path contained in $\bigcup C$, possibly with self-intersections. We define $N(\gamma)$ as the number of transversal crossings of $\gamma$ with $\overline{st}$ **modulo** 2 and counted with multiplicities. If $C' \subset C$ does not separate $s$ and $t$, then for any closed path $\gamma$ contained in $\bigcup C'$ we have $N(\gamma) = 0$.

For each two distinct curves $c$ and $c'$ from $C$ that intersect, we fix an arbitrary intersection point and denote it by $x_{c,c'}$. Given a curve $c \in C$ and two points $x, y$ on $C$, let $c[x \to y]$ be any path contained in $c$ connecting $x$ to $y$.

The set $C$ of input curves defines the intersection graph $\mathbb{G} = \mathbb{G}(C) = (C, \{cc' \mid c \cap c' \neq \emptyset\})$, which has $k$ edges; see Fig. 2. To each edge $cc'$ of $\mathbb{G}$ we attach the weight (abstract length) $w(c) + w(c')$. For any walk $\pi$ in $\mathbb{G}$ we use $\text{len}_{\mathbb{G}}(\pi)$ for its length and $C(\pi) = V(\pi)$ for the set of curves along $\pi$.

For each curve $r \in C$, we fix a shortest-path tree $T_r$ of $\mathbb{G}$ from $r$; if there are several, we fix one of them arbitrarily. For any $T_r$ of $\mathbb{G}$ and any edge $e \in E(\mathbb{G}) \setminus E(T_r)$, let $\tau(T_r, e)$ denote the closed walk obtained by concatenating the edge $e$ with the two paths in $T_r$ from $r$ to the endpoints of $e$. When $\tau(T_r, e)$ is a cycle it is called a *fundamental cycle* with respect to $T_r$.

Consider a walk $\pi = c_0c_1 \cdots c_t$ in $\mathbb{G}$. Let $\gamma$ be a path in $\mathbb{R}^2$. We say that $\gamma$ is a *$\pi$-path* if there are paths $\gamma_1, \dots, \gamma_{t-1}$ such that: the path $\gamma_i$ is contained in $c_i$ ($i = 1, \dots, t-1$), the path $\gamma_i$ goes from $x_{c_{i-1},c_i}$ to $x_{c_i,c_{i+1}}$ ($i = 1, \dots, t-1$), and the concatenation of $\gamma_1, \dots, \gamma_{t-1}$ gives $\gamma$. See Fig. 3(a) for an example.

If the walk $\pi = c_0c_1 \cdots c_t$ is closed, which means that $c_t = c_0$, then a closed path $\gamma$ is a *closed $\pi$-walk* if there are paths $\gamma_1, \dots, \gamma_t$ such that: the path $\gamma_i$ is contained in $c_i$ ($i = 1, \dots, t$), the path $\gamma_i$ goes from $x_{c_{i-1},c_i}$ to $x_{c_i,c_{i+1}}$ ($i = 1, \dots, t$ and $c_{t+1} = c_1$), and the concatenation of $\gamma_1, \dots, \gamma_t$ gives $\gamma$. See Fig. 3(b)–(c) for an example. If $\gamma$ is a $\pi$-path or a closed $\pi$-path, then $\gamma \subset \bigcup C(\pi)$. Even if $\pi$ is a cycle, a closed $\pi$-path may have self-intersections.

Given a walk $\pi = c_0c_1 \cdots c_t$ in $\mathbb{G}$ we can construct a $\pi$-path in linear time by concatenating $c_j[x_{c_{j-1},c_j} \to x_{c_j,c_{j+1}}]$ for $j = 1, \dots, t-1$. If $\pi$ is a closed walk with $c_0 = c_t$, we can obtain a closed $\pi$-path closing it with $c_0[x_{c_{t-1},c_0} \to x_{c_0,c_1}]$. When $C$ is a family of pseudosegments, there is a unique $\pi$-path for each walk $\pi$ and a unique closed $\pi$-path for each closed walk $\pi$.

We will mainly use closed $(\tau(T_r, e))$-paths, where $r$ is a curve of $C$ and $e \in E(\mathbb{G}) \setminus E(T_r)$. By $\gamma(r, e)$ we denote an arbitrary closed $(\tau(T_r, e))$-path.

### 2.1 The algorithm and its time complexity

First, we select the minimum-weight solution $C_{\leq 2}$ consisting of one or two curves from $C$. We do this by testing separately each curve and each pair of curves from $C$. Of course, it may be that $C_{\leq 2}$ is undefined. We remove from $C$ any curve that alone separates $s$ and $t$, and keep using $C$ for the remaining set of curves. Next we compute the set

$$P = \{(r, e) \in C \times E(\mathbb{G}) \mid e \in E(\mathbb{G}) \setminus E(T_r) \text{ and } N(\gamma(r, e))) \text{ is odd}\}.$$

We choose $(r^*, e^*) \in \arg \min_{(r,e) \in P} \text{len}_{\mathbb{G}}(\tau(T_r, e))$, and compute $C_{>2} = C(\tau(T_{r^*}, e^*))$. It may happen that $P$ is empty, which means that $(r^*, e^*)$ and $C_{>2}$ are undefined. Finally we return the lightest set of curves between $C_{\leq 2}$ and $C_{>2}$, if they are both defined; the only one of them that is defined, if only one among $C_{\leq 2}$ and $C_{>2}$ is defined; and "$C$ does not separate $s$ and $t$", if both $C_{\leq 2}$ and $C_{>2}$ are undefined. This finishes the description of the algorithm. We will refer to this algorithm as ALGORITHM-2PS.

The algorithm can be implemented in $\mathcal{O}(n^2k + n^2 \log n)$ time in a straightforward way. We obtain a better bound by computing $C_{>2}$ as follows. For each $r \in C$ and $c \in C$ we compute and store some

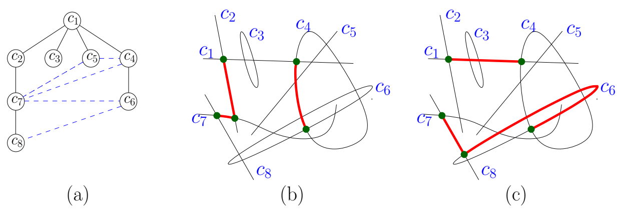


Figure 4: (a) Tree $T_{c_1}$ for the scenario of Fig. 2 assuming curves of unit weight. (b) Possible $(T_{c_1}[c_8])$-path and $(T_{c_1}[c_6])$-path used to compute $N_{c_1}(c_8)$ and $N_{c_1}(c_6)$. (c) Possible $(c_7c_8c_6c_4)$-path and $(c_4c_1c_2)$-path that are used to compute $N(\gamma(c_1, c_6c_8))$ in Lemma 1.

information $I(r, c)$ such that, for each $r \in C$ and $cc' \in E(\mathbb{G}) \setminus E(T_r)$, we can obtain from $I(r, c)$ and $I(r, c')$ the values $N(\gamma(r, cc'))$ and $\text{len}_{\mathbb{G}}(\tau(T_r, cc'))$ in $O(1)$ time. This information $I(r, c)$ is essentially the length of the path $T_r[c]$ in $T_r$ from $r$ to $c$ and the value of $N(\cdot)$ for a $T_r[c]$-path; see Fig 4. Such information $I(r, c)$ can be computed in $O(1)$ time per pair $(r, c)$ after computing shortest path trees from each vertex by making a top-to-bottom traversal of each $T_r$.

**Lemma 1** ALGORITHM-2PS *can be modified to run in $\mathcal{O}(nk + n^2 \log n)$ time. Furthermore, if the weights of the curves $C$ are $0$ or $1$, then* ALGORITHM-2PS *can be modified to run in $\mathcal{O}(nk + n^2)$ time.*

## 3 Correctness of the algorithm for 2-Point-Separation

Since in ALGORITHM-2PS we test each curve of $C$ whether it separates $s$ and $t$, and, if it does, then remove it from $C$, and since every such separating curve is tested for optimality, **we can assume henceforth that no curve in $C$ separates $s$ and $t$.** We show that this assumption implies that the choice of closed $(\tau(T_r, e))$-paths made in the algorithm is irrelevant.

**Lemma 2** *Let $\pi$ be a closed walk in $\mathbb{G}$ and let $\gamma$ and $\gamma'$ be two closed $\pi$-paths. Then $N(\gamma) = N(\gamma')$.*

### 3.1 3-Path-Condition

Consider the set of closed walks

$$\Pi(C) = \{\pi \mid \pi \text{ a closed walk in } \mathbb{G}(C); \text{ each closed } \pi\text{-curve } \gamma \text{ has } N(\gamma) \text{ odd}\}.$$

For the moment, we drop the dependency on $C$ and use $\Pi = \Pi(C)$; towards the end we will use $\Pi(C_*)$ for some $C_* \subseteq C$. We have the following property, known as 3-path-condition.

**Lemma 3** *Let $\alpha_0, \alpha_1, \alpha_2$ be 3 walks in $\mathbb{G}$ from $c$ to $c'$. For $i = 0, 1, 2$, let $\pi_i$ be the closed walk obtained by concatenating $\alpha_{i-1}$ and the reverse of $\alpha_{i+1}$, where indices are modulo 3. If one of the walks $\pi_0$, $\pi_1$, or $\pi_2$ is in $\Pi$, then at least two of them are in $\Pi$.*

When a family of closed walks satisfies the 3-path-condition, there is a general method to find a shortest element in the family. The method is based on considering so-called fundamental cycles defined by shortest-path trees, which is precisely what ALGORITHM-2PS is doing specialized for the family $\Pi$. This is basically the fundamental cycle method used in Topological Graph Theory. See [9] or [8, Chapter 4] for the original approach, and [3] for a recent extension to weighted, directed graphs.

**Lemma 4** *If $\Pi$ is nonempty, then the closed walk $\tau^* = \tau(T_{r^*}, e^*)$ computed by* ALGORITHM-2PS *is a cycle and is a shortest closed walk of $\Pi$.*

### 3.2 Feasibility

The next step is to show that, when $C_{>2}$ is defined, it is a feasible solution. For this we find a closed, simple path contained in $C_{>2}$ that separates $s$ and $t$.

**Lemma 5** *Assume that $\Pi$ is nonempty and let $\pi^*$ be any shortest cycle in $\Pi$. The set of curves $C(\pi^*)$ separates $s$ and $t$.*

We next argue that the algorithm computes a feasible solution, when it exists. We know that $\tau(T_{r^*}, e^*)$ separates, when it is defined, but could it happen that $\Pi$ is empty and thus $(r^*, e^*)$ is undefined? The following lemma asserts that this is not the case.

**Lemma 6** *If $C$ separates $s$ and $t$ but no two curves $C$ separate $s$ and $t$, then $\Pi$ is nonempty.*

### 3.3 Main result

We can now prove that ALGORITHM-2PS is correct. The interesting case is when each optimal solution has at least 3 curves. We can use Lemma 6 to argue that $\Pi(C_*)$ is non-empty for each optimal solution $C_*$, which in turn implies that $\Pi(C)$ is non-empty. It follows from Lemma 4, that $\tau^*$ is defined and it is a shortest cycle in $\Pi$. Since $C(\tau^*)$ is a feasible solution and the shortest cycle in $\Pi(C_*)$ cannot be shorter than the shortest cycle in $\Pi(C)$, we obtain that $w(C(\tau^*)) \leq w(C_*)$ and thus $C(\tau^*)$ is also an optimal solution.

**Theorem 7** *The weighted version of* 2-POINTS-SEPARATION *can be solved in $\mathcal{O}(nk + n^2 \log n)$ time, where $n$ is the number of input curves and $k$ is the number of pairs of curves that intersect. When the curves have weights $0$ and $1$, the running time becomes $\mathcal{O}(nk + n^2)$.*

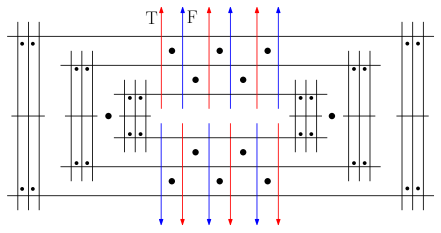


Figure 5: Variable gadget for Points-Separation with horizontal/vertical segments. The segments with arrows may be extended.

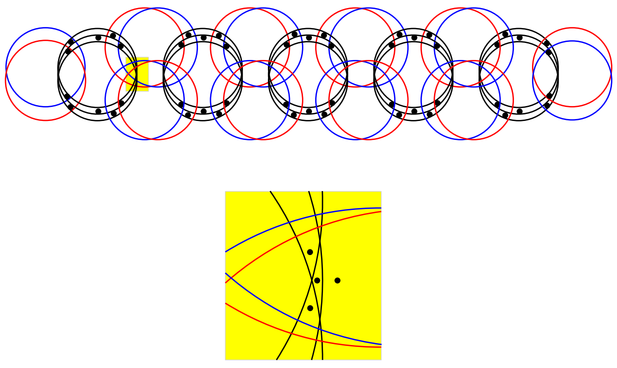


Figure 6: Variable gadget for Points-Separation with unit circles

## 4 Hardness of Point-Separation

We show that Points-Separation is NP-hard for two families of curves: (i) horizontal and vertical segments, and (ii) unit circles. We reduce from Planar-3-SAT, which was shown to be NP-hard in [7]. Planar-3-SAT is the restriction of 3-SAT to formulae whose incidence graph is planar and has a plane 3-legged rectilinear representation [5].

In our construction, we replace each variable and clause by a gadget. Our drawings use black segments/circles for curves that must be in any feasible solution. This is enforced by extra points.

The variable gadgets shown in Figures 5,6. A minimum-cardinality feasible solution for the variable gadget contains all the black curves and either all the red curves or all the blue curves. We use this red/blue alternative to encode True/False assignments.

In the case of segments, a clause is represented as shown in Fig. 7. It consists of additional black segments that must be included in any feasible solution and two additional points. For each variable that occurs in the clause, we elongate one segment from the corresponding variable gadget to separate the two additional points.

For unit circles, the rectangle representing a clause is deformed into an M-like corridor that is made from unit circles, as shown in Fig 8. The corridor is narrowed near the variables such that a red or blue circle (depending on the literal) disconnects the corridor.

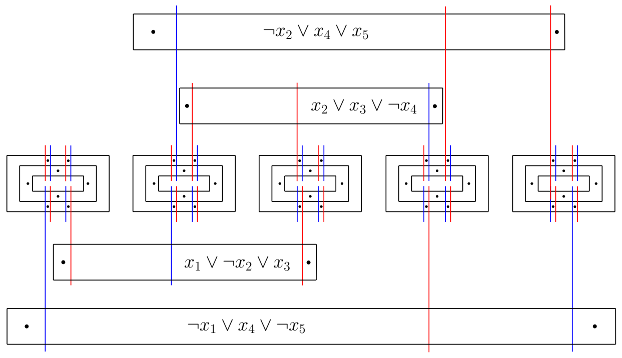

$\neg x_2 \vee x_4 \vee x_5$
$x_2 \vee x_3 \vee \neg x_4$
$x_1 \vee \neg x_2 \vee x_3$
$\neg x_1 \vee x_4 \vee \neg x_5$

Figure 7: An example construction with segments; the frames in the gadgets are shown as rectangles.

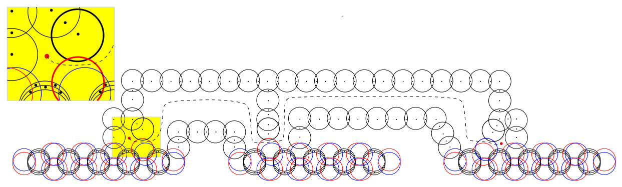


Figure 8: Clause $(x_2 \vee x_3 \vee \neg x_4)$ with unit circles.

**Theorem 8** *Points-Separation is NP-hard for families of horizontal/vertical segments and for families of unit circles.*

## References

[1] H. Alt, S. Cabello, P. Giannopoulos, and C. Knauer. Minimum cell connection and separation in line segment arrangements. *CoRR*, abs/1104.4618, 2011.

[2] S. Bereg and D. G. Kirkpatrick. Approximating barrier resilience in wireless sensor networks. In *Proc. 5th ALGOSENSORS*, volume 5804 of *LNCS*, pages 29–40. Springer, 2009.

[3] S. Cabello, É. C. de Verdière, and F. Lazarus. Finding shortest non-trivial cycles in directed graphs on surfaces. In *Proc. 26th SoCG*, pages 156–165, 2010.

[4] M. Gibson, G. Kanade, and K. Varadarajan. On isolating points using disks. In *Proc. 19th ESA*, volume 6942 of *LNCS*, pages 61–69. Springer, 2011.

[5] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Discret. Math.*, 5(3):422–427, 1992.

[6] S. Kumar, T.-H. Lai, and A. Arora. Barrier coverage with wireless sensors. *Wireless Networks*, 13(6):817–834, 2007.

[7] D. Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[8] B. Mohar and C. Thomassen. *Graphs on surfaces.* Johns Hopkins Studies in the Mathematical Sciences. John Hopkins University Press, 2001.

[9] C. Thomassen. Embeddings of graphs with no short noncontractible cycles. *J. of Comb. Theory, B*, 48(2):155–177, 1990.

# Kinetic Euclidean 2-centers in the Black-Box Model[*]

Mark de Berg[†] Marcel Roeloffzen[†] Bettina Speckmann[†]

**Abstract**

We study the 2-center problem for moving points in the plane. Given a set $P$ of $n$ points, the Euclidean 2-center problem asks for two congruent disks of minimum size that together cover $P$. Our methods work in the black-box KDS model, where we receive the locations of the points at regular time steps and we know an upper bound $d_{\max}$ on the maximum displacement of any point within one time step.

We show how to maintain a $(1+\varepsilon)$-approximation of the Euclidean 2-center in amortized sub-linear time per time step, under certain assumptions on the distribution of the point set $P$. In many cases—namely when the distance between the centers of the disks is relatively large or relatively small—the solution we maintain is actually optimal.

## 1 Introduction

The clustering problem is to partition a given set of objects into clusters, that is, into subsets consisting of similar objects. These objects are often (represented by) points in some 2- or higher dimensional space, and the similarity between points corresponds to the distance between them. We are interested in a setting with two clusters of points in the plane. Given a set $P$ of $n$ points, the Euclidean 2-center problem—or 2-center problem for short—asks for two congruent disks of minimum size that together cover $P$. The 2-center problem can also be interpreted as a facility-location problem, where the goal is to place two facilities such that the distance from any client in $P$ to its nearest facility is minimized.

The 2-center problem and the more general $k$-center problem—which asks for $k$ disks to cover $P$—have been studied extensively since their introduction by Sylvester [20] in 1857. Closely related is the rectilinear $k$-center problem which asks for $k$ congruent squares to cover the point set. Both the Euclidean and the rectilinear $k$-center problem are NP-hard [14] when $k$ is part of of the input, but polynomial-time solutions are possible when $k$ is a constant. The rectilinear $k$-center problem can be solved quite efficiently for small $k$. For $k = 2, 3$ the optimal rectilinear $k$-center can be computed in $O(n)$ time [8, 13, 19] and for $k = 4, 5$ in $O(n \log n)$ time [15, 16]. In contrast, no sub-quadratic algorithm was known for the Euclidean 2-center for many years, until Sharir [18] developed an $O(n \log^9 n)$ time algorithm. The currently best solution takes $O(n \log^2 n(\log \log n)^2)$ time [6]. Other results include an $\Omega(n \log n)$ lower bound for $k = 2$ [17] and algorithms that compute a $(1+\varepsilon)$-approximation of the $k$-center [1, 2]. For the 2-center problem in $\mathbb{R}^2$ such an $(1+\varepsilon)$-approximation can be computed in $O(n) + (2/\varepsilon)^{O(1)}$ time.

**The kinetic 2-center problem.** The results mentioned so far are for static point sets, but also *kinetic* versions of the 2-center problem have been studied. Here we want to maintain the optimal 2-centers as the points in $P$ move. Unfortunately, even under the restriction that the speed of the points in $P$ is bounded by a given value $v_{\max}$, the speed of the centers cannot be bounded if one maintains the optimal 2-center. For mobile facility location this is undesirable as the centers represent moving facilities which often have a bounded speed as well. Hence, Durocher and Kirkpatrick [9] describe a general strategy for maintaining an approximate 2-center in such a way that the speed of the disk centers is bounded. One variant of their strategy achieves an approximation ratio of $8/\pi \approx 2.55$ while the maximum speed of the disk centers is bounded by $(8/\pi + 1)\, v_{\max} \approx 3.55\, v_{\max}$. Maintaining this approximation is done using the kinetic data structures (KDS) framework by Basch *et al.* [3]. When viewed as a clustering problem the centers have no explicit meaning and no bound on their speed is necessary. For this case efficient KDSs for the discrete version of the $k$-center problem, where the disk centers must be chosen from the input set $P$, have been given [7, 11].

**Problem statement.** The previous results on the kinetic 2-center problem [7, 9, 11] use the standard KDS model, where the trajectories of the points are known in advance. However, in many applications the trajectories are not known and the standard KDS framework cannot be used. Hence, we study the kinetic 2-center problem in the so-called *black-box model* [4, 5, 12]. In the black-box model the locations of the points are reported at regular time steps $t_0, t_1, \cdots$, and we are given a value $d_{\max}$ such that any point can move at most distance $d_{\max}$ from one time step to the next. Thus, when $p(t)$ denotes the location

[*]M.R. and B.S. were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 600.065.120 and 639.022.707, respectively.

[†]Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands. {*mdberg,mroeloff,speckman*}*@win.tue.nl*.

of point $p$ at time $t$, we have $\mathrm{dist}(p(t_i), p(t_{i+1})) \leqslant d_{\max}$ for all $p \in P$ and every time step $t_i$. We want to maintain the 2-center of the set $P(t) := \{p(t) : p \in P\}$ at every time step while using coherence to speed up the computations. This is not possible without restricting the relation of the maximum displacement $d_{\max}$ and the distribution of the point set $P$: if all points lie within distance $d_{\max}$ from each other then the distribution at time $t+1$ need not have any relation to the distribution at time $t$, and so there is no coherence that can be used. Following our previous papers [4, 5] we assume the following.

> ***Displacement Assumption***: *There is a maximum displacement $d_{\max}$ and constant $k$ such that for each point $p \in P$ and any time step $t_i$ we have*
>
> - $\mathrm{dist}(p(t_i), p(t_{i+1})) \leqslant d_{\max}$, *and*
> - *there are at most $k$ other points from $P$ within distance $d_{\max}$ from $p(t_i)$.*

Under this assumption, we formulate our bounds in terms of the so-called *$k$-spread* [10] of $P$, which is defined as follows. Let $\mathrm{mindist}_k(P) := \min_{p\in P} \mathrm{dist}(p, \mathrm{NN}_k(p, P))$ denote the smallest distance from any point $p \in P$ to its $k$-nearest neighbor $\mathrm{NN}_k(p, P)$. Then the $k$-spread $\Delta_k$ of $P$ is defined as $\Delta_k(P) := \mathrm{diam}(P)/\mathrm{mindist}_k(P)$. The 1-spread is simply the regular spread of a point set. We use the $k$-spread instead of regular spread, since two points may pass by each other at a very close distance, causing a small value for $\mathrm{mindist}_1(P)$ and, consequently, a high spread. It is much less likely that $k$ points are very close simultaneously, and so $\mathrm{mindist}_k(P)$ tends to not be very small. For a good $k$-spread we also need the diameter not to be too large. This is somewhat unnatural for the 2-center problem: when the two clusters are far apart, the $k$-spread may become very large even though within each cluster, the points are very evenly distributed. Hence we introduce the so-called *$(2, k)$-spread* $\Delta_{2,k}(P)$:

$$\Delta_{2,k}(P) := \min_{P_1,P_2} \max(\Delta_k(P_1), \Delta_k(P_2)),$$

where the minimum is taken over all possible partitions of $P$ into two subsets $P_1, P_2$. (The partition defining $\Delta_{2,k}(P)$ does not need to be the same as the partition defining the optimal clustering in the 2-center problem, but since $\Delta_{2,k}(P)$ is the minimum over all partitions this can lead only to a better $(2, k)$-spread.) We express our results using the $(2, k)$-spread of $P$, that is, using the maximum value of $\Delta_{2,k}(P(t_i))$ over all time steps $t_i$, which we abbreviate as $\Delta_{2,k} := \max_{t_i} \Delta_{2,k}(P(t_i))$.

**Results and Organization.** We study the kinetic Euclidean 2-center problem from a clustering point of view: without restrictions on the speed of the centers.

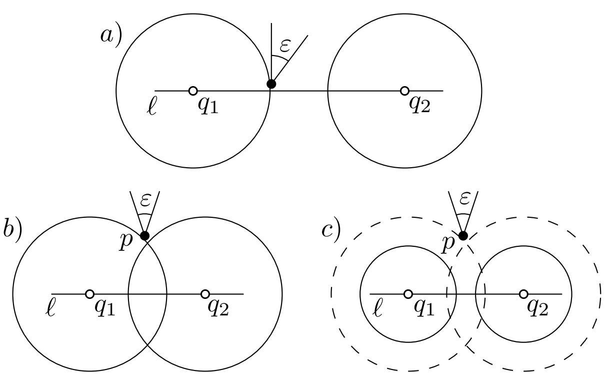


Figure 1: When $a$) $\mathrm{dist}(q_1, q_2) \geqslant 2r + \varepsilon r$ or $b$) $\mathrm{dist}(q_1, q_2) \leqslant 2r - \varepsilon r/2$ the 2-center disks that cover $P_\varepsilon$ also cover $P$, and $c$) otherwise blowing up the disks by a factor $1 + \varepsilon$ ensures they cover $P$.

We investigate the Euclidean 2-center problem and show how to maintain a $(1 + \varepsilon)$-approximation, for any $0 < \varepsilon \leqslant \pi/4$ in $O((k/\varepsilon^3)\Delta_{2,k} \log^3 n(\log\log n)^2)$ amortized time. In many cases—when the distance between the centers of the disks is relatively large or small—the solution we maintain is optimal.

## 2 The Euclidean 2-center

The Euclidean 2-center problem asks for two congruent disks of minimum size that together cover $P$. Our global strategy to maintain the Euclidean 2-center kinetically is to find a subset $Q \subseteq P$ containing points that are in some sense on the outside of $P$. We then compute the optimal 2-center for $Q$ and show that it is an approximation of the 2-center of $P$. Maintaining the approximate 2-center can then be done by maintaining $Q$. First we define what exactly it means for a point to be on the outside of the point set.

Define a point $p \in P$ to be *$\varepsilon$-interesting* if there is a wedge $W_\varepsilon(p)$ with apex $p$ and opening angle $\varepsilon$ such that $W_\varepsilon(p)$ does not contain any other points of $P$. Let $P_\varepsilon$ denote the set of $\varepsilon$-interesting points in $P$. We show that it suffices to consider the points in $P_\varepsilon$ to get (an approximation of) an optimal solution to the 2-center problem on $P$. In the following we use $\mathrm{disk}(q, r)$ to denote the disk of radius $r$ centered at $q$.

**Lemma 1** *Let* $\mathrm{disk}(q_1, r)$ *and* $\mathrm{disk}(q_2, r)$ *be the two disks of an optimal solution for the Euclidean 2-center problem on $P_\varepsilon$, for some $\varepsilon < \pi/4$. If* $\mathrm{dist}(q_1, q_2) \leqslant 2r - \varepsilon r/2$ *or* $\mathrm{dist}(q_1, q_2) \geqslant 2r + \varepsilon r$ *then* $\mathrm{disk}(q_1, r)$ *and* $\mathrm{disk}(q_2, r)$ *are an optimal solution for the 2-center problem on $P$; otherwise* $\mathrm{disk}(q_1, r + \varepsilon r)$ *and* $\mathrm{disk}(q_2, r + \varepsilon r)$ *are a $(1 + \varepsilon)$-approximation for the 2-center problem on $P$.*

*Proof (sketch).* First consider the case $\mathrm{dist}(q_1, q_2) \leqslant 2r - \varepsilon r/2$. Since $P_\varepsilon \subseteq P$, the disks in an optimal solution for $P$ cannot have radius smaller than $r$. Hence, it suffices to prove that $\mathrm{disk}(q_1, r) \cup \mathrm{disk}(q_2, r)$ covers $P$. Suppose for a contradiction there is an uncovered point in $P$. Assume without loss of generality that the line $\ell$ through $q_1$ and $q_2$ is horizontal, and let $p$ be the highest uncovered point above $\ell$, see Figure 1. (If all uncovered points lie below $\ell$ we can apply a similar argument to the lowest uncovered point.) The condition $\mathrm{dist}(q_1, q_2) \leqslant 2r - \varepsilon r/2$ implies that the "vertical" wedge $W_\varepsilon(p)$ does not intersect $\mathrm{disk}(q_1, r) \cup \mathrm{disk}(q_2, r)$, and the fact that $p$ is the highest uncovered point implies that $W_\varepsilon(p)$ does not contain any other uncovered point. Hence, $p \in P_\varepsilon$, contradicting that $\mathrm{disk}(q_1, r)$ and $\mathrm{disk}(q_2, r)$ form a solution for $P_\varepsilon$.

When $\mathrm{dist}(q_1, q_2) \geqslant 2r + \varepsilon r$ we can show in a similar way that $\mathrm{disk}(q_1, r) \cup \mathrm{disk}(q_2, r)$ covers $P$. When $2r - \varepsilon r/2 \leqslant \mathrm{dist}(q_1, q_2) \leqslant 2r + \varepsilon r$ we cannot guarantee this, but if we blow up the disks by a factor $(1+\varepsilon)$ we are essentially back in the first case and we can apply the same reasoning. □

By Lemma 1 we obtain a $(1+\varepsilon)$-approximation for the Euclidean 2-center problem if we can maintain the set $P_\varepsilon$. This seems difficult, so instead we maintain a superset $P^*_\varepsilon \supseteq P_\varepsilon$ defined as follows. Let $W_{\varepsilon/2}$ be a wedge of opening angle $\varepsilon/2$. We say that $W_{\varepsilon/2}$ is a *canonical* $(\varepsilon/2)$*-wedge* if the counter-clockwise angle that its angular bisector makes with the positive $x$-axis is $i\varepsilon/2$, for some integer $0 \leqslant i < \lceil 4\pi/\varepsilon \rceil$. We now define $P^*_\varepsilon$ as the set of points $p$ in $P$ that have an empty canonical $(\varepsilon/2)$-wedge (that is, a wedge not containing points from $P$) with apex $p$. The following observation implies that Lemma 1 is still true if we replace $P_\varepsilon$ by $P^*_\varepsilon$.

**Observation 1** *Any point $p \in P$ that is the apex of an empty $\varepsilon$-wedge is also the apex of an empty canonical $(\varepsilon/2)$-wedge, so $P_\varepsilon \subseteq P^*_\varepsilon$.*

We are left with the problem of maintaining $P^*_\varepsilon$. This is done in a similar fashion as we maintained the convex hull vertices in a previous paper [4]: Each point $p \in P$ gets a time stamp that indicates how many time steps it takes before $p$ can be in $P^*_\varepsilon$. At each time step we then consider only points whose time stamps have expired. Recall that there are $\lceil 4\pi/\varepsilon \rceil$ different classes of canonical wedges, corresponding to the orientation of their angular bisector. We treat each of these classes separately. Consider one such class, and assume without loss of generality that its angular bisector is pointing vertically upward. We wish to maintain the set $P^{*,\mathrm{up}}_\varepsilon$ of points whose upward canonical wedge is empty. Define $W^{\mathrm{down}}(p)$ to be the wedge with apex $p$ that is the mirrored image of the upward canonical wedge of $p$, and let $\mathcal{W}^{\mathrm{down}}(t) := \{W^{\mathrm{down}}(p(t)) : p \in P\}$ be the set of all such downward wedges. Then a point $q$ lies in the upward canonical wedge of $p$ if and only if $p \in W^{\mathrm{down}}(q)$.

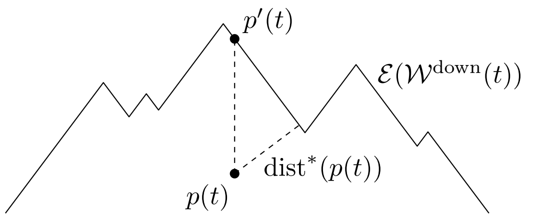


Figure 2: The point $p'(t)$ is the projection of $p(t)$ on $\mathcal{E}(\mathcal{W}^{\mathrm{down}}(t))$.

This implies the following lemma.

**Lemma 2** *Let $\mathcal{E}(\mathcal{W}^{\mathrm{down}}(t))$ denote the upper envelope of the downward wedges at time $t$. Then $p \in P^{*,\mathrm{up}}_\varepsilon(t)$ if and only if $p(t)$ is a vertex of $\mathcal{E}(\mathcal{W}^{\mathrm{down}}(t))$.*

Because of the bounded speed of the points, we know that points far from the upper envelope $\mathcal{E}(\mathcal{W}^{\mathrm{down}})$ need a lot of time before they can appear on the envelope. Hence, we can use the distance from $p$ to $\mathcal{E}(\mathcal{W}^{\mathrm{down}})$ to define its time stamp. To be able to compute time stamps quickly, we will not use the Euclidean distance from $p$ to $\mathcal{E}(\mathcal{W}^{\mathrm{down}})$ but an approximation of it. Let $p'(t)$ be the vertical projection of $p(t)$ onto $\mathcal{E}(\mathcal{W}^{\mathrm{down}}(t))$; see Figure 2. Then our approximated distance is defined as $\mathrm{dist}^*(p(t)) := \mathrm{dist}(p(t), p'(t)) \cdot \sin(\varepsilon/4)$. Note that $\mathrm{dist}^*(p(t))$ is equal to the distance from $p(t)$ to the boundary of the downward wedge $W^{\mathrm{down}}(p'(t))$. Because $W^{\mathrm{down}}(p'(t))$ is completely below (or on) $\mathcal{E}(\mathcal{W}^{\mathrm{down}})$, the actual Euclidean distance from $p(t)$ to $\mathcal{E}(\mathcal{W}^{\mathrm{down}})$ is at least $\mathrm{dist}^*(p(t))$. Hence, we can safely use $\mathrm{dist}^*(p(t))$ to define the time stamps. Thus, when we compute the time stamp of a point p at time $t$ we set

$$t^{\mathrm{up}}(p) := \min(\lfloor \mathrm{dist}^*(p(t))/2d_{\max} \rfloor + 1, n).$$

**Lemma 3** *If a point $p$ receives time stamp $t^{\mathrm{up}}(p)$ at time $t_i$, then $p$ cannot be on $\mathcal{E}(\mathcal{W}^{\mathrm{down}}(t_j))$ for $t_i < t_j < t_i + t^{\mathrm{up}}(p)$.*

The final time stamp of a point $p$ is defined as the minimum over all time stamps computed for $p$ for the $\lceil 4\pi/\varepsilon \rceil$ different wedge orientations. The algorithm for maintaining the Euclidean 2-center can now be summarized as follows. Initially (at time $t = t_0$) we compute a time stamp $t(p)$ for every point $p$, which is the minimum over the time stamps for the $\lceil 4\pi/\varepsilon \rceil$ canonical wedge orientations. Then at each time step $t = t_i$ we take the set $Q(t)$ of points whose time stamps expire at time $t$. For each canonical orientation we use a simple sweep-line algorithm to compute in $O(|Q(t)| \log |Q(t)|)$ time the envelope of the mirrored wedges of the points in $Q(t)$. Since there are $\lceil 4\pi/\varepsilon \rceil$ different orientations this takes $O((1/\varepsilon)|Q(t)| \log |Q(t)|)$ time in total. The collection of all points $p \in Q(t)$ that are a vertex of at least one of the envelopes is the set $P^*_\varepsilon(t)$. We then solve the Euclidean 2-center problem on $P^*_\varepsilon(t)$ using an algorithm for static points, giving us two disks disk$(q_1, r)$ and disk$(q_2, r)$. (To get the best running time, we use Chan's algorithm [6] for this.) If $2r - \varepsilon r/2 \leqslant \text{dist}(q_1, q_2) \leqslant 2r + \varepsilon r$ then we report disk$(q_1, r + \varepsilon r)$ and disk$(q_2, r + \varepsilon r)$, otherwise we report disk$(q_1, r)$ and disk$(q_2, r)$. Finally, we compute new time stamps for the points in $Q(t)$. Since we already have all the envelopes this can be done in $O((1/\varepsilon)|Q(t)|\log|Q(t)|)$ time in total.

The running time of our algorithm strongly depends on the number of points in $Q(t)$. Although in the worst case $|Q(t)|$ may be large, we can show that it is small on average. The proof of the following lemma is similar to a proof in a previous paper [4, Lemma 6].

**Lemma 4** *The number of points in $Q(t)$ is $O((1/\varepsilon^2)k\Delta_{2,k}\log n)$ amortized.*

Using that the static 2-center algorithm by Chan [6] on $m$ points runs in $O(m\log^2 m(\log\log m)^2)$ time, we obtain the following theorem.

**Theorem 5** *Let $P$ be a set of $n$ moving points that adheres to the Displacement Assumption with parameters $k$ and $d_{\max}$, let $\Delta_{2,k}$ denote the maximum $(2,k)$-spread of $P$ at any time $t$, and let $0 < \varepsilon \leqslant \pi/4$. Then we can maintain a $(1+\varepsilon)$-approximation of the Euclidean 2-center for $P$ in the black-box KDS model in $O((k/\varepsilon^3)\Delta_{2,k}\log^3 n(\log\log n)^2))$ amortized time per time step and using $O(n)$ space.*

## 3 Conclusions

We have shown how to maintain an approximation of the Euclidean 2-center problem in amortized sublinear time in the black-box model under certain assumptions on the distribution of the points. In the solution presented here the centers can "jump" between time steps. That is, between two consecutive time steps the distances between the centers can be very large compared to $d_{\max}$. For clustering this is not a problem, but for facility location problems this is undesirable. Durocher and Kirkpatrick [9] show a lower bound of $\sqrt{2}$ on the approximation ratio when the centers move with bounded speed. They provide an approximation scheme that achieves an approximation ratio of $8/\pi \approx 2.55$. We have also investigated bounded speed approximations for Euclidean kinetic 2-center problem in the black-box model. In the full paper we show how to obtain a 2.28-approximation for the Euclidean 2-center, such that the centers move at most $4\sqrt{2}d_{\max}$ per time step. There, we also study the rectilinear version of the kinetic 2-center problem (with and without speed restriction for the centers).

## References

[1] P.K. Agarwal and C.M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica* 33:201–226, 2002.

[2] M. Bādoiu, S. Har-Peled and P. Indyk. Approximate clustering via core-sets. In *Proc. 34th ACM Symp. Th. Comp.*, pages 250–257, 2002.

[3] J. Basch, L.J. Guibas and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Symp. Disc. Alg.*, pages 747–756, 1997.

[4] M. de Berg, M. Roeloffzen and B. Speckmann. Kinetic convex hulls and Delaunay triangulations in the black-box model. In *Proc. 27th ACM Symp. Comp. Geom.*, pages 244–253, 2011.

[5] M. de Berg, M. Roeloffzen and B. Speckmann. Kinetic compressed quadtrees in the black-box model with applications to collision detection for low-density scenes. In *Proc. 20th Euro. Symp. Alg.*, pages 383–394, 2012.

[6] T.M. Chan. More planar two-center algorithms. *Comp. Geom.* 13:189–198, 1999.

[7] B. Degener, J. Gehweiler and C. Lammersen. Kinetic facility location. *Algorithmica* 57:562-584, 2010.

[8] Z. Drezner. On the rectangular $p$-center problem. *Naval Res. Log. Quart.* 34:229–234, 1987.

[9] S. Durocher and D. Kirkpatrick. Bounded-velocity approximations of mobile Euclidean 2-centres. *Int. J. Comp. Geom. Appl.* 18(3):161–183, 2008.

[10] J. Erickson. Dense point sets have sparse Delaunay triangulations. *Disc. Comp. Geom.* 30:83–115, 2005.

[11] S. Friedler and D. Mount. Approximation algorithm for the kinetic robust K-center problem. *Comp. Geom.* 43:572–586, 2010.

[12] J. Gao, L.J. Guibas and A. Nguyen. Deformable spanners and applications. In *Proc. 20th ACM Symp. Comp. Geom.*, pages 190–199, 2004.

[13] M. Hoffmann. A simple linear algorithm for computing rectangular three-centers. In *Proc. 11th Can. Conf. Comp. Geom.*, pages 72-75, 1999.

[14] N. Megiddo and K.J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comp.* 13(1):182196, 1984.

[15] D. Nussbaum. Rectilinear p-piercing problems. In *Proc. 1997 Int. Symp. Symb. Algebraic Comp.*, pages 316–324, 1997.

[16] M. Segal. On piercing of axis-parallel rectangles and rings. In *Proc. 1997 Int. Symp. Symb. Algebraic Comp.*, pages 430–442, 1997.

[17] M. Segal. Lower bounds for covering problems. *J. Math. Model. Alg.* 1(1):17–29, 2002.

[18] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Disc. Comp. Geom.* 18:125–134, 1997.

[19] M. Sharir and E. Welzl. Rectilinear and polygonal $p$-piercing and $p$-center problems. In *Proc. 12th ACM Symp. Comp. Geom.*, pages 122–132, 1996.

[20] J.J. Sylvester. A question in the geometry of situation. *Quart. J. Math.* 1:79, 1857.

# New representation results for planar graphs

Farhad Shahrokhi
Department of Computer Science and Engineering, UNT
P.O.Box 13886, Denton, TX 76203-3886, USA farhad@cs.unt.edu

## Abstract

A universal representation theorem is derived that shows any graph is the intersection graph of one chordal graph, a number of co-bipartite graphs, and one unit interval graph. Central to the the result is the notion of the clique cover width which is a generalization of the bandwidth parameter. Specifically, we show that any planar graph is the intersection graph of one chordal graph, four co-bipartite graphs, and one unit interval graph. Equivalently, any planar graph is the intersection graph of a chordal graph and a graph that has clique cover width of at most seven. We further describe the extensions of the results to graphs drawn on surfaces and graphs excluding a minor of crossing number of at most one.

## 1 Introduction and Summary

Graph theory, geometry, and topology stem from the same roots. Representing graphs as the intersection graphs of geometric or combinatorial objects is highly desired in certain branches of combinatorics, discrete and computational geometry, graph drawing and information visualization, and the design of geographic information systems (GIS). A suitable intersection model not only provides a better understanding of the underlying graph, but it can also lead to computational advances. A remarkable result in this area is Koebe's (also Thurston's) theorem, asserting that every planar graph is the touching graph of planar disks. A similar result is due to Thomassen [12] who showed that every planar graph is the intersection graph of axis parallel boxes in $R^3$. Another noteworthy result is due to Gavril [7] who proved that every chordal graph (a graph with no chordless cycles) is the intersection graph of a collection of subtrees of a tree.

Any (strict) partially ordered set [14] $(S, <)$ has a directed acyclic graph $\hat{G}$ associated with it in a natural way: $V(G) = S$, and $ab \in E(G)$ if and only if $a < b$. The *comparability graph* associated with $(S, <)$ is the undirected graph which is obtained by dropping the orientation on edges of $\hat{G}$. The complement of a comparability graph is an *incomparability graph.* Incomparability graphs are well studied due to their rich structures and are known to be the intersection graph of planar curves [3]. A interesting result in this area is due to Pach and Törőcsik [9] who showed, given a set of straight line segments in the plane, there are four incomparability graphs whose edge intersections gives rise to the intersections of the segments. Moreover, recent work in combinatorial geometry has shown the connections between the intersection patterns of arbitrary planar curves and properties of incomparability graphs [6], [5].

An an interval graph is the intersection graph of a set of intervals on the real line [13]. It is easily seen that an interval graph is an incomparability graph. A unit interval graph is the intersection graph of a set of unit intervals.

Throughout this paper, $G = (V(G), E(G))$ denotes a connected undirected graph. Let $d \geq 1$, be an integer, and for $i = 1, 2, ..., d$ let $H_i$ be a graph with $V(H_i) = V$, and let $G$ be a graph with $V(G) = V$ and $E(G) = \cap_{i=1}^{d} E(G_i)$. Then we say $G$ is the *intersection graph* of $H_1, H_2, ..., H_d$, and write $G = \cap_{i=1}^{t} H_i$. A clique cover $C$ in $G$ is a partition of $V(G)$ into cliques. Throughout this paper, we will write $C = \{C_0, C_1, ..., C_t\}$ to indicate that $C$ is an ordered set of cliques. For a clique cover $C = \{C_0, C_1, ..., C_t\}$, in $G$, let the *width* of $C$, denoted by $W(C)$, denote $\max\{|j - i||xy \in E(G), x \in C_i, y \in C_j, C_i, C_j \in C\}$. The *clique cover width* of $G$ denoted by $CCW(G)$ is the smallest width all ordered clique covers in $G$. Note that $CCW(G) \leq BW(G)$, where $BW(G)$ denotes the bandwidth of $G$. A co-bipartite graph is the complement of a bipartite graph. Clearly, any co-bipartite graph is an incomparability graph.

### 1.1 Our Results

We recently proved the following result [11].

**Theorem 1** *Let $C$ be a clique cover in $G$ with $0 < W(C) \leq w, w \geq 1$. Then, there are $\lceil \log(w) \rceil + 1$ co-bipartite graphs $H_i, i = 1, 2, ..., \lceil \log(w) \rceil + 1$, and a unit interval graph $H_{\lceil \log(w) \rceil} + 2$, so that $G = \cap_{i=1}^{\lceil \log(w) \rceil + 2} H_i$.*

The main result in this paper is Theorem 5, which asserts any planar graph is the intersection graph of a chordal graph and a graph whose clique cover width is bounded by seven. The application of Theorem 1, then, gives another version of the result as stated in the abstract. Theorem 5 is obtained using the Universal Representation Theorem, or Theorem 2, which is interesting on its own, and asserts that any graph is intersection graph of a chordal graph and a graph whose clique cover width is bounded. Nonetheless, the upper bound on the clique cover width of the second graph depends on the properties of the tree decompositions of the original graph. Theorem 5 is further extended to graphs drawn on surfaces, and graphs excluding a minor with the crossing number of at most one.

## 2 Main Results

**Definition 1** *A tree decomposition [10] of a graph $G$ is a pair $(X, T)$ where $T$ is a tree, and $X = \{X_i | i \in V(T)\}$ is a family of subsets of $V(G)$, each called a bag, so that the following hold:*

- $\cup_{i \in V(T)} X_i = V(G)$
- *for any $uv \in E(G)$, there is an $i \in V(T)$ so that $v \in X_i$ and $u \in X_i$.*
- *for any $i, j, k \in V(T)$, if $j$ is on the path from $i$ to $k$ in $T$, then $X_i \cap X_k \subseteq X_j$.*

**Theorem 2** *(Universal Representation Theorem) Let $G$ be a graph and let $L = \{L_1, L_2, ..., L_k\}$ be a partition of vertices, so that for any $xy \in E(G)$, either $x, y \in L_i$ where $1 \leq i \leq k$, or, $x \in L_i, y \in L_{i+1}$, where, $1 \leq i \leq k-1$. Let $(X, T)$ be a tree decomposition of $G$. Let $t^* = \max_{i=1,2,...,k}\{|L_i \cap X_j| | j \in V(T)\}$. (Thus, $t^*$ is the largest number of vertices in any element of $L$ that appears in any bag of $T$). Then, there is a graph $G_1$ with $CCW(G_1) \leq 2t^* - 1$ and a chordal graph $G_2$ so that $G = G_1 \cap G_2$.*

**Proof.** For any $v \in V(G)$, let $X_v$ be the set of bags in $X$ that contains vertex $v$, and let $T_v$ be the subtree of $T$ on the vertex set $X_v$. Let $G_2$ be the intersection graph of these subtrees. Thus, for any $v, w \in V(G)$, $vw \in E(G_2)$, if $X_v \cap X_w \neq \emptyset$. It is well known that $G_2$ is chordal. See work of Gavril [7]. Now let $\omega$ be the largest clique in $G_2$ among all cliques whose vertices are entirely in $L_i$, for some $i = 1, 2, ..., k$. It follows from established properties on the tree decomposition that all vertices in $\omega$ should appear in one bag $B$ in $X$. Consequently, $|\omega| \leq |B \cap L_i| \leq t^*$. Next observe that for $i = 1, 2, ..., k$, $G_2[L_i]$ is chordal and hence perfect, and thus there must be at most $t^*$ disjoint independent sets $L_i^j, j = 1, 2, ..., t^*$ whose union is $L_i$. Now construct $G_1, V(G_1) = V(G)$, as follows: $E(G_1) = E(G) \cup E'$, where $E'$ is obtained by placing an edge between any vertex pair in each independent set ${L_i}^j$ for $i = 1, 2, , ..., k, j = 1, 2, ..., t^*$. Clearly, $G = G_1 \cap G_2$. In addition, for $i = 1, 2, ..., k$, $G_1[L_i]$ is covered with at most $t^*$ disjoint cliques, hence any ordering of these cliques will give rise to a clique cover $C$ of $G_1$ with $W(C) \leq 2t^* - 1$, since any edge $e \in E(G_1)$ either has both ends in one previously prescribed clique in $G_1[L_i]$, or must have end points in two consecutive elements in $L$. $\Box$.

The following definitions are from [2].

**Definition 2** *A maximal spanning forest of $G$ is a spanning forest $T$ that contains a spanning tree from each component of $G$. Thus, when $G$ is connected, any spanning tree of $G$ is also a maximal spanning forest. Let $T$ be a maximal spanning tree of $G$, and let $ab \in E(G) - E(T)$; The detour of $ab$ in $T$ is the unique $ab$ path in $T$. Let $e \in E(T)$. The edge remember number of $e$, denoted by $er(e, T, G)$, is the number of edges in $E(G) - E(T)$ whose detour contains $e$; Equivalently, $er(e, T, G)$ is the number of fundamental cycles in $G$ relative to $T$, that contain $e$. Similarly, for $v \in V(G)$, the vertex remember number number of $v$ denoted by $vr(v, T, G)$, is the number of edges in $E(G) - E(T)$ whose detour, or the fundamental cycle associated with it, contains $v$. To remedy technical issues, for any $e \in E(G) - E(T)$, we define $er(e, T, G) = 0$. The edge remember number and vertex remember number of $G$ in $T$, denoted by $er(G, T)$ and $vr(G, T)$, are the largest remember numbers overall edges in $E(T)$ and vertices in $V(T)$, respectively.*

**Definition 3** *Let $T$ be a maximal spanning tree of $G$, and let $\hat{T}$ be a forest that is obtained by inserting vertices of degree two to the edges of $T$. Thus, $\hat{T} = (V(T) \cup E(T), E(\hat{T}))$. Now, for any $v \in V(T)$ place $v$ in $X_v$, and for any $e = ab \in E(T)$ place $a$ and $b$ in $X_e$. Next, for any $e = ab \in E(G) - E(T)$, take one of $a$ or $b$, say $a$, and place it in $X_v$, for any $v$ which is on the unique $ab-$detour in $T$; Similarly, place $a$ in $X_e$ for any edge $e$ which is on the unique $ab-$detour in $T$. Finally, define, $\hat{X} = \{X_i | i \in V(T) \cup E(T)\}$.*

Bodlaender [2] showed the following.

**Theorem 3** *Let $T$ be a maximal spanning tree of $G$, and let $\hat{T}$ and $\hat{X}$ be as defined above. Then, $(\hat{X}, \hat{T})$ is a tree decomposition of $G$ whose width is at most $\max\{vr(G, T), er(G, T) + 1\}$.*

In light of the above result, we will refer to $(\hat{T}, \hat{X})$ (in definition 3) as a *tree decomposition of $G$ relative to $T$*. Note that the construction in definition 3 would allow the same vertex to appear in $X_v$ or $X_e$ more than once, where each appearance is associated with an end point of an edge $e \in E(G) - E(T)$, representing a distinct fundamental cycle containing $v$, or, $e$. With that in mind, we have , $|X_v| = vr(v, T, G) + 1$ and $|X_e| = er(e, T, G)+2$. However, when viewing $|X_v|$ and $|X_e|$ as sets, the duplicate members would be removed, thereby, $=$ would become $\leq$.

The following Lemma is extended from [2]. The notations and claims are slightly perturbed to exhibit additional properties of the construction of Bodlaender, that we will use later.

**Lemma 4** *Let $G$ be a plane graph, let $O$ be the set of all vertices in the outer boundary of $G$, let $H, V(H) = V(G)$ be a graph obtained by removing all edges in the outer boundary $G$. Let $T'$ be a maximal spanning forest of $H$ and let $(\hat{X}', \hat{T}')$ be a tree decomposition of $H$ relative to $T'$.*

*$(i)$ $T$ can be extended to a maximum spanning forest $T$ of $G$ so that $vr(v,T,G) \leq vr(v,T',H)+\Delta(G)$ and $er(e,T,G) \leq er(e,T',H)+2$, for all $v \in V(G)$ and $e \in E(T)$.*

*$(ii)$ $(\hat{X}', \hat{T}')$ can be extended to a tree decomposition $(\hat{X}, \hat{T})$ of $G$ relative to $T$ so that $|X_v \cap O| \leq |X'_v \cap O| + \Delta(G)$ and $|X_e \cup O| \leq |X'_e \cup O| + 2$ for all $v \in V(G)$ and $e \in E(T)$* .[1]

**Proof.** For $(i)$, let $K$ be graph with $V(K) = V(G)$ and $E(K) = E(T') \cup (E(G) - E(H))$, and note that the external face of $K$ is the same as external face of $G$. Extend $T'$ to a maximal spanning tree $T$ of $K$ by adding edges from $E(G) - E(H)$. Note that for any $e = xy \in E(K) - E(T)$, $x$ and $y$ must be on the boundary of $G$. Thus, the associated $xy$ detour $p$ in $T$ plus $e$ must form the boundary of a non-external face in $K$. Since any edge in $T$ is common to at most 2 non external faces, and each vertex in $T$ is common to at most $\Delta(G)$ many non-external faces, in $K$, it follows that for any $e \in E(T)$ and any $v \in V(G)$, $er(e,T,K) \leq 2$ and $vr(v,K,T) \leq \Delta(G)$. As $T$ is also a maximal spanning tree of $G$ and each fundamental cycle in $G$ is either a fundamental cycle of $K$ relative to $T$, or a fundamental cycle of $H$ relative to $T'$, we must have $er(e,G,T) \leq er(e,T',H)+er(e,T,K) \leq er(e,T'H)+2$, and $vr(v,G,T) \leq er(e,T',H) + vr(v,T,K) \leq vr(v,T',H)+\Delta(G)$.

$(ii)$ follows from $(i)$. In particular, note that additional 2 or $\Delta(G)$ fundamental edges that contribute to $vr(v,G,T)$ and $er(e,G,T)$, respectively, are those edges in $E(G)-E(T)$ that have both end points in $O$. Now obtain a tree decomposition of $G$ relative to $T$, by extending each bag of $\hat{T}'$, to a bag of $\hat{T}$ by the possible addition f one end point of such a fundamental edge, as described in definition 3. □

By a plane graph we mean an embedding of a planar graph in the plane. A plane graph is $1-$outer planar, if it is outer planar. For $k \geq 2$, a plane graph $G$ is $k-$outer planar, if after removal of all vertices (and edges incident to these vertices) in the external face of $G$, a $k-1$outer planar graph is obtained.

**Theorem 5** *Let $G$ be a planar graph, then, there is a graph $G_1$ with $CCW(G_1) \leq 7$ and a chordal graph $G_2$ so that $G = G_1 \cap G_2$.*

[1]In $(i)$ and $(ii)$ we follow the assumption that $er(e,T',G) = 0$ and $X'_e = \emptyset$, for $e \in E(T) - E(T')$.

**Proof.** Assume $G$ is $k-$outer planar. Thus, there are graphs $G = G_1, G_2, ..., G_k$ so that for $i = 1,2,...,k$, $G_i$ is $(k-i+1)-$outer planar, and $G_{i+1}$ is obtained by removing the vertices in the outer face of $G_i$. For $i = 1,2,...,k$, let $O_i$ denote the set of vertices on the outer face of $G_i$. Note that for $i = 1,2,...,k$, one can replace any vertex $v$ of degree $d \geq 4$ in the outer face of $O_i$ by a path $p_v$ of $d-2$ vertices of degree 3, so that $G$ is transformed to another $k-$outer planar graph $G'$. Specifically, for $i = 1,2,...,k$, let $O'_i$ denote the set of vertices corresponding to $O_i$, after this transformation. Note that $G'$ is $k-$outer planar and has maximum degree 3, let $G'_1 = G'$, and for $i = 2,...,k+1$, let $G'_i$ denote the graph that is obtained after removing all edges in the outer face of $G'_{i-1}$, and note that $G'_i$ is $(k-i+1)-$outer planar and of maximum degree 3. Note that $G'_{k+1}$ is acyclic and let $T_{k+1} = G'_{k+1}$. Clearly, $vr(v, T_{k+1}, T_{k+1}) = 0, er(e, T_{k+1}, T_{k+1}) = 0$, for any $x \in V(G)$, and any $e \in E(T_{k+1})$. Thus, for the tree decomposition $(\hat{X}_{k+1}, \hat{T}_{k+1})$ of $G_{k+1}$ relative to $T_{k+1}$, and bags $X_v, X_e$, $v \in V(G), e = ab \in E(T_{k+1})$, we have $|X_v| = 1$ (since $X_v = \{v\}$), and $|X_e| = 2$ (since $X_e = \{a,b\}$), respectively. Next, for $j = k, k-1, ...1$, let $T_j$ and $(\hat{X}_j, \hat{T}_j)$ be a maximal spanning forest and a tree decomposition of $G'_j$ relative to $T_j$, that are obtained by the application of Part $(i)$ and Part $(ii)$ of Lemma 4, to $T_{j+1}$ and $(\hat{X}_{j+1}, \hat{T}_{j+1})$, respectively. Thus, $(\hat{X}_1, \hat{T}_1)$ is a tree decomposition of $G'$. Then, one can show (by induction) that for any $j, i = k, k-1, ..., 1$, and any $X^j_v, X^j_e \in \hat{X}_j$ with $v \in V(G), e \in E(T_j)$

$|X^j_v \cap O'_i| = |X^{j-1}_v \cap O'_i|$ if $i \neq j$, whereas, $|X^j_v \cap O'_i| \leq 1 + \Delta(G'_j) \leq 1+3 = 4$ if $i = j$,

and

$|X^j_e \cap O'_i| = |X^{j-1}_e \cap O'_i|$ if $i \neq j$, whereas, $|X^j_e \cap O'_i| \leq 2+2 = 4$ if $i = j$.

Hence, for $i = 1,2,...,k$, and $X^1_v, X^1_e \in \hat{X}_1$ with $v \in V(G)$ and $e \in E(T_1)$, we have, $|X^1_v \cap O'_i| \leq 4$ and $|X^1_e \cap O'_i| \leq 4$. Next, for any $v \in V(G)$, contract all the vertices in $p_v$ to $v$, thereby, for $i = 1,2,...,k$ contracting $O'_i$ to $O_i$. For any bag $X^1_t \in \hat{X}_1$ with $t \in V(G) \cup E(T_1)$, let $Y_t = (X^1 - p_v) \cup \{v\}$. Now let $Y = \{Y_t | t \in V(G) \cup E(T_1)\}$. Since $G$ is a minor of $G'$, it follows that $(Y, T^1)$ is a tree decomposition of $G$ with the property that for any $Y_t \in Y$ with $t \in V(T_1) \cup E(T_1)$, and any $i = 1,2,...,k$, we have $|Y_t \cap O_i| \leq 4$. Now the result follows from Theorem 2, by taking $L = \{O_1, O_2, ..., O_k\}$. □.

Combining Theorems 1 and 5 we obtain the following.

**Theorem 6** *Let $G$ be a planar graph, then, there are co-bipartite graphs $G_1, G_2, G_3, G_4$, a unit interval graph $G_5$, and a chordal graph $G_6$ so that $G = \cap^6_{i=1} G_i$.*

### 2.1 Extensions

The result for planar graphs give rise to the following.

**Theorem 7** *Let $G$ be a graph of genus $g$. Then, there is an integer $c = O(\log(g))$, co-bipartite graphs $G_i, i = 1, 2, ..., c$, a unit interval graph $G_{c+1}$, and a chordal graph $G_{c+2}$ so that $G = \cap_{i=1}^{c+2} G_i$.*

**Proof Sketch.** One can show the claim by induction on $g$, where Theorems 5 and 6 establish the base of the induction. □

**Theorem 8** *Let $G$ be a graph that does not have as a minor, a graph $H$ whose crossing number is at most one. Then there is an integer $c = O(\log(C_H))$, co-bipartite graphs $G_i, i = 1, 2, ..., c$, a unit interval graph $G_{c+1}$, and a chordal graph $G_{c+2}$ so that $G = \cap_{i=1}^{c+2} G_i$. Here, $C_H = 20^{2(2|V(H_0)|+4|E(H_0)|)^5}$.*

**Proof Sketch.** It is known that any graph that does not have a minor $H$ of crossing number of at most one, can be obtained by taking the clique sum of a finite set of graphs, where each graph is either planar, or has a tree width of at most $C_H$ [10]. So $G = H_1 \bigoplus H_2 ... \bigoplus H_k$, where $\bigoplus$ stands for the clique sum operation, and for $i = 1, 2, ..., k$, each $H_i$ is either planar, or has a tree width of at most $C_H$. We prove the claim by induction on $k$. When $k = 1$ the result follows from Theorems 5, 2, 1, and the definition of $C_H$. Now assume that the claim is true for $k = t - 1$, let $k = t \geq 2$, and set $F = H_1 \bigoplus H_2 ... \bigoplus H_{t-1}$. Then, $G = F \bigoplus H_k$. By induction, $F = F_1 \cap F_2$, where $F_2$ is chordal and $CCW(F_1) \leq C_H$. Moreover, since $H_{t-1}$ is either planar, or has a tree width of at most $C_H$, by Theorem 5 we have $G_{t-1} = F_3 \cap F_4$, where $CCW(F_3) \leq 2C_H$ and $F_4$ is chordal. Now let $G_1 = F_1 \bigoplus F_3$, and $G_2 = F_2 \bigoplus F_4$, then, it is easy to verify that $G_2$ is chordal. To finish the proof, one can verify using properties of the clique cover width that, $CCW(G_2) \leq 2C_H$. Now the claim follows from Theorem 1. □

## 3 Computational Aspects

All constructions provided here can be done in polynomial time, with the exception of Theorem 8.

In [11] we have shown that if $G$ is the intersection graph of a chordal graph and a graph whose clique cover width is bounded by a constant, then $G$ can be separated with a splitting ratio of $1/3 - 2/3$, for a variety of measures, where the measure associated with the separator is "small". Consequently, the planar separator theorem [8] and its extensions follow from the representation results in this paper.

We highly suspect that the computation of the clique cover width is an NP$-$hard problem, due to its connection with the bandwidth problem.

## References

[1] Bodlaender H.L, A Tourist Guide through Treewidth. Acta Cybern. 11, 1993, 1-22.

[2] Bodlaender H., A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science, 209, 1998, 1-45.

[3] Golumbic M., Rotem D., Urrutia J., Comparability graphs and intersection graphs, Discrete Mathematics 43 (1), 1983, 37-6.

[4] Chan T., Polynomial-time approximation schemes for packing and piercing fat objects , Journal of Algorithms, 46(2), 2003, 178 - 189.

[5] Fox J. and Pach J., String graphs and incomparability graphs, Advances in Mathematics, 2012, 1381-1401.

[6] Fox J., Pach J., A separator theorem for string graphs and its applications, Combinatorics, Probability and Computing 19, 2010, 371-390.

[7] Gavril, F., The intersection graphs of subtrees in trees are exactly the chordal graphs, Journal of Combinatorial Theory, Series B 16, 1974, 47-56.

[8] Lipton R. J., Tarjan R.E. , A separator theorem for planar graphs, SIAM Journal on Applied Mathematics 36, 1979, 177-189

[9] Pach J., Törőcsik J., Some geometric applications of Dilworth's theorem, *Disc. Comput. Geometry*, 21, 1994, 1-7.

[10] Robertson N., Seymour, P. D. Graph minors III: Planar tree-width, Journal of Combinatorial Theory, Series B 36 (1), 1984, 49-64.

[11] Shahrokhi F., in preparation.

[12] Thomassen, C., Interval representations of planar graphs, Journal of Combinatorial Theory, Series B 40, 1986, 9-20.

[13] Trotter W.T., New perspectives on interval orders and interval graphs, in Surveys in Combinatorics, Cambridge Univ. Press, 1977, 237-286.

[14] Trotter, W.T., Combinatorics and partially ordered sets: Dimension theory, Johns Hopkins series in the mathematical sciences, The Johns Hopkins University Press, 1992.

# New sequential and parallel algorithms for computing $\beta$-spectrum [*]

Mirosław Kowaluk [†] Gabriela Majewska [‡]

## 1 Abstract

$\beta$-skeletons, a prominent member of the neighborhood graph family, have interesting geometric properties and a broad range of interesting applications. This paper focuses on computing $\beta$-spectrum, a labeling of the edges of the Delaunay Triangulation, $DT(V)$, which makes it possible to quickly find the lune-based $\beta$-skeleton of $V$ for any query value $\beta \in [1, 2]$. We consider planar $n$ point sets $V$ with $L_p$ metric, $1 < p < \infty$ and present two new algorithms: a $\mathcal{O}(n \log^2 n)$ time sequential, and a $\mathcal{O}(\log^4 n)$ time parallel $\beta$-spectrum labeling. The parallel algorithm uses $\mathcal{O}(n)$ processors in the CREW-PRAM model.

## 2 Introduction

$\beta$-skeletons [KR85] in $R^2$ belong to the family of proximity graphs, geometric graphs in which two vertices (points) produce an edge if and only if they satisfy particular geometric requirements.

Well-known examples of these graphs include *Gabriel Graph* (1-beta skeleton), defined by Gabriel and Sokal [GS69]; they can be computed from the Delaunay Triangulation of the point set in linear time.

*Relative Neighborhood Graph RNG*, $\beta$-skeletons for $\beta = 2$, were introduced by Toussaint [Tou80] in the context of pattern recognition. Supowit [Su83] designed the first $\mathcal{O}(n \log n)$ time algorithm and Jaromczyk and Kowaluk [JK87] showed how to construct $RNG$ from Delaunay Triangulation $DT$ for the $l_p$ metric ($1 < p < \infty$) in $\mathcal{O}(n\alpha(n))$ time, where $\alpha$ is a functional inverse of Ackermann's function; the algorithm is based on the concept of elimination paths that will play a role in this paper as well. This result was later improved to $\mathcal{O}(n)$ time [JKY89] for $\beta$-skeletons for $1 \leq \beta \leq 2$.

Two different forms of $\beta$-neighborhoods have been studied for $\beta > 1$ (see e.g. [ABE98, E02]), leading to two different families of graphs: lune-based $\beta$-skeletons and circle-based $\beta$-skeletons. In this work, we focus on lune-based $\beta$-skeletons.

With each pair of vertices $u, v$ we can associate the largest value $\beta$, called $\beta$-value of $uv$, such that the edge $uv$ belongs to the $\beta$-skeleton. A set of all edges spanned by set of points $V$, each labeled with its $\beta$-value is called a $\beta$-spectrum of $V$. Hurtado, Liotta and Meijer [HLM02] showed an algorithm which computes both lune-based and circle-based $\beta$-spectrum for a set of $n$ points in $\mathcal{O}(n^2)$ time.

On the other hand, there are very few parallel algorithms for proximity graphs [AL93, CG92, CK10]. In particular, parallel algorithms for $\beta$-skeletons have not been studied and this paper makes an initial effort to fill this gap.

We will present two new algorithms computing $\beta$-spectrum for $1 \leq \beta \leq 2$: a sequential algorithm with a $\mathcal{O}(n \log^2 n)$ running time, and a parallel one that takes $\mathcal{O}(\log^4 n)$ time and uses $\mathcal{O}(n)$ processors in the CREW PRAM model.

## 3 Basic definitions and facts

We consider point sets in the two-dimensional plane $R^2$ with the $L_p$ metric (with distance function $d_p$), where $1 < p < \infty$.

**Definition 1** *For a given set of points $V = \{v_1, v_2, \ldots, v_n\}$ in $R^2$ and parameters $\beta \geq 0$ and $p$ we define graph $G_\beta(V)$ – called a lune-based $\beta$-skeleton – as follows: two points $v_1, v_2$ are connected with an edge if and only if no point from $V \setminus \{v_1, v_2\}$ belongs to the set $N_p(v_1, v_2, \beta)$ where:*

1. *for $\beta = 0$ the set $N_p(v_1, v_2, \beta)$ is simply a segment $v_1v_2$;*
2. *if $0 < \beta < 1$ then $N_p(v_1, v_2, \beta)$ is an intersection of two discs in $l_p$, each with radius $\frac{|v_1v_2|}{2\beta}$, whose boundaries contain both $v_1$ and $v_2$;*
3. *for $1 \leq \beta < \infty$ the set $N_p(v_1, v_2, \beta)$ is an intersection of two $l_p$ discs, each with radius $\frac{\beta|v_1v_2|}{2}$, whose centers are in points $(\frac{\beta}{2})v_1 + (1 - \frac{\beta}{2})v_2$ and in $(1 - \frac{\beta}{2})v_1 + (\frac{\beta}{2})v_2$ respectively;*
4. *for $\beta = \infty$, $N_p(v_1, v_2, \beta)$ is the unbounded strip between lines perpendicular to the segment $v_1v_2$ and containing $v_1$ and $v_2$.*

**Definition 2** *For an edge $v_1v_2$ let $\bar{\beta}$ be the largest real number such that no point from $V \setminus \{v_1, v_2\}$ belongs to $N_p(v_1, v_2, \bar{\beta})$. We call this $\bar{\beta}$ a $\beta$-value for $v_1v_2$. The set of all edges spanned by $V$, each labeled by its $\beta$-value is called a $\beta$-spectrum of $V$. Additionally, the $\beta$-spectrum of $V$ for $\beta \in [x, y]$ is the subset*

[*]This research is supported by the ESF EUROCORES programme EUROGIGA, CRP VORONOI.

[†]Institute of Informatics, University of Warsaw, kowaluk@mimuw.edu.pl

[‡]Institute of Informatics, University of Warsaw, gm248309@students.mimuw.edu.pl

*of $\beta$-spectrum of $V$ such that $\beta$-values for all edges in this subset satisfy $x \leq \beta$-value$\leq y$.*

Let us assume that points in $V$ are in general position. The following fact connecting $\beta$-skeletons with the minimum spanning tree $MST(V)$ and Delaunay triangulation $DT(V)$ of $V$ is due to Kirkpatrick and Radke [KR85]:

**Fact 1** *For $1 \leq \beta \leq \beta' \leq 2$ the following inclusions hold true: $RNG(V) \subseteq G_{\beta'}(V) \subseteq G_{\beta}(V) \subseteq GG(V) \subseteq DT(V)$.*

## 4 $\beta$-spectrum for $\beta \in [1, 2]$

This section describes two algorithms (sequential and parallel) for computing $\beta$-spectrum for $1 \leq \beta \leq 2$ by using generalized elimination paths defined as follows:

**Definition 3** *Two triangles $t_1$ and $t_2$ in $DT(V)$ are called neighbors in $DT(V)$ if they share a common edge.*

*For each vertex $v$ and edge $e$ that belong to the same triangle $t_0 \in DT(V)$, we define inductively a* generalized elimination path *as a sequence of edges in $DT(V)$ that starts with $e_0 = e$. Every pair of consecutive edges in this sequence belong to a triangle in $DT(V)$. Inductively, let us assume that we have already constructed a path $e_0, \ldots, e_i$ and for $i > 0$ let $t_i$ in $DT(V)$ be defined by edges $e_{i-1}$ and $e_i$. Furthermore, for $i \geq 0$ let $t_{i+1}$ and $t_i$ be neighbors that share edge $e_i$ in $DT(V)$. As $e_{i+1}$, we select a longer (if there exists) of the two, different from $e_i$, edges of triangle $t_{i+1}$. When we reach a triangle and an edge, such that its neighbor triangle does not exist, or we reach a base of an isosceles triangle, or we reach an already visited triangle, then the construction of the elimination path terminates. We call the last edge of the sequence a* root.

By merging generalized paths, we form *generalized elimination trees GET* and a generalized elimination forest $GEF(V)$.

We can construct the forest of generalized elimination trees in linear time.

Based on [JKY89], we know that for $\beta \in [1, 2]$, every edge that does not belong to the $\beta$-skeleton, belongs to some generalized elimination path. Therefore, to compute the $\beta$-value for edge $e$, it suffices to locate in the generalized elimination tree the beginning (vertex) of a generalized elimination path corresponding to the largest empty region $N_p$ for edge $e$.

Recall that GET is a tree of the elimination paths and the polygon of GET is defined as the union of all the $DT(V)$ triangles that are crossed by elimination paths.

**Definition 4** *The central polygon for GET is defined as this part of the polygon of GET that corresponds to the $DT(V)$ triangles intersected by the generalized elimination path in GET leading to the middle vertex on the boundary of the polygon of GET. The order of vertices of the polygon of GET is defined by starting with the leftmost vertex of the root edge and moving clockwise around the polygon (see Figure 1).*

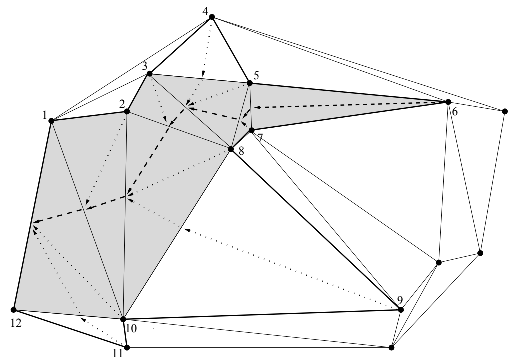


Figure 1: Generalized elimination tree and central polygon

We remove from the sequence of vertices the vertices that belong to the just constructed central polygon and recur on the two remaining sets of vertices constructing recursively a *central polygons tree CPT*, the tree of central polygons connected by their common edges. The common edge of a central polygon $C$ and its parent $D$ in $CPT$ is called the *base of C*.

**Lemma 1** *All central polygons can be constructed in $\mathcal{O}(n)$ time.*

As the next step, for each central polygon, we construct its *logarithmic structure of Voronoi Diagrams*, as follows. The vertices of the central polygon are numbered starting from the leftmost vertex of the root or the base edge in the $CPT$. First, Voronoi Diagrams are constructed for individual vertices on the boundary of this central polygon. Then, Voronoi diagrams are constructed for sets of vertices with numbers from $[s2^k, (s+1)2^k]$ where $0 \leq s \leq \lfloor \frac{n}{2^k} \rfloor$, $0 \leq k \leq \lfloor \log n \rfloor$.

**Lemma 2** *Constructing the logarithmic structure of Voronoi Diagrams for a given central polygon takes $\mathcal{O}(n \log n)$ time.*

Now we are ready to present the algorithm. First, let us consider the Euclidean metric. The bisector of edge $ab$ contains vertices of the lune $N_2(a, b, \beta)$ for all $1 \leq \beta \leq 2$. This bisector and the line containing the segment $ab$ divide the plane into four parts. We consider logarithmic structure of Voronoi Diagrams for each closed quarter plane. Let $Q$ be a given closed quarter plane and let $c$ be the endpoint of edge $ab$

such that $c \in Q$. We search for the closest to $c$ intersection point $P(a, b, Q)$ of the edge $ab$ and the Voronoi Diagram of $Q \cap V$, if it exists.
To this end, we analyze the boundary of the Voronoi region for $c$ and elements of the logarithmic structure of Voronoi Diagrams for vertices of analyzed central polygon belonging to the quarter plane $Q$ (we consider diagrams containing maximum number of centers such that intersection of the sets of centers of diagrams is empty).

For each analyzed Voronoi Diagram in the logarithmic structure, we obtain one candidate for $\beta$-value. Starting at the point where the previous analysis has terminated, we traverse Voronoi Diagrams in one direction: we analyze vertices of the central polygon from the point $c$ towards the intersection of the boundary of the central polygon with the bisector of the edge $ab$. All the tests take a total of $\mathcal{O}(n \log n)$ time. Additionally, it is necessary to analyze all of the $\mathcal{O}(\log n)$ central polygons intersected by the bisector of $ab$ and belonging to the same $CPT$. The minimum of all the $\beta$-value candidates for the given edge is selected.

For metrics $L_p$, when $p \neq 2$, the line $k$ passing through vertices of $N_p(a, b, \beta)$ does not have to be the bisector of $ab$; the direction of $k$ depends on the $\beta$. However, our algorithm does not depend on the direction of the line $k$ and the necessary information, such as a position of the analyzed centre with respect to $k$, point $P(a, b, Q)$, can be precomputed during analyzing consecutive regions of the logarithmic structure of Voronoi Diagrams.

**Theorem 3** *$\beta$-spectrum for a set $V$ of $n$ points, where $1 \leq \beta \leq 2$ and $1 < p < \infty$ can be computed in a $\mathcal{O}(n \log^2 n)$ time.*

Now we will sketch a $\mathcal{O}(n)$ processors CREW-PRAM algorithm for computing the $\beta$-spectrum. We start with constructing the Delaunay Triangulation of $V$; this step takes $\mathcal{O}(\log^2 n)$ time. Next, we create $GEF(V)$ and divide polygons corresponding to the trees into central polygons; this step takes $\mathcal{O}(\log^2 n)$ time. Then, we build the logarithmic structure of the Voronoi Diagrams for all of these central polygons in a $\mathcal{O}(\log^3 n)$ time.

An additional data structure is needed to quickly search for candidate $\beta$-values in all of the Voronoi Diagrams. Any edge of a generalized elimination path in a central polygon is called *a diagonal* of this polygon. We are interested in finding Voronoi regions that intersect the respective diagonal of the central polygon. To this end, we compute the convex hull of the set of centers of the Voronoi Diagram [AG86] and the intersection of the diagram edges with the boundary of this convex hull.

With binary search for each edge of the diagrams, we can find in $\mathcal{O}(\log n)$ time a sequence of the centers that belong to regions intersecting the border of the convex hull and the interior of the central polygon. We divide the set of centers into halves. Edges that separate the corresponding two sets of the Voronoi regions are ordered with pointer-jumping. We repeat this procedure until all the regions have been separated.

**Lemma 4** *Let $ab$ be a diagonal of a central polygon and $Q$ be the part of the plane bounded by lines passing through $ab$ and vertices of $N_p(a, b, \beta)$. For each diagram in the logarithmic structure of Voronoi Diagrams a data structure for searching for points $P(a, b, Q)$ can be created in a $\mathcal{O}(\log^2 n)$ time. Each query, the location of a point $P(a, b, Q)$ takes a $\mathcal{O}(\log^2)$ time.*

For each diagonal $ab$ we have to check $\mathcal{O}(\log n)$ diagrams in the logarithmic structure of the Voronoi Diagrams. The next central polygon which can affect the $\beta$-value can be found in a $\mathcal{O}(\log n)$ time.

**Theorem 5** *In PRAM-CREW model, $\beta$-spectrum, $1 \leq \beta \leq 2$, can be computed with $\mathcal{O}(n)$ processors in a $\mathcal{O}(\log^4 n)$ time.*

**Acknowledgements**
The authors would like to thank Jerzy W. Jaromczyk for important discussions and comments.

## References

[AL93] S.G. Akl, K.A. Lyons, *Parallel Computational Geometry*, Prentice Hall, 1993

[ABE98] A.B. Amenta, M.W. Bern, D. Eppstein, *The crust and the $\beta$-skeleton: combinatorial curve reconstruction*, *Graphical Models Image Processing*, 60/2 (2), 1998, 125-135

[AG86] M.J. Atallah and M. T. Goodrich, *Eficient parallel solutions to some geometric problems*, *J. Parallel Distrib. Comput.*, 3, 1986, 293-327

[CG92] R. Cole, M.T. Goodrich, *Optimal parallel algorithms for polygon and point-set problems*, *Algorithmica*, tom 7, 1992, 3-23

[CK10] M.Connor, P.Kumar, *Fast Construction of k-Nearest Neighbor Graphs for Point Clouds*, *IEEE Transactions on Visualization and Computer Graphics*, issue 4, 2010, 599-608

[E02] D. Eppstein, *$\beta$-skeletons have unbounded dilation*, *Computational Geometry*, volume 23, 2002, 43-52

[GS69] K.R. Gabriel, R.R. Sokal, *A new statistical approach to geographic variation analysis*, *Systematic Zoology* 18, 1969, 259-278

[HLM02] F. Hurtado, G. Liotta, H. Meijer, *Optimal and suboptimal robust algorithms for proximity graphs*, *Computational Geometry*, North Holland, Amsterdam, 1985, 217-248

[JK87] J.W. Jaromczyk, M. Kowaluk, *A note on relative neighborhood graphs*, *Proceedings 3rd Annual Symposium on Computational Geometry*, Canada, Waterloo, ACM Press, 1987, 233-241

[JKY89] J.W. Jaromczyk, M. Kowaluk, F. Yao, *An optimal algorithm for constructing $\beta$-skeletons in $L_p$ metric*, manuscript, 1989

[KR85] D.G. Kirkpatrick, J.D. Radke, *A framework for computational morphology*, *Computational Geometry*, North Holland, Amsterdam, 1985, 217-248

[Su83] K.J. Supowit, *The relative neighborhood graph, with an aplication to minimum spanning trees*, *Journal of the ACM* volume 30, issue 3, 1983, 428-448

[Tou80] G.T. Toussaint, *The relative neighborhood graph of a finite planar set*, *Pattern Recognition* 12, 1980, 261-268

# Voronoi Diagrams from Distance Graphs [*]

Mario Kapl[†] Franz Aurenhammer[‡] Bert Jüttler[†]

## Abstract

We present a new type of Voronoi diagram in $\mathbb{R}^2$ that respects the anisotropy exerted on the plane by a given distance graph. It is based on a metric obtained by smoothly and injectively embedding of $\mathbb{R}^2$ into $\mathbb{R}^m$, and a scalar-valued function for re-scaling the distances.

A spline representation of the embedding surface is constructed with the Gauß-Newton algorithm, which approximates the given distance graph in the sense of least squares. The graph is required to satisfy the generalized polygon inequality.

We explain a simple method to compute the Voronoi diagrams for such metrics, and give conditions under which Voronoi cells stay connected. Several examples of diagrams resulting from different metrics are presented.

## 1 Introduction

The Voronoi diagram of a given set of sites is a powerful and popular concept in geometry which possesses a wide range of applications, e.g. to motion planning, geometrical clustering and meshing [2]. Beside the classical Euclidean Voronoi diagram there exists a large number of generalizations of this structure.

Two examples relevant for the present note are the anisotropic Voronoi diagrams described in [7] and [9]. For each point $p$ (say in $\mathbb{R}^2$), a different metric is defined which specifies the distances to all other points, as seen from $p$. However, the distance between two arbitrary points does *not* define a metric, because either the triangle inequality or the symmetry is violated. This is no serious hinderance for computing the anisotropic Voronoi diagram, though.

A different approach is followed in [8]. A Voronoi diagram on a parametric surface is generated, by taking geodesic distances between the points on the surface. The corresponding structure in the parameter domain of the surface is a Voronoi diagram which is possibly anisotropic. The computation of this type of diagram is rather expensive, as geodesic distances have to be computed frequently.

In this note we introduce a new metric framework on $\mathbb{R}^2$, called scaled embedding-generated (SEG) metrics, and use it to define a class of anisotropic Voronoi diagrams. It is based on a smooth one-to-one embedding of $\mathbb{R}^2$ into $\mathbb{R}^m$, for $m \geq 2$, and a scalar-valued scaling function. The construction of SEG metric Voronoi diagrams has several advantages. We have only one distance function for all points, which indeed defines a metric on $\mathbb{R}^2$, and the computation of distances is fast and simple. Also, some properties of such diagrams can be derived from the properties of the Euclidean Voronoi diagram in $\mathbb{R}^2$ and $\mathbb{R}^3$.

## 2 Preliminaries

We recall some basic concepts needed in the subsequent considerations.

**Definition 1** The *medial axis* of a set $\mathbf{X} \subset \mathbb{R}^m$ is the (closure of the) set of all points in $\mathbb{R}^m$ that have at least two closest points in $\mathbf{X}$.

**Definition 2** [1, 6] The *local feature size* at a point $\mathbf{p} \in \mathbf{X}$, denoted by $\mathrm{LFS}(\mathbf{p})$, is the Euclidean distance from $\mathbf{p}$ to the nearest point of the medial axis of $\mathbf{X}$.

**Definition 3** [1, 6] Let $\mathbf{P_x} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ be a finite subset of $\mathbf{X}$. We call $\mathbf{P_x}$ an *$\varepsilon$-sample* of $\mathbf{X}$ if for each point $\mathbf{p} \in \mathbf{X}$ there is a sample point $\mathbf{x}_i \in \mathbf{P_x}$ with $||\mathbf{p} - \mathbf{x}_i|| \leq \varepsilon \cdot \mathrm{LFS}(\mathbf{p})$.

**Definition 4** Let $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots\}$ be a finite set of points (called sites) in $\mathbb{R}^m$. For a given metric $D$ on $\mathbb{R}^m$, we define the *Voronoi cell* of a site $\mathbf{p}_i \in \mathbf{P}$ as the open set

$$V_D^i(\mathbf{P}) = \{\mathbf{p} \in \mathbb{R}^m \mid D(\mathbf{p}, \mathbf{p}_i) < D(\mathbf{p}, \mathbf{p}_j) \text{ for all } j \neq i\}.$$

The *Voronoi diagram* $V_D(\mathbf{P})$ is given by the complement of all Voronoi cells in $\mathbb{R}^m$,

$$V_D(\mathbf{P}) = \mathbb{R}^m \setminus \left( \bigcup_i V_D^i(\mathbf{P}) \right).$$

We denote the Voronoi diagram with respect to the Euclidean metric by $V(\mathbf{P})$. A Voronoi diagram is called *orphan-free* if all its Voronoi cells are connected. In the case of the Euclidean metric, the diagram is always orphan-free, because its regions are

[*]This work has been supported by the ESF EUROCORES Programme EuroGIGA - Voronoi, Austrian Science Foundation (FWF).

[†]Institute of Applied Geometry, Johannes Kepler University, Linz, Austria, {mario.kapl|bert.juettler}@jku.at

[‡]Institute for Theoretical Computer Science, University of Technology, Graz, Austria, auren@igi.tu-graz.ac.at.

intersections of open halfspaces of $\mathbb{R}^m$, and thus are convex polyhedra.

## 3 The SEG metric framework

We now introduce the metric framework we would like to work with.

**Definition 5** Let $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}^m$, for $m \geq 2$, be a continuous one-to-one embedding, with $\mathbf{x}(u, v) = (x_1(u, v), \ldots, x_m(u, v))$. In addition, let $r \mapsto d(r)$, for $r \geq 0$, be a scalar-valued scaling function with the following properties:

- $d(0) = 0$
- $d(r) > 0$, for $r > 0$
- $d'(r) \geq 0$, for $r \geq 0$
- $d(r)/r$ is monotonically decreasing, for $r > 0$

We define the distance $D$ between two points $\mathbf{u}_1 = (u_1, v_1)$ and $\mathbf{u}_2 = (u_2, v_2)$ in $\mathbb{R}^2$ as

$$D(\mathbf{u}_1, \mathbf{u}_2) = d(||\mathbf{x}(u_1, v_1) - \mathbf{x}(u_2, v_2)||). \qquad (1)$$

**Theorem 6** *The distance $D$ given by* (1) *defines a metric on $\mathbb{R}^2$.*

**Proof.** For all $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \in \mathbb{R}^2$, the distance $D$ has to satisfy the following conditions:

(i) $D(\mathbf{u}_1, \mathbf{u}_2) \geq 0$,
(ii) $D(\mathbf{u}_1, \mathbf{u}_2) = 0$ iff $\mathbf{u}_1 = \mathbf{u}_2$,
(iii) $D(\mathbf{u}_1, \mathbf{u}_2) = D(\mathbf{u}_2, \mathbf{u}_1)$, and
(iv) $D(\mathbf{u}_1, \mathbf{u}_3) \leq D(\mathbf{u}_1, \mathbf{u}_2) + D(\mathbf{u}_2, \mathbf{u}_3)$.

Conditions (i) and (iii) are trivially fulfilled. To show condition (ii), we have to use the fact that the embedding $\mathbf{x}(u, v)$ has no self-intersections, since the map $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}^m$ is one-to-one. The triangle inequality (iv) remains to be shown. For the sake of brevity we denote the Euclidean distance $||\mathbf{x}(\mathbf{u}_i) - \mathbf{x}(\mathbf{u}_j)||$ by $l_{i,j}$. Since the Euclidean metric satisfies the triangle inequality, we have

$$l_{1,3} \leq l_{1,2} + l_{2,3}.$$

Now we distinguish two cases.
*Case* 1: $(l_{1,3} \leq l_{1,2})$ or $(l_{1,3} \leq l_{2,3})$.
Since $d'(r) \geq 0$ for $r \geq 0$ we have

$$d(l_{1,3}) \leq d(l_{1,2}) \text{ or } d(l_{1,3}) \leq d(l_{2,3}),$$

$$d(l_{1,3}) \leq d(l_{1,2}) + d(l_{2,3}).$$

This shows the triangle inequality (iv).
*Case* 2: $l_{1,3} > l_{1,2}$ and $l_{1,3} > l_{2,3}$.
Since $d(r)/r$ is monotonic decreasing we know that

$$\frac{d(l_{1,3})}{l_{1,3}} \leq \frac{d(l_{1,2})}{l_{1,2}} \text{ and } \frac{d(l_{1,3})}{l_{1,3}} \leq \frac{d(l_{2,3})}{l_{2,3}}.$$

Now we have

$$\begin{aligned} d(l_{1,2}) + d(l_{2,3}) &= l_{1,2}\frac{d(l_{1,2})}{l_{1,2}} + l_{2,3}\frac{d(l_{2,3})}{l_{2,3}} \geq \\ \geq\ & l_{1,2}\frac{d(l_{1,3})}{l_{1,3}} + l_{2,3}\frac{d(l_{1,3})}{l_{1,3}} = (l_{1,2} + l_{2,3})\frac{d(l_{1,3})}{l_{1,3}} \geq \\ \geq\ & l_{1,3}\frac{d(l_{1,3})}{l_{1,3}} = l_{1,3}, \end{aligned}$$

which proves the triangle inequality (iv). □

We will call $D$ the *scaled embedding-generated (SEG) metric* in the sequel. For $m = 2$ or $m = 3$, the embedding $\mathbf{x}(u, v)$ is a parametric surface without self-intersections in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively. Two examples of possible scaling functions are

$$d(r) = ar \quad \text{or} \quad d(r) = a \ln(br + 1) \qquad (2)$$

for suitable constants $a, b > 0$.

With the help of the metric $D$, we can define generalized disks with radius $r > 0$ and center $\mathbf{c} \in \mathbb{R}^2$,

$$B_r(\mathbf{c}) = \{\mathbf{p} \in \mathbb{R}^2 \mid D(\mathbf{p}, \mathbf{c}) \leq r\}.$$

Clearly, for embedding dimension $m = 2$, these disks are topological disks, by the properties of $\mathbf{x}(u, v)$. Moreover, we have:

**Lemma 7** *Let $m = 3$, $\mathbf{c} \in \mathbb{R}^2$, and $\bar{r} = d^{-1}(r)$. If $\bar{r} < LFS(\mathbf{x}(\mathbf{c}))$, then the disks $B_r(\mathbf{c})$ are topological disks (and generalized circles are topological circles).*

## 4 Fitting SEG metrics to distance graphs

We now explain a method for computing suitable SEG metrics. The idea is to construct a spline embedding $\mathbf{x}(u, v)$ which approximates a given *distance graph* $G$ on $n$ points in the unit square $[0, 1]^2$ of the parameter domain $\mathbb{R}^2$.

To achieve high accuracy in the approximation, we require $G$ to satisfy the *generalized polygon inequality*, that is, for each edge $(p, q)$ the associated length $L_{p,q}$ is at most the length of any existing path in $G$ from $p$ to $q$.

For simplicity, the scaling function is temporarily set to the identity, $d(r) = r$.

We will construct an embedding surface $\mathbf{x}(u, v) = (x_1(u, v), \ldots, x_m(u, v))$ with $m \geq 3$, where the first two coordinate functions are the linear functions $x_1(u, v) = c^{(1)} u$ and $x_2(u, v) = c^{(2)} v$. The remaining coordinate functions $x_i(u, v)$ for $i \geq 3$ are given by B-spline functions of degree $(p_1, p_2)$,

$$x_i(u, v) = \sum_{j=0}^{n_1} \sum_{k=0}^{n_2} c_{j,k}^{(i)} M_j^{p_1}(u) N_k^{p_2}(v)$$

with $c_{j,k}^{(i)} \in \mathbb{R}$. The basic functions $(M_j^{p_1}(u))_{j=0,\ldots,n_1}$ and $(N_k^{p_2}(u))_{k=0,\ldots,n_2}$ are B-splines of degree $(p_1, p_2)$

with respect to the open knot sequences $\mathcal{S} = (s_j)_{j=0,\ldots,n_1+p_1+1}$ and $\mathcal{T} = (t_k)_{k=0,\ldots,n_1+p_1+1}$, respectively. Now we compute the unknown coefficients $\mathbf{c} = (c^{(1)}, c^{(2)}, c^{(3)}_{0,0}, \ldots, c^{(3)}_{n_1,n_2})$ by solving the minimization problem

$$\mathbf{c} = \arg\min \sum_{(p,q)\in G} \Big( \underbrace{||\mathbf{x}(p) - \mathbf{x}(q)||^2 - L^2_{p,q}}_{R_{p,q}(\mathbf{c})} \Big)^2.$$

Since this optimization problem is non-linear and the objective function is a sum of squares, we use the Gauß-Newton algorithm to solve it. For each iteration step, we minimize the objective function

$$\Big( \sum_{(p,q)\in G} (R_{p,q}(\mathbf{c}^0) + \nabla R_{p,q}(\mathbf{c}^0)(\Delta\mathbf{c} - \mathbf{c}^0))^2 \Big) + \omega ||\Delta\mathbf{c} - \mathbf{c}^0||^2, \quad (3)$$

which includes a Tikhonov regularization term, with respect to $\Delta\mathbf{c}$.

In the objective function (3), the vector $\mathbf{c}^0$ denotes the solution from the last step, $\Delta\mathbf{c}$ is the update, and $\nabla R_{p,q}$ is the row vector given by the partial derivatives of $R_{p,q}$ with respect to the control points $\mathbf{c}_{j,k}$. In addition, $\omega > 0$ is the parameter for the Tikhonov regularization term.

The obtained embedding $\mathbf{x}(u, v)$ has no self-intersections, as long as none of the coefficients $c^{(1)}$ and $c^{(2)}$ becomes zero in the optimization process. To avoid this case, or the case that one of them is close to zero, we allow the user to specify these coefficients in our implementation. We have used this approach in all the examples presented below.

The use of the two linear functions above can be seen as some kind of regularization. By using sufficiently small coefficients $c^{(1)}$ and $c^{(2)}$ (combined with a sufficiently large dimension of the embedding), one can minimize the influence of this regularization while still avoiding self-intersections of the embedding. The remaining coefficients $c^{(i)}_{j,k}$ of the initial solutions were chosen randomly from the interval $[-\frac{1}{10}, \frac{1}{10}]$.

Note that different initial solutions give different results. In our experience, however, the obtained different solutions were approximations of similar quality of the distance graph $G$. Moreover, we noticed that for most of our tested distance graphs an embedding of $\mathbb{R}^2$ into $\mathbb{R}^m$ for $m \in \{3, 4, 5\}$ leads to a satisfactory result. It seems, that as long as the generalized polygon inequality is fulfilled, the distance graph is usually ‘almost exactly’ approximated by the produced embedding. But especially in the case of a distance graph with a high valency for each point, an embedding into a higher dimension could probably be needed.

The resulting embedding $\mathbf{x}(u, v)$ induces an SEG metric on the unit square $[0, 1]^2$. By extending this restricted embedding to a continuous one-to-one embedding $\mathbf{x} : \mathbb{R}^2 \to \mathbb{R}^m$, we obtain an SEG metric on $\mathbb{R}^2$, which accurately approximates a given distance graph.

## 5 SEG metric Voronoi diagrams

We will now use the metric $D$ in (1) to construct a respective Voronoi diagram in $\mathbb{R}^2$, in a simple way.

Let $\mathbf{P} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots\}$ be a set of sites in $\mathbb{R}^2$. The SEG metric Voronoi diagram $V_D(\mathbf{P})$ for $D$ can be generated in the following way. Given the sites $\mathbf{u}_i \in \mathbb{R}^2$, we first calculate the corresponding points $\mathbf{x}_i = \mathbf{x}(\mathbf{u}_i)$ on the embedding $\mathbf{x}(u, v)$, which has been used to define $D$. Then we compute, for the obtained set of points $\mathbf{P}_\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ in $\mathbb{R}^m$, their Euclidean Voronoi diagram $V(\mathbf{P}_\mathbf{x})$. By intersecting the resulting Voronoi cells with the embedding surface $\mathbf{x}(u, v)$, we obtain a Voronoi diagram on $\mathbf{x}(u, v)$, which defines for the corresponding parameter values $(u, v) \in \mathbb{R}^2$ the desired SEG metric Voronoi diagram $V_D(\mathbf{P})$ in $\mathbb{R}^2$.

This approach is similar to that in [3], who showed that the anisotropic Voronoi diagram of [9] can be obtained by intersecting a so-called power diagram in $\mathbb{R}^5$ with a suitable surface. (Power diagrams are generalized Voronoi diagrams whose cells are still convex polyhedra; see e.g. [2].)

Before giving examples of SEG metric Voronoi diagrams constructed with our approach, let us consider some conditions under which $V_D(\mathbf{P})$ is orphan-free. Disconnectedness of a Voronoi cell in $V_D(\mathbf{P})$ means that the corresponding $m$-dimensional polyhedral cell in the Euclidean Voronoi diagram intersects the embedding surface more than once.

Clearly, $V_D(\mathbf{P})$ is orphan-free in the case $m = 2$, since $\mathbf{x}(u, v)$ then is a smooth one-to-one embedding into $\mathbb{R}^2$, and the Euclidean Voronoi diagram is always orphan-free. We further have:

**Lemma 8** *Let $m = 3$, and assume that $\mathbf{x}(u, v)$ is $C^2$-smooth. If the set of sites $\mathbf{x}(\mathbf{P})$ is a 0.18-sample of the surface $\mathbf{X} = \mathbf{x}(\mathbb{R}^2)$, then the resulting diagram $V_D(\mathbf{P})$ is orphan-free.*

**Proof.** It is sufficient to show that each Voronoi cell $V^i(\mathbf{P}_\mathbf{x})$ of the Euclidean Voronoi diagram $V(\mathbf{P}_\mathbf{x})$ in $\mathbb{R}^3$ intersects the embedding $\mathbf{x}(u, v)$ in a topological disk. Since $\mathbf{x}(u, v)$ is a $C^2$-smooth embedding into $\mathbb{R}^3$, and $\mathbf{x}(\mathbf{P})$ is a 0.18-sample of $\mathbf{X}$, we can apply Lemma 3.10 in [6], which exactly states the desired fact. □

## 6 Examples

Consider the two distance graphs shown in Figure 1. For both graphs we have constructed embeddings into $\mathbb{R}^3$. Voronoi diagrams have been generated for the resulting approximating SEG metrics, for a set of uniformly distributed sites.

Two more involved distance graphs are depicted in Figure 2. To obtain SEG metrics with accurrate approximations, we have generated spline embeddings into $\mathbb{R}^5$ and $\mathbb{R}^7$, respectively. The obtained SEG metric Voronoi diagrams are shown for a set of uniformly

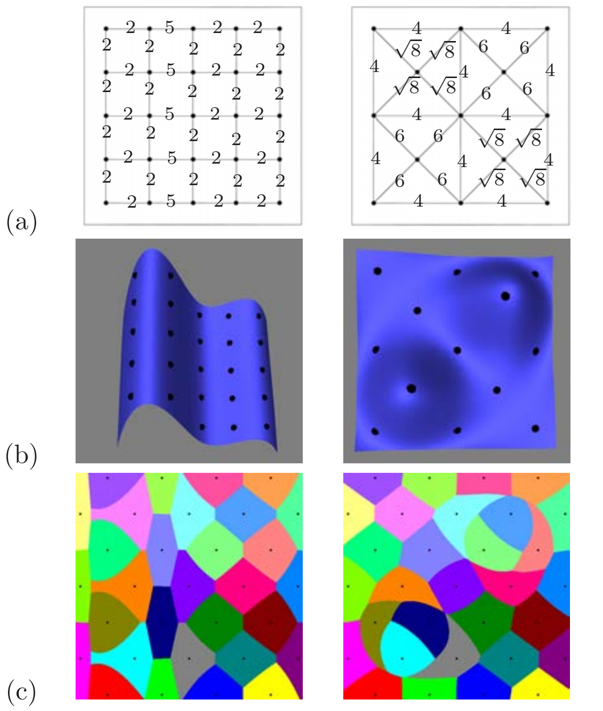


Figure 1: (a) Distance graphs; (b) Embeddings into $\mathbb{R}^3$; (c) Examples of SEG metric Voronoi diagrams.

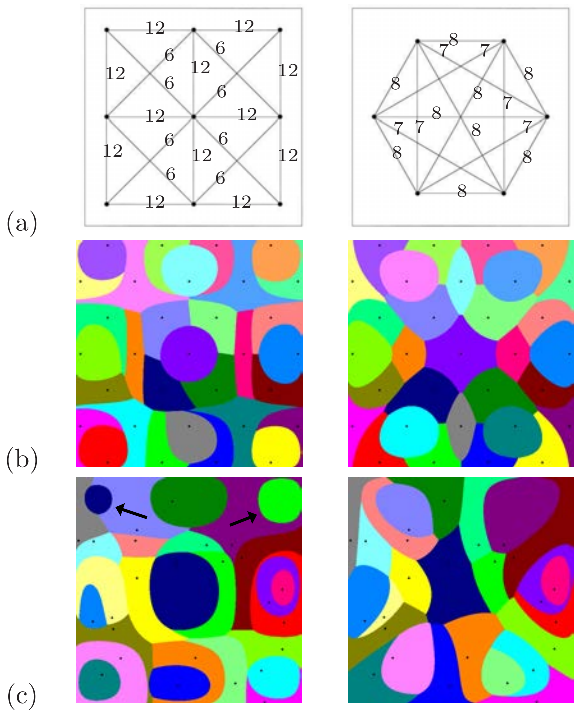


Figure 2: (a) Distance graphs; (b) and (c) Examples of SEG metric Voronoi diagrams.

distributed sites in (b), and for a set of non-uniformly distributed sites in (c). Occurring orphans are indicated by arrows.

## 7 Conclusion

We have introduced the concept of scaled embedding-generated (SEG) metrics, and have studied some of their properties. SEG metrics are a versatile tool for reflecting the anisotropy specified by distance graphs in the plane. Also, they lead to a new type of generalized Voronoi diagram in $\mathbb{R}^2$ in a canonical way.

As a possible application of our framework, we could generate a metric, which gives us the time of travel between cities. For the construction of the associated spline embedding, only a small number of selected times would be needed.

Various questions remain open, for example, conditions under which Voronoi cells are connected (or simply connected), if the embedding is in dimensions higher than 3; cf. the results in [4, 5]. In the non-orphan-free case, bounds on the number of connected Voronoi sub-cells are of interest.

Instead of the Voronoi diagram, also the medial axis for shapes with respect to the generalized disks defined by the new metric is worth studying.

## References

[1] N. Amenta, S. Choi, and R.K. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications* 19 (2001), 127–153.

[2] F. Aurenhammer and R. Klein. Voronoi diagrams. In: J. Sack and G. Urrutia (eds.), *Handbook of Computational Geometry*, Elsevier, Amsterdam, 2000, 201–290.

[3] J.-D. Boissonnat, C. Wormser, and M. Yvinec. Anisotropic diagrams: Labelle Shewchuk approach revisited. *Theoretical Computer Science* 408 (2008), 163–173.

[4] G.D. Canas and S.J. Gortler. Orphan-free anisotropic Voronoi diagrams. *Discrete & Computational Geometry* 46 (2011), 526–541.

[5] G.D. Canas and S.J. Gortler. Duals of orphan-free anisotropic Voronoi diagrams are triangulations. *Proc. 28th Ann. Symposium on Computational Geometry*, 2012, 219–228.

[6] T.K. Dey. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis.* Cambridge Monographs on Applied and Computational Mathematics 23, Cambridge University Press, 2007.

[7] Q. Du and D. Wang. Anisotropic centroidal Voronoi tessellations and their applications. *SIAM Journal on Scientific Computing* 26 (2005), 737–761.

[8] R. Kunze, F.-E. Wolter, and T. Rausch. Geodesic Voronoi diagrams on parametric surfaces. *Proc. Conference on Computer Graphics International*, 1997, 230–237.

[9] F. Labelle and J.R. Shewchuk. Anisotropic Voronoi diagrams and guaranteed-quality anisotropic mesh generation. *Proc. 19th Ann. Symposium on Computational Geometry*, 2003, 191–200.

# A Sweepline Algorithm for Higher Order Voronoi Diagrams[*]

Evanthia Papadopoulou[†] Maksym Zavershynskyi[†]

## Abstract

We present an algorithm to construct order-$k$ Voronoi diagrams with a sweepline technique. The sites can be points or line segments. The algorithm has $O(k^2 n \log n)$ time complexity and $O(nk)$ space complexity.

## 1 Introduction

Given a set of $n$ simple geometric objects in the plane, called sites, the order-$k$ Voronoi diagram of $S$, $V_k(S)$ is a partitioning of the plane into regions, such that every point within a fixed order-$k$ Voronoi region has the same set of $k$ nearest sites. For $k = 1$ this is the *nearest-neighbor Voronoi diagram* and for $k = n - 1$ the *farthest-site Voronoi diagram*.

The structural complexity of the order-$k$ Voronoi diagram in the $L_p$ metric is $O(k(n-k))$ for both points [11, 13] and line segments [15].

A standard simple technique to compute the order-$k$ Voronoi diagram is an iterative construction [11], where the order-$k$ Voronoi diagram is computed from the order-$(k-1)$ diagram, for increasing values of $k$, in $O(k^2 n \log n)$ time and $O(k^2(n-k))$ space. For points, more sophisticated techniques have been developed based on randomized approaches and duality between the order-$k$ Voronoi diagram and the $k$-level of planes in $R^3$, see e.g. [16, 1, 5, 4]. These algorithms exploit the fact that the sites are points, they construct the $k$-level in an arrangement of planes, and they are not simple to generalize to line segments, to the best of our knowledge. The best expected running time is $O(n \log n + nk2^{c \log^* k})$ [16], for a constant $c$, which is near-optimal, however, mostly of theoretical interest as mentioned in [16].

In this paper we investigate the plane sweep paradigm for the construction of higher order Voronoi diagrams of polygonal sites (including points and line segments) as a simple alternative to the iterative construction. Plane sweep has not been considered so far for the construction of higher order Voronoi diagrams. It is essentially an iterative construction, which, does not require the pre-computation and storage of lower order Voronoi diagrams, but achieves the construction in a single plane sweep pass of the input data. This ability can be useful in practice, especially when information involving all order-$i$, $i \leq k$, Voronoi diagrams is required. For example, in [14], the geometric min-cut problem in a VLSI layout is addressed by iteratively computing higher order Voronoi diagrams of (weighted) line segments that represent polygons. Information of low-order Voronoi diagrams is important in this application and the importance grows weaker fast as $k$ increases. However, an order-1 diagram is not sufficient. A direct plane sweep construction of the order-$i$ Voronoi diagram, $i \leq k$, for small $k$, is valuable to this application (see [14]).

Our algorithm is based on planesweep [8], where the *nearest neighbor* Voronoi diagram is constructed in $O(n \log n)$ time and $O(n)$ space. It constructs order-$i$ Voronoi diagrams, $i \leq k$, in $O(k^2 n \log n)$ time and $O(nk)$ space.

## 2 Preliminaries

In this abstract we describe the algorithm for disjoint line segments (including points). It can be extended to any planar straight-line graph (for the definition of the order$-k$ Voronoi diagram in this case see [15]).

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of $n$ disjoint line segments in $\mathbb{R}^2$, called sites. We make a *general position assumption* that no more than three sites can touch the same circle (excluding abutting line segments). The assumption can later be removed. The Euclidean distance between two points $p$, $q$ is denoted as $d(p, q)$. The distance between point $p$ and line segment $s$ is the minimum Euclidean distance $d(p, s) = \min_{q \in s} d(p, q)$. The order-$k$ Voronoi diagram $V_k(S)$ of $S$ is a partitioning of the plane into order-$k$ Voronoi regions such that every point within a fixed region $\mathcal{V}_k(H, S)$ has the same $k$ nearest sites $H$. A maximal interior-connected subset of a region is called a face.

Let $l$ be a horizontal line such that the halfplane $l^+$ above $l$ intersects at least $k$ line segments in $S' \subseteq S$. The locus of points equidistant from line segment $s \in S'$ and $l$ is denoted as $w(s)$ and it is called the *wave-curve* of $s$ (see Fig. 1). Consider an arrangement $A$ of wave-curves $w(s)$, $s \in S'$. The $i$-level of $A$ is the set of points $x$, such that $x$ belongs to some wave-curve $w(s)$ and has $i$ wave-curves below, including $w(s)$ [7]. The

[*]Supported in part by the Swiss National Science Foundation (SNF) grant 200021-127137. Also by SNF grant 20GG21-134355 within the collaborative research project EuroGIGA/VORONOI of the European Science Foundation.

[†]Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland,
{evanthia.papadopoulou, maksym.zavershynskyi}@usi.ch

intersection point of two wave-curves $w(s_1)$ and $w(s_2)$ on level $i$ has $i$ or $i+1$ wave-curves below, including $w(s_1)$ and $w(s_2)$; $x$ is called a *breakpoint* of level $i$. The $i$-level of $A$ is denoted as $A_i$. Level $A_i$ is a sequence of *waves*, each wave being a portion of a wave-curve between two consecutive intersection points on level $i$. A single wave-curve may contribute more than one wave to $A_i$. The ordinary wavefront in the standard plane sweep construction of Voronoi diagrams is the 1-level of $A$.

The following lemma states the basic idea of the sweepline technique for Voronoi diagrams [6].

**Lemma 1** *Let $l$ be a horizontal line and let $S' \subseteq S$ be set of sites that intersect $l^+$. Consider a point $x$ above $A_i$ and let $x \in \mathcal{V}_i(H, S')$ then $x \in \mathcal{V}_i(H, S)$. In other words the part of $V_i(S)$ above $A_i$ is not affected by the sites strictly below $l$.*

**Lemma 2** *Let $l$ be a horizontal line and let $S' \subseteq S$ be a set of sites that intersect $l^+$. Consider a point $x$, $x \in A_i$ and $x$ belongs to a single wave-curve $w(s_i)$. Let $w(s_1), \ldots, w(s_{i-1})$ be a set of wave-curves that are below $x$. Then $x \in \mathcal{V}_i(H, S)$, where $H = \{s_1, \ldots, s_i\}$.*

The bisector of two segments $s_j$ and $s_h$ is the locus of points equidistant from both, $b(s_j, s_h) = \{x \mid d(x, s_j) = d(x, s_h)\}$. The edge of the order-$i$ Voronoi diagram that bounds two adjacent faces $\mathcal{F}_j$ and $\mathcal{F}_h$ of regions $\mathcal{V}_i(H \cup \{s_j\}, S)$ and $\mathcal{V}_i(H \cup \{s_h\}, S)$ is portion of bisector $b(s_j, s_h)$. The following corollary implies that while the sweepline moves down, the intersections of two levels $A_i$ and $A_{i+1}$ move along the Voronoi edges of the order-$i$ Voronoi diagram.

**Corollary 3** *Let $x$ be an intersection point of two levels $A_i$ and $A_{i+1}$ incident to wave-curves $w(s_j)$ and $w(s_h)$. Let $H$ be a set of sites that correspond to the wave-curves below $x$. Then $x$ belongs to the Voronoi edge bounding two faces of regions $\mathcal{V}_i(H \cup \{s_j\}, S)$ and $\mathcal{V}_i(H \cup \{s_h\}, S)$.*

Voronoi vertices in $V_i(S)$ are classified into *new* and *old* [11, 15]. A Voronoi vertex of $V_i(S)$ is called *new* (respectively *old*) if it is the center of a disk that touches 3 sites and its interior intersects exactly $i-1$ (respectively $i-2$) sites. By the definition of the order-$i$ Voronoi diagram every Voronoi vertex of $V_i(S)$ is either *new* or *old*. A *new* Voronoi vertex in $V_i(S)$ is an *old* Voronoi vertex in $V_{i+1}(S)$. Under the general position assumption, an *old* Voronoi vertex in $V_i(S)$ is a *new* Voronoi vertex in $V_{i-1}(S)$.

During the sweeping process events occur when three wave-curves intersect at a common point. The following lemma ties up such an event with a Voronoi vertex of the order-$i$ Voronoi diagram of $S$.

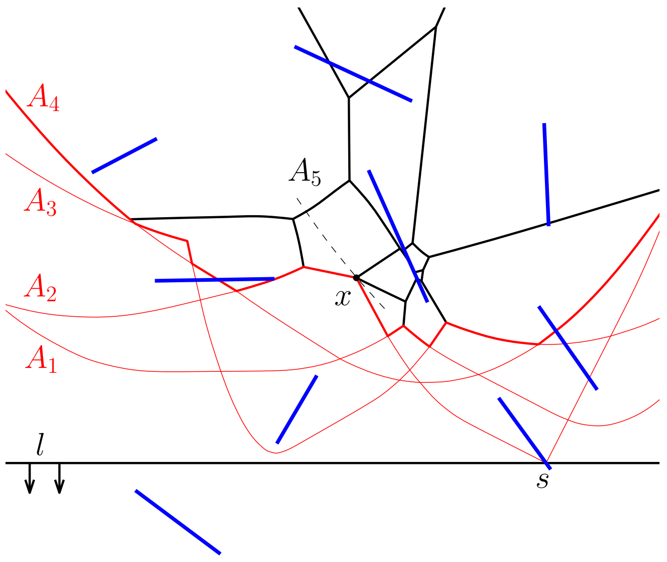


Figure 1: Constructing order-4 Voronoi diargam via sweepline technique.

**Lemma 4** *Let $l$, $H$ and $A$ be a horizontal line, a set of line segments that intersect $l^+$ and an arrangement of wave-curves created by the line segments in $H$. Let $x$ be an intersection point of three waves (see Fig. 1) such that $x$ belongs to three levels $A_{i+2}$, $A_{i+1}$ and $A_i$. Then $x$ is a new Voronoi vertex of $V_i(S)$ and an old Voronoi vertex of $V_{i+1}(S)$.*

Since wave-curves are $x$-monotone level $A_i$ is also an $x$-monotone curve. Corollary 3 implies that the breakpoints of $A_i$ lie on Voronoi edges of the final diagrams $V_i(S)$ and $V_{i-1}(S)$. As line $l$ moves down the waves forming $A_i$ change their shapes continuously. We maintain the sweepline status for level $A_i$ and we are interested in simulating the discrete event points that change the topological structure of $A_i$ and Voronoi diagrams $V_i(S)$ and $V_{i-1}(S)$. There are two types of events:

**Site-event:** Line $l$ hits a new site $s$. Then a new wave-curve $w(s)$ should be added to $A$. $w(s)$ is added to all levels of $A$, in level-$i$ similarly to the processing of a standard site event at level-1.

**Circle-event:** Three wave-curves intersect at a common point. A circle event corresponds to a new vertex at some level-$i$ and an old vertex at level-$i+1$ (Lemma 4).

The following lemma describes how the topology of $A$ changes during a circle-event.

**Lemma 5** *Consider a circle-event at point $x \in A_i$ such that $x$ is a new Voronoi vertex of $V_i(S)$. Then at this event a single wave disappears from $A_i$.*

The circle-event triggers topological changes on three levels $A_i, A_{i+1}, A_{i+2}$[1]. Thus to handle a circle-event we simulate the changes on all three levels. At the instant when the three wave-curves intersect at a common point the sweepline touches the bottom-most point of the disk that is tangent to three sites. For every triple of consecutive waves we create a circle-event in advance and we process it when the sweepline touches the bottom-most point of the disk. Let $w(s_1), w(s_2), w(s_3)$ be three consecutive waves that generate such event on level $A_i$. We delete the middle wave $w(s_2)$ from the level $A_i$. On level $A_{i+1}$ the waves appear as $w(s_2), w(s_1), w(s_3), w(s_2)$, so we switch the order of waves $w(s_1)$ and $w(s_3)$. And on level $A_{i+2}$ the waves appear as $w(s_3), w(s_1)$ so we add the wave $w(s_2)$ in between.

A new wave may be introduced to a level $A_i$ by a site-event or from the levels below $A_i$. Therefore in order to maintain $A_i$ during the sweepline it is sufficient to handle all site-events and all circle-events that appear on levels $1, \dots, i$.

## 3 The Algorithm

We sweep the plane with the horizontal line $l$ while maintaining levels $A_1, \dots, A_k$. Each level $A_i$ corresponds to a sequence of waves $w(s_1), w(s_2), \cdots$. For each level $A_i$ we store the ordered list $L_i$ of corresponding sites allowing search/insertion/deletion in logarithmic time. We also store a priority queue $Q$ of events ordered lexicographically by $y$-coordinate. A site-event occurs when line $l$ hits a new line segment. We order site-events by their $y$-coordinate of the top-most endpoint. We also create site-events for all the bottom-most endpoints.

We store circle-event as a disk that touches three line segments and we order them by their $y$-coordinate of the bottom-most point of the disk. We create circle-events every time the new consecutive triple of waves appears at some level after the site-events and circle-events. Every time the adjacency relations of levels $A_1, \dots, A_k$ change we need to create new events for new consecutive triples of waves and remove those events that do not correspond to consecutive triples anymore. For brievity we denote as *update_triplets*$(L_i, r_j, r_h)$ an operation that updates those triples that involve positions $r_j, \dots, r_h$ in list $L_i$. Operation *substitute*$(L_i, r, [a, b, c], [d, e, f])$ substitutes in list $L_i$ at position $r$ subsequence $a, b, c$ with subsequence $d, e, f$. We output order-$i$ Voronoi diagram as a set of Voronoi vertices and incident bisectors.

```
1: Initialize $Q = [s_1, \dots, s_n]$ sorted by $y$-coordinate
   of their topmost endpoint.
2: Extract the first site $s_1$.
3: Initialize $L_1 = [s_1]$ and $L_i = []$ for $i = 2, \dots, k$.
4: while $Q$ non empty do
5:   $x \leftarrow$ top_most$(Q)$
6:   if $x$ is a new site then
7:     Search for the wave in $L_1$ to which $x$ belongs.
8:     Let $s \in L_1$ be the corresponding line segment
       and $r$ be the position in list $L_1$.
9:     substitute$(L_1, r, [s], [s, x, s])$
10:    update_triplets$(L_1, r, r + 2)$
11:    $s' \leftarrow s$
12:    for $i = 2, \dots, k$ do
13:      if $L_i$ is empty then
14:        $L_i = [x, s', x]$
15:        exit for loop
16:      else
17:        Search for a position in $L_i$ to which be-
           longs $x$.
18:        Let $s \in L_i$ be the corresponding line seg-
           ment and $r$ be the position in list $L_i$.
19:        substitute$(L_i, r, [s], [s, x, s', x, s])$
20:        update_triplets$(L_i, r, r + 4)$
21:        $s' \leftarrow s$.
22:      end if
23:    end for
24:  else
25:    Let $x$ be a disk tangent to line segments
       $s_1, s_2, s_3$ and the event occurs at levels
       $A_i, A_{i+1}, A_{i+2}$ at positions $r, r', r''$, respec-
       tively (if $i + 1 > k$ or $i + 2 > k$ then the
       corresponding values are empty).
26:    if $i = k$ or $i + 1 = k$ then
27:    output $x$ and three bisectors incident to it
28:    substitute$(L_i, r, [s_1, s_2, s_3], [s_1, s_3])$
29:    update_triplets$(L_i, r, r + 2)$
30:    if $i + 1 \leq k$ then
31:      substitute$(L_{i+1}, r', [s_2, s_1, s_3, s_2], [s_2, s_3, s_1, s_2])$
32:      update_triplets$(L_{i+1}, r', r' + 3)$
33:    end if
34:    if $i + 2 \leq k$ then
35:      substitute$(L_{i+2}, r'', [s_3, s_1], [s_3, s_2, s_1])$
36:      update_triplets$(L_{i+2}, r'', r'' + 2)$
37:    end if
38:  end if
39: end while
```

The algorithm maintains the $i$-levels of arrangement $A$, $i \leq k$, while sweeping the plane with a horizontal line. Since the breakpoints of the $i$-level move along the edges of $V_i(S)$ and $V_{i-1}(S)$ the correctness of the algorithm follows.

The algorithm can extend to line segments forming a straight-line graph. Note that in this case if two line segments $s_q$ and $s_t$ share a common endpoint $p$ then the intersection of wave-curves $w(s_q)$ and $w(s_t)$ is not a point but a curve. We assign this common portion to endpoint $p$, $w(p)$, and we treat $w(s_q)$ and $w(s_t)$ as they are undefined at $w(p)$.

[1]This is under the general position assumption. If we remove the general position assumption then the changes may occur to more than 3 levels.

## 4 Complexity analysis

**Lemma 6** *The maximum size of queue $Q$ and the maximum total size of lists $L_1, \ldots, L_k$ are $O(nk)$.*

**Proof.** Lists $L_1, \ldots, L_k$ correspond to levels $A_1, \ldots, A_k$. Since the wave-curves are Jordan curves $g_{\leq k}(n) = O\left(k^2 g_1\left(\lfloor n/k \rfloor\right)\right)$ [17, 5] where $g_{\leq k}(n)$ is the maximum complexity of levels $A_1, \ldots, A_k$ and $g_1(m)$ is the maximum complexity of the lower envelope of $m$ wave-curves. The lower envelope of waves corresponds to wavefront of the order-1 Voronoi diagram [8], therefore $g_1(m) = O(m)$. Thus the maximum complexity of $A_1, \ldots, A_k$, $g_{\leq k}(n)$ is equal to $O(nk)$.

The number of site-events is $O(n)$. Every circle-event in event queue $Q$ corresponds to a triple of adjacent waves at some level $A_i$, $1 \leq i \leq k$. Therefore the number of circle-events is proportional to the total size of levels $A_1, \ldots, A_k$, which is $O(nk)$. Thus $Q$ is of size $O(nk)$. $\square$

**Theorem 7** *The algorithm can be implemented to run in time $O(k^2 n \log n)$ and $O(nk)$ space.*

**Proof.** The site-events correspond to the insertion of the new wave-curves in levels $A_1, \ldots, A_k$. The number of site-events is bounded by the number of sites, $O(n)$. When a new line segment intersects a halplane $l^+$ we insert it in lists $L_1, \ldots, L_k$. This requires a binary search on every list and therefore it takes $O(\log |L_i|)$ per list, where $|L_i|$ denotes the size of the list. Since the maximum complexity of $L_i$ is bounded by the structural complexity of the order-$i$ Voronoi diagram, we need $O\left(\log\left(i(n-i)\right)\right) = O(\log n)$ per level $L_i$, or $O(k \log n)$ for all levels. Therefore it takes $O(nk \log n)$ time to process all the site-events.

The circle-events correspond to the Voronoi vertices of the order-$i$ Voronoi diagrams, $i = 1, \ldots, k$. Every such event requires constant time. Since the number of order-$i$ Voronoi vertices is bounded by $O(i(n-i))$ thus it implies that the total number of circle-events is bounded by $\sum_{i=1}^{k} O(i(n-i)) = O(k^2 n)$. Every site-event and circle-event requires an update of the triples that involve the line segments that are adjacent to the places where the changes occured. Insertions and deletions into the event queue $Q$ require $\log |Q|$ time per each inserted/removed circle-event, where $|Q|$ - is the size of the queue. Lemma 6 implies that the size of the queue is $O(nk)$. Therefore it takes $O(\log(nk)) = O(\log n)$ time per event. And the total running time is $O\left(k^2 n \log n\right)$.

During the execution of the algorithm we store the event queue $Q$, lists $L_1, \ldots, L_k$ and we output the order-$k$ Voronoi diagram $V_k(S)$. Then Lemma 6 implies the total space complexity. $\square$

## References

[1] P. Agarwal, M. de Berg, J. Matousek, and O. Schwarzkopf. *Constructing levels in arrangements and higher order Voronoi diagrams.* SIAM J. Comput. 27(3): 654-667 (1998)

[2] F. Aurenhammer *Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure* ACM Comput. Surv. 23(3): 345-405 (1991)

[3] F. Aurenhammer, R. Drysdale, and H. Krasser. *Farthest line segment Voronoi diagrams.* Inf. Process. Lett. 100(6): 220-225 (2006)

[4] T. Chan. *Random Sampling, Halfspace Range Reporting, and Construction of $\leq k$-Levels in Three Dimensions* SIAM J. Comput. 30(2): 561-575 (2000)

[5] K. L. Clarkson *New applications of random sampling in computational geometry.* Discrete and Computational Geometry. 2: 195-222 (1987)

[6] F. K. H. A. Dehne and R. Klein *"The Big Sweep": On the Power of the Wavefront Approach to Voronoi Diagrams.* Algorithmica 17(1): 19-32 (1997)

[7] H. Edelsbrunner and R. Seidel *Voronoi Diagrams and Arrangements.* Discrete & Computational Geometry 1: 25-44 (1986)

[8] S. Fortune. *A Sweepline Algorithm for Voronoi Diagrams.* Algorithmica 2: 153-174 (1987)

[9] M. I. Karavelas. *A robust and efficient implementation for the segment Voronoi diagram.* In Proc. 1st Int. Symp. on Voronoi Diagrams in Science and Engineering, Tokyo: 51-62 (2004)

[10] D. G. Kirkpatrick. *Efficient Computation of Continuous Skeletons* FOCS 1979: 18-27

[11] D. T. Lee. *On k-Nearest Neighbor Voronoi Diagrams in the Plane.* IEEE Trans. Computers 31(6): 478-487 (1982)

[12] D. T. Lee and R. L. S. Drysdale. *Generalization of Voronoi Diagrams in the Plane.* SIAM J. Comput. 10(1): 73-87 (1981)

[13] C.H. Liu, E. Papadopoulou, and D.T. Lee *An output-sensitive approach for the $L_1$;/$L_\infty$ k-Nearest-Neighbor Voronoi diagram.* ESA, LNCS 6942, 70-81 (2011).

[14] E. Papadopoulou *Net-aware critical area extraction for opens in circuits via higher-order Voronoi diagrams.* IEEE Trans. on Comp.-Aided Design, vol. 20, no.5, 583-597, May 2011.

[15] E. Papadopoulou and M. Zavershynskyi *On higher order Voronoi diagrams of line segments.* ISAAC, Taipei: 177-186 (2012)

[16] E.A. Ramos On Range Reporting, Ray Shooting and $k$-Level Construction. *Symposium on Computational Geometry. 390-399 (1999)*

[17] M. Sharir and P. Agarwal. *Davenport-Schinzel Sequences and their Geometric Applications.* Cambridge University Press, 1995.

[18] C.-K. Yap. *An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments.* Discrete & Computational Geometry 2: 365-393 (1987)

# On the complexity of the partial least-squares matching Voronoi diagram

Matthias Henze[*] Rafel Jaume[†] Balázs Keszegh[‡]

## Abstract

Given two point sets of sizes $n$ and $m$, we study the partial matching problem of translating the smaller point set to a position where it is best resembled by an equally sized subset of the larger point set. Measuring the similarity is done by the sum of squares of the Euclidean distances between the matched points in either set. A Voronoi-type diagram can be associated in a natural way. We prove that the complexity of this diagram is $O(m!\,m^d n^{2d})$, where $n$ is the size of the bigger set. A combinatorial problem on stable matchings that arises from our argument is of independent interest.

## 1 Introduction

Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be finite point sets in $\mathbb{R}^d$ with $m \le n$. In the partial point matching problem, we are interested in finding an optimal injective match of $B$ within $A$ with respect to a given measure of similarity and a set of available transformations. In this paper, we study the sum of squared Euclidean distances as a measure and will allow the point set $B$ to be translated to any desired position, that is, we want to solve

$$\begin{aligned} &\text{minimize} && c_\pi(t) = \sum_{i=1}^{m} \|b_i + t - a_{\pi(i)}\|^2 && (1)\\ &\text{subject to} && \pi : [m] \hookrightarrow [n] \text{ an injection,} \\ & && t \in \mathbb{R}^d. \end{aligned}$$

The injectivity requirement on the matching is crucial. Dropping it reduces the problem to traversing regions of an overlay of the $m$ Voronoi diagrams of the point sets $A - b_i$, $i \in [m]$, which can be done efficiently.

As an approach to decide whether problem (1) can be solved in polynomial time in $n$ and $m$, Rote [8] associated a Voronoi-type diagram with this matching problem, that may be called the *partial (least-squares) matching Voronoi diagram* $\mathcal{V}(A, B)$ of $A$ and $B$. In this diagram each injection $\pi : [m] \hookrightarrow [n]$ defines a (possibly empty) region $P_\pi$ that consists of all translations $t$ for which $\pi$ is the optimal assignment for $B + t$. More precisely,

$$P_\pi = \left\{ t \in \mathbb{R}^d : c_\pi(t) \le c_\sigma(t) \text{ for all } \sigma : [m] \hookrightarrow [n] \right\}.$$

Rote [8] observed that every $m$-element subset of $A$ is the image of at most one injection that contributes a region to the diagram. This is due to the translation invariance of the optimal matching between two equally sized sets. Further, by

$$c_\pi(t) = \sum_{i=1}^{m} \|b_i - a_{\pi(i)}\|^2 + 2\left\langle t, \sum_{i=1}^{m} (b_i - a_{\pi(i)}) \right\rangle + m\|t\|^2,$$

the regions $P_\pi$ are intersections of finitely many affine half-spaces and hence the partial matching Voronoi diagram is a polyhedral subdivision of $\mathbb{R}^d$. Allowing multiplicities in the point set $B$, we get as a special case the $m$th order Voronoi diagram $\mathcal{V}_m(A)$ of $A$ by setting $b_1 = \ldots = b_m = 0$.

A polynomial bound on the number of *cells*, i.e., full-dimensional regions, of $\mathcal{V}(A, B)$ would yield an efficient algorithm to solve the matching problem (1). Such an algorithm would traverse all cells and solve the minimization problem inside a fixed cell by means of quadratic programming. For more details on this approach we refer to [8]. In the special case $\mathcal{V}_m(A)$ the number of cells is known to be polynomial in $n$ and $m$. A rough bound of $O(n^{2d})$ follows from the complexity of hyperplane arrangements (see [3]), and the best bound that is sensitive to $m$ is due to Clarkson & Shor [2] and reads $O(n^{\lfloor \frac{d}{2} \rfloor} m^{\lceil \frac{d}{2} \rceil})$ for $n/m \to \infty$.

The following result implies that, for $d = 1$, the number of cells in $\mathcal{V}(A, B)$ is at most quadratic.

**Theorem 1 (Rote [8])** *Every line intersects the interior of at most $m(n - m) + 1$ cells of $\mathcal{V}(A, B)$.*

This bound is best possible which can be seen by taking $A$ and $B$ to be equally spaced points on a line, and where the spacing between the points in $A$ is very small and between the points in $B$ very large.

We may generalize this example to arbitrary dimension by putting such an instance on $\frac{n}{d}$ and $\frac{m}{d}$ points in each of the $d$ coordinate axes. Doing this carefully leads to a diagram with $\Omega\left(\frac{m^d (n-m)^d}{d^{2d}}\right)$ cells and motivates the subsequent conjecture.

[*]Institute of Computer Science, Freie Universität Berlin, Germany, `matthias.henze@fu-berlin.de`; Research supported by ESF EUROCORES programme EuroGIGA-VORONOI, (DFG): RO 2338/5-1.

[†]Institute of Computer Science, Freie Universität Berlin, Germany, `jaume@mi.fu-berlin.de`; Research supported by La-Caixa and the DAAD.

[‡]Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences Budapest, Hungary, `keszegh@renyi.hu`; Research supported by OTKA, grant NK 78439 and by OTKA under EUROGIGA project GraDR 10-EuroGIGA-OP-003.

**Conjecture 1** *The partial matching Voronoi diagram of point sets $A, B \subset \mathbb{R}^d$ of sizes $n = |A| \geq |B| = m$ consists of $O(m^d(n-m)^d)$ cells.*

## 2 Counting stable matchings

In this section, we present a combinatorial result that will be used later on in order to derive first bounds on the complexity of $\mathcal{V}(A, B)$. The combinatorial problem we are considering is a stable marriage type question.

The stable marriage problem was first introduced by Gale & Shapley [4] in 1962. It is usually stated as trying to marry $n$ men with $n$ women, each of them with a ranking of the people of the opposite gender, in a way that no non-married pair would bilaterally want to have an affair. In the classical paper [4], it is shown that such a stable marriage exists for any given set of rankings. In fact, in general there are a lot of stable marriages for a single instance. Knuth [7] and Irving & Leather [5] gave some bounds on this number.

A lot of variants of the problem have been studied (see [6] for a survey), e.g., stable marriages with incomplete rankings or with ties, the stable roommates problem or the hospital/residents problem. The variant we are interested in was first introduced by Shapley & Scarf [9] in 1974, and is called the "House Allocation Problem." The literature on this problem focuses on algorithmic questions, while we are interested in a bound on the number of stable marriages.

Throughout this section, we let $\mathcal{D} = (d_1, \ldots, d_m) \in S_n^m$ be a list of permutations on $n$ elements. These may be regarded as $m$ linear orderings on $[n]$ in the sense that $a \leq_i b$ if and only if $d_i^{-1}(a) \leq d_i^{-1}(b)$. The elements of $[m]$ are usually denoted by $i, j, \ldots$ and the elements of $[n]$ by $a, b, \ldots$.

**Definition 1** *We say that an injection $\mu : [m] \hookrightarrow [n]$ is a* matching *and its image $M(\mu) = \mu([m])$ is called the* matched set *of $\mu$.*

- *A matching $\mu$ is said to be* better *than $\nu$ (with respect to $\mathcal{D}$), if*
$$\mu(i) \leq_i \nu(i) \quad \text{for all} \quad i \in [m].$$
- *A matching $\mu$ is called* stable *(with respect to $\mathcal{D}$), if there is no better matching $\nu \neq \mu$.*

The following observations about this kind of matchings can be easily proved.

**Observation 1**

*i) Stable matchings are non-parallel, i.e., given two stable matchings $\mu$ and $\nu$, there are $i, j \in [m]$ such that $\mu(i) <_i \nu(i)$ and $\nu(j) <_j \mu(j)$.*

*ii) If $\mu$ is stable, then*
$$\{a \in [n] : \exists i \in [m] \text{ s.t. } a \leq_i \mu(i)\} \subseteq M(\mu).$$
*In particular, only the $m$ smallest values in each of the orderings $\leq_1, \ldots, \leq_m$ can appear in a stable matching. Hence, at most $m^2$ different elements $a \in [n]$ are used.*

*iii) If $\mu$ is stable, there is no sequence $i_1, \ldots, i_{m'}$ of length $m' \leq m$ satisfying*
$$\mu(i_{k-1}) <_{i_k} \mu(i_k) \quad \text{for all} \quad k \in [m'],$$
*where $i_0 := i_{m'}$.*

**Definition 2** *Let $\rho \in S_m$ be a permutation on $[m]$. The* $\rho$-greedy matching *$\mu$ for $\mathcal{D}$ is the unique injection that satisfies $\mu(i) \leq_{\rho(i)} b$, for all $i \in [m]$ and for all $b \in [n] \setminus \{\mu(1), \ldots, \mu(i-1)\}$. A matching is called* greedy *if it is $\rho$-greedy for some $\rho \in S_m$.*

The following characterization of stable matchings already appears in [1, Lem. 1] for the case $m = n$. Therein, stable matchings are called Pareto-efficient and greedy matchings serial-dictatorship mechanisms. We give a more focused and shorter proof of this result that moreover covers the general case $m \leq n$.

**Theorem 2** *A matching is stable if and only if it is greedy.*

**Proof.** First of all, let $\mu$ be a $\rho$-greedy matching and let us assume on the contrary that there is a better matching $\nu$. Let $i \in [m]$ be the smallest index such that $\nu(i) <_{\rho(i)} \mu(i)$. This contradicts the definition of the greedy matching $\mu$ since it would have chosen the element $\nu(i)$. Hence, $\mu$ is stable.

Conversely, if $\mu$ is a stable matching, then there must be some index $i \in [m]$ such that $\mu(i) = d_i(1)$, i.e., $\mu(i)$ is the smallest in the order $\leq_i$. Assuming the contrary means, by Observation 1 ii), that for all $i \in [m]$ there exists an $i' \in [m] \setminus \{i\}$ such that $\mu(i') = d_i(1)$. In particular, there is a cycle $i_1, \ldots, i_{m'}$ such that $\mu(i_{k-1}) = d_{i_k}(1)$, for all $k \in [m']$, where $i_0 := i_{m'}$. The matching $\nu$ obtained from $\mu$ by exchanging $\mu(i_k)$ for $d_{i_k}(1)$, for all $k \in [m']$, is then better than $\mu$, contradicting stability.

We now construct the permutation $\rho$ as follows. Let $\rho(1) = i_1$, where $\mu(i_1) = d_{i_1}(1)$. Then, we consider the induced matching $\mu_1 : [m] \setminus \{i_1\} \hookrightarrow [n] \setminus \{\mu(i_1)\}$. It is easy to see that this is a stable matching, as a better matching would yield a better matching for $\mu$ as well. Again we find an element $i_2 \in [m] \setminus \{i_1\}$ such that $\mu_1(i_2) = d_{i_2}(1)$ and set $\rho(2) = i_2$. Repeating this process yields the permutation $\rho \in S_m$ and $\mu$ is $\rho$-greedy. $\square$

**Corollary 3** *The number of stable matchings with respect to a given $\mathcal{D}$ is at most $m!$ and, in general, this bound cannot be improved.*

**Proof.** By Theorem 2, stable matchings are greedy matchings. There exists only one greedy matching for each of the $m!$ permutations of $[m]$.

For the lower bound construction observe that if all the permutations $d_i$ are equal, then every $\rho \in S_m$ induces a different greedy matching. $\square$

Motivated by the translation invariance discussed in the introduction, we are actually interested in the number of matched sets of stable matchings. To this end, we define

$$m^g = \{M(\mu) : \mu \text{ is a stable/greedy matching}\}$$

and

$$M^g = \bigcup_{\mu \text{ is a stable/greedy matching}} M(\mu).$$

The latter set contains those elements of $[n]$ that appear in some stable/greedy matching. Clearly, $|m^g| \le \binom{|M^g|}{m}$, which motivates us to determine the order of magnitude of $|M^g|$ in the worst case. We have already seen in Observation 1 ii), that $|M^g| \le m^2$. We are not aware of an upper bound of the type $o(m^2)$. Our best lower bound construction is quite apart from this upper bound and comes from the following recursive example:

**Claim 1** *There exists a $\mathcal{D}$ such that*

$$|M^g| \ge \frac{m}{2} \log 4m \in \Omega(m \log m).$$

**Proof.** We give a recursive construction. In each step we double the size of $m$. For $m = 1$, evidently $|M^g| = 1$. We say that a construction reaches some element $a \in [n]$ if $a \in M^g$, i.e., $a \in M(\mu)$ for some stable/greedy matching $\mu$. Assuming that we already constructed a $\mathcal{D}$ for $m$ that reaches $\frac{m}{2} \log 4m$ elements, we construct a $\mathcal{D}'$ for $2m$ that reaches $m \log 8m$ elements. In order to do that, we let $m$ new elements be the smallest in the preference orders of $d_1, \dots, d_m$ and let the same elements in the same order be the smallest in the preference orders for $d_{m+1}, \dots, d_{2m}$. Thus $d_i(1) = d_{i+m}(1)$ for all $i \in [m]$. The remaining preferences of $d_1, \dots, d_m$ are chosen according to the construction for $\mathcal{D}$. The rest of the preferences of $d_{m+1}, \dots, d_{2m}$ are set equivalently, just using a new set of elements of $[n]$ (we consider $n$ not to be fixed). In this way, any $\rho$ that starts with the first $m$ elements of $[2m]$ in some order and then traverses the second $m$ elements behaves exactly like a $\rho'$ preference order on the previous construction $\mathcal{D}$. The same thing applies to any $\rho$ that starts on the second $m$ elements. Thus, the number of elements of $[n]$ that belong to some set $M(\rho)$ is at least $m + |M^g| + |M^g| = m + 2\frac{m}{2} \log 4m = m \log 8m$, as desired. $\square$

A curious observation is that it is NP-hard to decide whether an element of $[n]$ belongs to $M^g$.

We now investigate $|m^g|$ directly, because from the considerations above we only get $|m^g| \le \binom{|M^g|}{m} \le \binom{m^2}{m} \in O((me)^m)$. The trivial bound $|m^g| \le m!$ is much better and basically this is the best upper bound we know up to the order of magnitude.

Our best lower bound comes from the following simple example:

**Claim 2** *There exists a $\mathcal{D}$ such that*

$$|m^g| \ge \binom{m}{\lfloor \frac{m}{2} \rfloor} \in \Omega\left(\frac{2^m}{\sqrt{m}}\right).$$

**Proof.** Choose $\mathcal{D}$ such that for every $\le_i$ the $\lfloor \frac{m}{2} \rfloor$ smallest elements are the same, i.e., $d_i(k) = d_j(k)$ for all $i, j \in [m], k \in [\lfloor \frac{m}{2} \rfloor]$, and the next elements are all different, i.e., $d_i(\lfloor \frac{m}{2} \rfloor + 1) \ne d_j(\lfloor \frac{m}{2} \rfloor + 1)$ for $i \ne j$. Now, when we greedily choose the smallest elements according to the order of some $\rho \in S_m$, in the first $\lfloor \frac{m}{2} \rfloor$ steps we choose one of the smallest $\lfloor \frac{m}{2} \rfloor$ elements and after exactly $\lfloor \frac{m}{2} \rfloor$ steps all of these have been chosen. Then, in the last $\lceil \frac{m}{2} \rceil$ steps we choose by construction the $\lfloor \frac{m}{2} \rfloor + 1$st smallest element with respect to the actual $\le_i$. Thus, any subset of size $\lceil \frac{m}{2} \rceil$ appears in the second half of some ordering $\rho$, and hence we have $\binom{m}{\lfloor \frac{m}{2} \rfloor}$ different sets in $m^g$. $\square$

**Problem 1** *Determine if the two previous claims exhibit worst case examples, i.e., is it true that $|m^g| \in O(2^m)$ and $|M^g| \in O(m \log m)$?*

## 3 A polynomial bound in the size of the bigger set

We are now prepared to discuss the geometric part of the argument that establishes an upper bound for the number of cells of $\mathcal{V}(A, B)$ which is polynomial in the size of the bigger set $A$. This result is formalized in the following theorem.

**Theorem 4** *The partial matching Voronoi diagram $\mathcal{V}(A, B)$ of point sets $A, B \subset \mathbb{R}^d$ of sizes $n = |A| \ge |B| = m$ consists of $O(m!\, m^d n^{2d})$ cells.*

The bisector between two points $a_i, a_j \in A$ is the hyperplane defined by

$$B(a_i, a_j) = \{x \in \mathbb{R}^d : \|x - a_i\| = \|x - a_j\|\}.$$

Consider the hyperplane arrangement $\mathcal{A}$ in $\mathbb{R}^d$ consisting of the $m\binom{n}{2}$ hyperplanes $B(a_i, a_j) - b_k$, where $i, j \in [n]$ and $k \in [m]$.

**Lemma 5** *Every cell $C \in \mathcal{A}$ intersects the interior of at most $m!$ different cells of $\mathcal{V}(A, B)$.*

**Proof.** Let $C \in \mathcal{A}$ be a cell and let $b_k \in B$. Every $t \in \operatorname{int} C$ defines a linear ordering $\leq_k$ on $[n]$: For $i, j \in [n]$, we write $i \leq_k j$ if and only if $\|b_k + t - a_i\| \leq \|b_k + t - a_j\|$. This ordering induces a permutation $d_k$ on $[n]$ that, by definition of $\mathcal{A}$, depends on $k$ and $C$, but not on the actual $t \in \operatorname{int} C$. In addition, it is easy to see that the optimal matching for any $t$ must be stable in the sense of Definition 1. Otherwise, a better matching would attain a smaller value for the cost function, contradicting the optimality of the first. Hence, by Corollary 3, at most $m!$ matchings can be optimal for some $t \in \operatorname{int} C$ and the lemma follows. ☐

**Proof.** [Theorem 4] With the help of Lemma 5, Theorem 4 follows by observing that the number of cells in $\mathcal{A}$ is $O(m^d n^{2d})$. This is true because the number of cells in a $d$-dimensional arrangement of $h$ hyperplanes is $O(h^d)$ (see [3]) and, in our case, we are dealing with $m\binom{n}{2} = O(mn^2)$ bisecting hyperplanes. ☐

The arguments leading to Theorem 4 are not specific to the choice of the sum of squared Euclidean distances as the cost function nor to the choice of translations as transformations in the matching problem (1). They apply to arbitrary distance functions and to every cost function that is increasing with the distances. Considering other transformations amounts to subdividing other parameter spaces. The resulting bounds on the number of optimal assignments will then depend on the complexity of the arrangement of bisectors in that parameter space. Note, that the bisectors are defined by the chosen distance function.

A particularly interesting example is the case in which, instead of translating $B$, we look to find optimal assignments when we are allowed to transform $B$ subject to any linear mapping. In this case, the parameter space we want to subdivide is $\mathbb{R}^{d \times d}$, the space of all $(d \times d)$-matrices. The preference orders are now defined by the relations $\|M \cdot b_k - a_i\| \leq \|M \cdot b_k - a_j\|$, where $M \in \mathbb{R}^{d \times d}$, which again correspond to linear inequalities in the parameter space. The associated hyperplane arrangement in $\mathbb{R}^{d \times d}$ consists of $O(m^{d^2} n^{2d^2})$ cells, and hence the number of optimal assignments is $O(m!\, m^{d^2} n^{2d^2})$.

The particular case of rotating $B$ around a fixed point was posed as a problem in [8]. Still, our bound is only useful when $m$ is much smaller than $n$ and the polynomiality remains open even for equally sized sets $A$ and $B$.

## 4 Conclusions and further work

We considered the problem of bounding the number of optimal assignments of a point set $B$ to a subset of a bigger set $A$ with respect to the sum of squared distances and translations. Our main result is a bound that is polynomial in the size of the bigger set, independently of the size of the smaller one. We have also seen that our arguments extend to assignments with respect to more general cost functions and other sets of available transformations.

In view of Conjecture 1, the natural line of further investigations is reducing the factor $m!$ that appears in the bound of Theorem 4. As illustrated by Claim 2, our combinatorial approach cannot yield an upper bound of the type $o\left(\frac{2^m}{\sqrt{m}}\right)$. Further improvements have to rely more on the geometric properties of the problem. For instance, in the least-squares matching Voronoi diagram, each subset $A' \subseteq A$ of size $|A'| = m$ gives rise to at most one cell, and also not every stable matching that is counted in Section 2 necessarily comes from an optimal assignment. Both are properties that we have not been able to exploit so far.

**Acknowledgement.** This work was carried out during a visit of the third author in Berlin, funded by a EUROCORES short term visit grant in the framework of the EuroGIGA programme.

## References

[1] A. Abdulkadiroğlu and T. Sönmez, *Random serial dictatorship and the core from random endowments in house allocation problems*, Econometrica **66** (1998), no. 3, 689–701.

[2] K. L. Clarkson and P. W. Shor, *Applications of Random Sampling in Computational Geometry, II*, Discrete Comput. Geom. **4** (1989), 387–421.

[3] H. Edelsbrunner, *Algorithms in combinatorial geometry*, EATCS Monographs on Theoretical Computer Science, vol. 10, Springer-Verlag, Berlin, Heidelberg, 1987.

[4] D. Gale and L. S. Shapley, *College admissions and the stability of marriage*, Amer. Math. Monthly **69** (1962), no. 1, 9–15.

[5] R. W. Irving and P. Leather, *The complexity of counting stable marriages*, SIAM J. Comput. **15** (1986), no. 3, 655–667.

[6] K. Iwama and S. Miyazaki, *A Survey of the Stable Marriage Problem and Its Variants*, Proc. International Conference on Informatics Education and Research for Knowledge-Circulating Society (Washington, DC, USA), IEEE Computer Society, 2008, pp. 131–136.

[7] D. E. Knuth, *Mariages stables et leurs relations avec d'autres problèmes combinatoires*, Les Presses de l'Université de Montréal, Montreal, Que., 1976, Introduction à l'analyse mathématique des algorithmes, Collection de la Chaire Aisenstadt.

[8] G. Rote, *Partial Least-Squares Point Matching under Translations*, Proc. 26th European Workshop on Computational Geometry, 2010, pp. 249–251.

[9] L. S. Shapley and H. Scarf, *On cores and indivisibility*, J. Math. Econom. **1** (1974), no. 1, 23–37.

# A Truly Local Strategy for Ant Robots Cleaning Expanding Domains

Rolf Klein, David Kriesel and Elmar Langetepe*

## Abstract

The problem of *cleaning expanding domains* requires ant-like robots to clean a growing contamination in a grid world. It can be seen as a dynamic variant of online coverage. To the best of our knowledge, we present the first approach that does not require any central guidance or global data structure. We do so by rendering the geometric complexity of problem instances measurable and by maintaining complexity guarantees over time.

## 1 Introduction

Four main principles characterize our everyday life. (i) Our world is highly dynamic. (ii) We catch on to very little of what happens in it, for we possess only limited and local means of perception. (iii) Still, we can store and process a vanishingly small part from the little we perceive. (iv) If nothing else, at least there are many of us. Boiled down, life itself can be seen as a system of limited individuals organized in a decentral manner.

Given this point of view, one might notice that most theory of motion planning considers only subsets of these principles. Various problems have been analyzed in the fields of offline and online motion planning, postulating computationally powerful agents, often solitary ones, in mostly static domains, for example [4]. During the last years, the awareness of (iv) grew: Artificial multi-agent systems became increasingly inspired by advantages of biological, decentrally organized systems – *swarms*. They exhibit fault-resistance and cost-efficiency while being able to solve complex tasks. At the same time, their individual parts tend to lack computational power, storage capacity and global domain knowledge. As a result, awareness of (iii) and (ii) increased, too. Numerous static *ant robotics* scenarios for single and multiple agents have been theoretically analyzed [1, 5–7].

However, there is far less mathematical and geometrical theory of ant-like systems in *dynamic* domains, though consciousness for (i) is high in practice. A reason might be that, despite heuristics for solving such problems may be easy to find, exact mathematical analysis often turns out to be surprisingly difficult. In this paper, we present an approach to the dynamic problem of *cleaning expanding domains* [2, 3, 7]. To the best of our knowledge, it is the first to operate completely without global, knowledge.

The rest of the article is organized as follows. In Section 2, we first give a problem definition. Secondly, we render its complexity accessible to the reader's intuition by commenting on related work from our perspective. Compared to related work, we outline our contributions in Section 2.2 as well as some restrictions we place on problem instances due to the fact that our approach is work in progress and writing space is tight. In Section 3, we briefly describe an agent model and cleaning strategy for expanding domains. The proposed strategy's correctness and run time analysis is then sketched in Section 4. Eventually, we draw some conclusions and sketch further research in Section 5.

## 2 Cleaning Expanding Domains

Let time be discrete and the world uniformly tesselated into square *cells*. At any time $t$, a cell $c$ has a binary contamination property, taking either the value *contaminated* or *clean*. Edges that separate a contaminated and a clean cell are called *border edges*. We refer to the set of all contaminated cells as the *contamination*. We also make use of the common compass orientations and neighbourhood definitions in grid worlds, named 8Neighbours ($\mathit{8nb}(c)$) and 4Neighbours ($\mathit{4nb}(c)$) in this paper.

Motivated by phenomena like oil spills, a contamination *spreads* as time passes by. This happens every $d$ time steps. During a spread, any clean cell in the 4Neighbours of a contaminated one becomes contaminated as well. The contamination is assumed to be connected in the beginning. We do not consider two diagonally adjacent contaminated cells as connected. Connectedness, in contrast to simply-connectedness, means that we permit the contamination to contain *holes*, i.e., clean enclaves within the contamination.

Before discussing the state of the art, let us shortly define the concepts of boundary cells and critical cells (depicted yellow and red in some of the following figures). Let $c$ be a contaminated cell. We say that $c$ is a *boundary cell* if another cell $x \in \mathit{8nb}(c)$ is clean and not part of a hole. In this paper, we refer to the set of boundary cells as the *boundary*. In order to see whether or not $c$ is *critical*, let us consider only $\mathit{8nb}(c)$ isolated from the rest of the contamination. If $\mathit{8nb}(c)$ contains more than one connected, contaminated component, we call $c$ critical. In other words, an agent only able to sense $\mathit{8nb}(c)$ cannot guarantee to maintain the contamination's connectivity if clean-

*All: Department of Computer Science, University of Bonn, Germany. EMails: rolf.klein@uni-bonn.de, mail@dkriesel.com, elmar.langetepe@cs.uni-bonn.de.

ing $c$. The wish for local classification of critical cells comes at the expense of restrictivity: Some cells may be classified critical even though their cleaning might actually have no impact on the contamination's connectivity.

## 2.1 History and State of the Art

The problem of grid domain cleaning using ant agents was first adressed in [7] for the special case of static contaminations without holes. The authors solve the problem by – colloquially spoken – *edge peeling*. They require a group of ant agents to start at a boundary cell. Then, the agents traverse the boundary, peeling off layers by cleaning any non-critical cell they encounter. The authors present an analysis of correctness and cleaning time for a number of $k$ agents.

In a further article, the same authors proceed discussing dynamic contaminations as defined above [3]. They present a general lower bound for cleaning strategies solving it and propose to reuse their edge peeling strategy. In order to understand their solution, let us first explain that extending the discussion from the static problem to its dynamic version turns out to be not trivial at all. We name just a few exemplary challenges faced during our own research. First, boundary parts of a contamination may be merged by spreads. This may not only leave agents guideless, also such merges may create new holes. From there, one is forced to take holes into account, regardless of an initial contamination's simply-connectedness. Consequently, when performing edge peeling-like strategies, agents have to distinguish between contamination and hole boundaries by only local means – a non-trivial task given that there are no restrictions on the contamination's size. Also, the paradigm of ant-robots does not allow advanced techniques like map building. As holes may cause arbitrarily located critical cells, agent cleaning operations complicate things further. Critical cells may lead edge peeling agents to create arbitrarily complex contamination boundaries. Eventually, agents and spreads can even complement one another in creating new, arbitrarily complicated holes (Fig. 1).

The authors evaded this hardly controllable interplay by requiring not only the contamination's initial simply-connectedness, but additionally an *elastic membrane*. The membrane is a continuously-updated global data structure acting like a rubber band around the contamination, artificially maintaining simply-connectedness over time (Fig. 2). Cells separated by the membrane are not considered connected.

It replaces the contamination boundary as guidance – the agents now traverse a globally maintained data structure. Given the membrane, the authors can prove that a group of agents is able to clean a membrane-enclosed problem instance if it spreads slow enough. Unfortunately, in combination with spreads and the agent's cleaning, the membrane can become arbitrarily meandering, even causing critical cells in actually solid contaminations. No geometric properties of the contamination's shape can be guaranteed at all. This is too bad, because one could make a virtue out of necessity here: Without membrane, boundaries of a dynamic contamination actually tend to become *less* meandering by spreads. Consequently, it is hard to find direct upper bounds on edge-peeling-like cleaning strategies. The authors manage to do so in a third article [2].

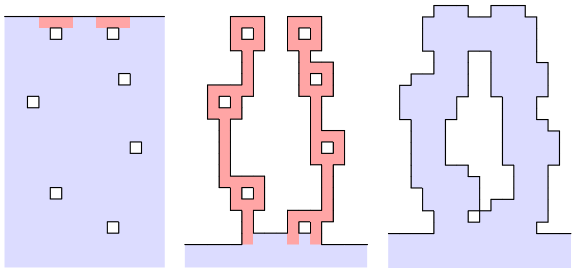


Figure 1: In the left image part, a section of a large, solid rectangular contamination with some minimal holes is illustrated. In the middle, the same section is illustrated after several edge peeling agent traversals. On the right side, we can see the spread outcome of the configuration in the middle. Critical cells are depicted red.

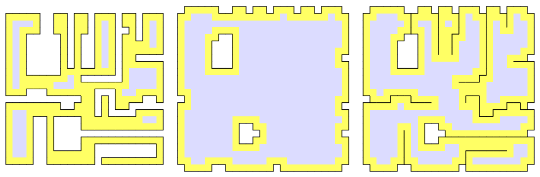


Figure 2: A contamination $C$ (left) and the spread outcomes of $C$ both without membrane (middle) and with membrane (right). Even though holes emerge, $C$ stays simply-connected, if the elastic membrane restriction is placed. Cells considered to be part of the boundary in each of the cases are painted yellow.

## 2.2 Contribution and Restrictions

To the best of our knowledge, we present the first approach to cleaning of expanding domains that at one go (i) does not require any global knowledge, and (ii) allows holes. Further, (iii) agent starting locations within the contamination are irrelevant. We achieve this in three steps.

First, we define a way of measuring a contamination's geometric complexity, rendering insightful analysis possible at all. Second, we propose a cleaning strategy. Third, we precisely analyze what complexity guarantees can be maintained across spreads and agent cleaning. Note that in this paper, we restrict the initial complexity of both contamination boundary and holes. Further, we consider only one single agent. This does not render our contributions invalid,

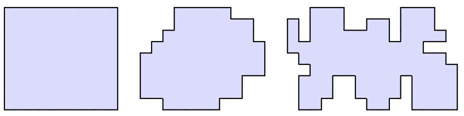


Figure 3: Three contamination complexities from left to right: 0, 1 and 2.

as we do prove them with respect to the given instances. In particular, contribution (i) is not impaired by the one-agent-limit, as we obviously do not make use of any globally maintained data structure, where related work requires one even if using only one agent. The extension to instances with greater complexities and more agents is subject to current research.

## 3 Complexities, Agent Model and Strategy

In order to be able to classify a contamination's complexity, let us make a few definitions. Let $C$ be a contamination. Simply put, a contamination boundary can be considered complex, if it contains spiral shaped parts. As one can easily see, $C$ is enclosed by a simple grid polygon $\mathit{poly}(C)$ that consists of border edge sequences connected by left and right turns. With respect to a clockwise traversal of $\mathit{poly}(C)$, let every left turn be of the value 1, and any right turn of the value $-1$. Then, we define the *complexity* of $C$, $\mathit{comp}(C)$, as the maximum additive subvector of these values. As one can always choose $\mathit{poly}(C)$ subsequences without any turn, $\mathit{comp}(C) \geq 0$. Example contaminations for complexities 0, 1 and 2 are illustrated in Fig. 3. Moreover, let $\mathit{box}(C)$ be the axis-aligned bounding box of $C$, an important figure for contamination size.

In this extended abstract, we limit the initial complexity of contaminations to 1. They may contain arbitrarily many rectangular holes.

Let an agent's position be a contaminated cell $p$ oriented to one of the four cardinal directions. It is equipped with memory of size $O(1)$, can store integers, and perform the operations *increment*, *decrement*, *testzero*, and *setzero* on them. Thus, it cannot implicitly store maps in arbitrarily large integers or similar sophisticated calculations. Any time step, an agent may change orientation, move to a cell $\in \mathit{4nb}(p)$ and clean $p$. It keeps an integer bearing counter for the 90° turns it performed, initialized with 0, increased at right turns and decreased at left turns. An agent can sense the contamination states of any cell in $\mathit{8nb}(p) \cup p$. An agent's goal is to clean all contaminated cells.

We assume that in any time step $t$, first a spread takes place if $t \mod d = 0$, and second the strategy is started (in the first time step) or resumed (in any further time step). Due to the paper's extended abstract nature, we omit a formal writing of the strategy. The agent can be seen as a finite state machine with two internal states: *Search mode* and *boundary mode*.

The agent starts in *search mode* with bearing counter $\omega = 0$. With this value, it moves through the contamination towards the starting orientation, searching for the outer boundary. If it encounters any boundary, it turns right, increasing its $\omega$ to 1. With $\omega \geq 1$, it follows the boundary it found using left hand rule. Note it cannot know, if it has found a neighbour cell of a hole, or the real contamination boundary. However, holes are rectangular by assumption, so if traversing a hole boundary, the agent is guaranteed to turn back left later on. This would make it decrease $\omega$ to 0 again, causing it to move freely through the contamination again. In this manner, the agent leaves any holes encountered without doing any change to them. As it moves monotonously towards two compass orientations in search mode, it eventually encounters the outer contamination boundary.

Once $\omega$ reaches 2, the agent knows to have reached the outer contamination boundary for this value cannot be caused by rectangular holes. It switches to *boundary mode*. In this mode, the agent follows the boundary using left-hand rule independently from $\omega$. Only in boundary mode, cells are cleaned. As we are used to from standard edge peeling, critical cells are not cleaned. Thus, the contamination's connectivity is maintained, so if an agent cleans the last contaminated cell it sees, it knows to have reached the goal and terminates. However, cleaning is controlled even more carefully: The agent starts cleaning cells after having performed a right turn on an uncritical cell. Such a cleaning phase is stopped by turning left or on critical cells. We call cells that are not cleaned because of a left turn or critical cell passed beforehand *uncleanable*. Additionally, *tails* (contaminated cell that touches three border edges) are cleaned independently from these cleaning phases.

Obviously, our strategy does not clean every uncritical boundary cell in encounters. What may seem strange in the first place, does actually enable a straight-forward and insightful analysis. We can prove that while the cleaning of such cells would not contribute to reducing $\mathit{box}(C)$'s size, it would disable us to provide guarantees on the contamination's complexity across agent cleaning operations.

If an agent recognizes a spread, it performs a full reset and switches to search mode again, rendering spreads actually a third way to end cleaning. Spreads are recognized if a clean cell located in an agent's perception range becomes contaminated. In case all cells in an agent's perception range already were contaminated before, an agent is unable to recognize a spread. However, in this case an agent is already in search mode, so spreads are not relevant at the time.

## 4 Correctness and Runtime Analysis Sketch

As stated above, the classification of contaminations' geometrical complexities, as well as providing guarantees on them over time, constitutes the key to a successful strategy and an insightful analysis. During the following paragraphs, let $C$ be a contamination with $comp(C) \leq 1$.

*How spreads change a contamination.* By showing that $poly(C)$ can always be partitioned to a small set of abstract building blocks, followed by an analysis how spreads change these blocks, we can prove that spreads cannot increase a contamination's complexity to a value greater than 1.

*How the agent changes a contamination.* Let an agent be positioned at $p$, and let it decide to clean $p$. By investigating all possible $8nb(p)$ configurations that can occur when an agent decides to clean, we can prove that it cannot create a $poly(C)$ subsequence with a maximum turn subvector greater than 1 either. This is an achievement of our conservative way of cleaning described above.

*Putting up with holes.* In contaminations with such low complexities, obviously no boundary parts can grow together by spreads, so no new holes can emerge. By proving that a low initial complexity is maintained, we achieve not having to take new holes and complexities into account across the entire run time. As an agent does not change any hole, holes stay rectangular until annihilated by spreads – another invariant maintained. Eventually, after the last hole disappeared, the contamination's simply-connectedness is also maintained from this point of time on.

Simply put, this leads to the following agent behavior. The agent finds the boundary and peels cells from it. If the contamination is simply-connected, we know that it does so until the contamination is cleaned because in this case, we can prove there are always cells cleanable for the agent. If holes exist, it cleans until the boundary is rendered entirely uncleanable due to critical cells caused by these holes (Fig. 4). In the latter case, the agent makes a virtue out of necessity and waits for the next spread, traversing the boundary on and on without a chance to clean. We can prove, that after a spread happened, an agent is guaranteed to be able to clean as much cells as are necessary to shrink both height and width of the contamination's bounding box by 4 cells, before cleaning can possibly be obstructed again by critical cells. All in all, we can prove our strategy's correctness: The agent completely cleans the contamination, if it spreads slow enough. More formally:

**Theorem 1 (Correctness)** *For any connected contamination $C$ with initial $comp(C) \leq 1$ featuring arbitrarily many rectangular holes, there exists a spreading time $d_{\min}$ so that one agent using our strategy reaches the goal and terminates. Moreover, the strategy terminates for any $d > d_{\min}$.*

Furthermore, we can prove that a spreading time of $d_{\min} \geq 5w + 5h + 10$ suffices, where $w$ is the width and $h$ is the height of $box(C)$. In this case, an agent

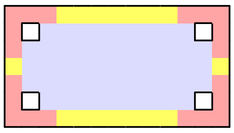


Figure 4: A contamination rendered completely uncleanable by four minimal holes.

can clean simply-connected contamination instances in an optimal $O(wh)$ time. As we already mentioned, it may be that the agent has to wait for spreads in order to get rid of holes. Let $l$ be the number of spreads needed to annihilate the thickest hole. Then, for the same $d_{\min}$, we further can prove, that the agent reaches its goal in $O(ld + wh)$ time.

## 5 Conclusions and future work

We presented the first solution to the problem of cleaning complexity-restricted instances of dynamic domains that (i) operates without central guidance and global knowledge, and (ii) generalizes from simply-connectedness to connectedness. We did so by precisely analysing geometrical changes in the contamination over time. Obviously, there remain interesting questions. For instance, one might wish for cleaning arbitrarily complex contaminations, or for strategies to be implemented by more than one agent. The mathematical analysis of both topics is subject to our current research.

## References

[1] H. Abelson and A.A. DiSessa. *Turtle geometry: The computer as a medium for exploring mathematics.* The MIT Press, 1986.

[2] Y. Altshuler and A.M. Bruckstein. Static and expanding grid coverage with ant robots: Complexity results. *Theoretical computer science*, 412(35):4661–4674, 2011.

[3] Y. Altshuler, V. Yanovski, I.A. Wagner, and A.M. Bruckstein. Multi-agent cooperative cleaning of expanding domains. *The International Journal of Robotics Research*, 30(8):1037–1071, 2011.

[4] H. Choset and P. Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *International Conference on Field and Service Robotics.* Citeseer, 1997.

[5] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24(3):197–224, 2003.

[6] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring simple grid polygons. *Computing and Combinatorics*, pages 524–533, 2005.

[7] I.A. Wagner, Y. Altshuler, V. Yanovski, and A.M. Bruckstein. Cooperative cleaners: A study in ant robotics. *The International Journal of Robotics Research*, 27(1):127–151, 2008.

# 2-modem Pursuit-Evasion Problem

Y. Bahoo [*] A. Mohades [*] M. Eskandari [†] M. sorouri [†]

## Abstract

In this paper we introduce a new version of the Pursuit-Evasion problem in which the pursuer is a 2-modem which pursues an unpredictable evader in a polygonal environment. A 2-modem searcher is a wireless device whose radio signal can penetrate two walls. We will present a new cell decomposition of a given polygon $P$ for the 2-modem searcher such that the combinatorial representation of the invisible regions of the searcher remains unchanged. In other words, when the searcher moves inside a cell, no evader can move from an invisible region to another one without detecting by the pursuer.

## 1 Introduction

Consider a simple polygon $P$ is given and there are some evaders and a pursuer which moves continuously in it. The classical pursuit-evasion problem asks for planning the motion of the pursuers in a polygon to eventually see an evader. During the motion of the pursuer, some parts of polygon may be invisible for him; these invisible regions completely depend on the type of the pursuer and its position in $P$. Let $p$ be an arbitrary point in $P$ (as an initial position of the searcher). A maximal connected closed set of points inside $P$ which are invisible for $p$ is called a shadow of $p$. Actually the shadows of $p$ are the subpolygons of $P$ which are denoted by $S_i(p)$. As shown in [1], when the searcher moves continuously inside $P$, four geometric events may happen for its shadows: merge, split, appear and disappear. Moreover, when two disjoint shadows of $p$ merge together and make one connected subpolygon, it is called the *merge event*. In contrast, when a shadow is divided into two components during the motion of the searcher, it is called the *split event*. Sometimes a shadow is destroyed when the searcher moves; this event is known as the *disappear event* and if a new shadow is created, we call it the *appear event*.

In [2], Guibas et al studied the problem of maintaining the distribution of evaders that move out of view and inferring the location of these targets from combinatorial data extracted by searchers. In this paper, we consider a special type of searchers, 2-modems. As defined in [3], we call a wireless device whose radio signal can penetrate two walls, a 2-modem. We will present a new cell decomposition of a given polygon $P$ for a 2-modem searcher such that the combinatorial representation of its shadows remains unchanged. In other words, when the searcher moves continuously inside a cell of this decomposition, the *merge*, *appear*, *disappear* or *split* event does not occur. The study of this problem is motivated by robotics applications such as surveillance, as explained in [2].

[*]Department of mathematics and computer sciences, Amirkabir university of technology, bahoo@aut.ac.ir, mohades@aut.ac.ir

[†]Department of mathematics, Alzahra university, eskandari@alzahra.ac.ir, sorouri@alzahra.ac.ir

## 2 The 2-Cell Decomposition

In this section, we introduce our new decomposition of a given polygon $P$ into convex cells, which provides our main tool for avoiding four events defined above. It is called the *2-cell decomposition.*

**Definition 1** *Let $v$ and $u$ be two vertices of $P$. The vertex $u$ is a* critical vertex *for $v$ if both of its edges are in the same half-plane bounded by the line $uv$.*

The 2-cell decomposition is created by three kinds of lines which are called the *partition lines*:
1)The lines that are the extensions of both edges of the reflex vertices of $P$ in it.
2)The portions of the lines through a pair of reflex vertices which are critical for each other.
3)The lines used in the *2-modem visibility polygon* of each vertex of the polygon which is introduced in [3].

The 2-visibility polygon of a vertex $v \in P$ is a subpolygon of $P$ which is visible by a 2-modem lied on $v$. Now we determine the exact portions of these lines which contribute to the 2-cell decomposition.

**Constructing the 2-cell decomposition of P:**

i) For each reflex vertex of $P$, draw the extensions of its edges until they hit the boundary of $P$.
ii) For each pair of reflex vertices $u$ and $v$ which are critical for each other and $uv \in P$, draw the line through $uv$ until it hits $\partial P$ and then omit $uv$.
iii) For each vertex $v$ of $P$, construct the 2-visibility polygon of $v$ as defined in [3].

For illustration of the lines of type 3 which are in the 2-visibility polygon of the vertices, we provide some examples. Let $u$ be an arbitrary vertex of $P$ and $v$ be a critical vertex of $u$. In Figure 1, we illustrate the parts of the ray $uv$ which is drawn in 2-cell decomposition by the bold pieces.

**Observation 1** *The third type of the segments in the 2-cell decomposition guarantees neither merge nor*

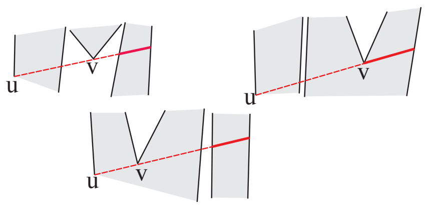


Figure 1: The bold parts are drawn in 2-cell decomposition.

*split event happen while moving within one cell of the decomposition.*

Now we would like to show that neither appear nor disappear event occurs when the 2-modem searcher moves inside a cell. For this purpose, we categorize the shadows by having at least one vertex of the polygon inside them or not. Thereby, two types of shadows can be defined as follows:

**Definition 2** *The* type 1 *shadow is a shadow which has at least one vertex of the polygon and the* type 2 *shadow is a shadow which has no vertex of the polygon; it occurs between two edges of the polygon.*

**Theorem 1** *When the 2-modem searcher moves continuously in a cell, no vertices of the polygon may enter into or exit from its shadow.*

**Proof.** Let $R$ be a cell in the 2-cell decomposition of $P$. At first we will show that if a searcher at an arbitrary point $p$ in $R$ has a shadow of type 1, during the moving from $p$ inside $R$ no vertices of $P$ can exit from its shadow. Let $q$ be another arbitrary point inside $R$ such that the searcher has moved to $q$. Let $v$ be a vertex of $P$ which is in $S(p)$. Since the cells are convex, the segment $pq$ is completely inside $R$. We suppose for a contradiction that $v$ does not belong to $S(q)$, so the segment $vp$ must intersect the polygon $P$ at least three times, but the segment $vq$ can intersect $P$ at most two times. So there is at least one vertex inside the triangle $pqv$ that is critical for the vertex $v$ (otherwise $v$ will be visible for $p$). We rotate the ray $\overrightarrow{vq}$ around $v$ towards inside the triangle $pqv$ until reach the first critical vertex for $v$. This vertex is denoted by $r$. The supporting line of the segment $vr$ is one of the partition lines and intersects the segment $pq$. Hence $p$ and $q$ are not in a same cell, a contradiction. Now similarly, we can show that if a searcher (with a shadow of type 1 or type 2) moves continuously in a cell, no vertices of $P$ can enter into its shadow. □

Notice that by Theorem 1, we conclude that if a 2-modem searcher which has a shadow of type 1, moves in the cell, the appear and the disappear events cannot occur for its shadow. Now we will prove this fact for the case of the type 2 shadow.

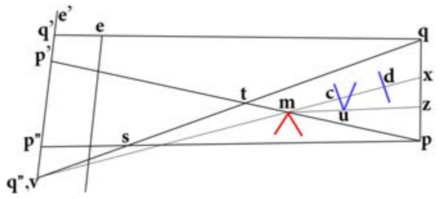


Figure 2: $mx$ intersects $\partial P$ more than once.

**Theorem 2** *If a 2-modem searcher which is contained in a cell $R$ and has a shadow of type 2, moves continuously in $R$, the appear and the disappear events cannot occur for its shadow.*

**Proof.** Assume that the 2-modem searcher lies on a point $p$ in $R$ and it has a shadow of type 2 which occurs between two edges of the polygon, named $e$ and $e'$. Let $q$ be an arbitrary point inside $R$ such that the searcher moves to $q$. Now we erect a coordinate system with y-axis lined up with the ray $\overrightarrow{pq}$ and the origin at $p$. We connect the point $q$ to the endpoints of $e$ and $e'$ and consider two of these line segments for constructing a triangle named $qq'q$" such that both segments $qq'$ and $qq$" intersect both edges $e$ and $e'$, see Figure 2. Also the shadow of the point $p$ occurs between two rays emitted from the point $p$. The intersections of these rays with $e'$ are called by $p'$ and $p$". See Figure 2. If there exists a portion of the triangle $qq'q$" enclosed by two edges $e$ and $e'$ which is not visible for the point $q$, we are done. Otherwise, suppose that the point $q$ can see the whole of this region. As shown in Figure 2, there is always a vertex of polygon $P$ on the line segment $pp'$ such that both edges of $m$ lie below the segment $pp'$ (because the shadow of $p$ is started by the ray $\overrightarrow{pp'}$). The points $t$ and $s$ are the intersections of the line $qq$" with the lines $pp'$ and $pp$", respectively. $e_m$ is one of the edges of the vertex $m$ which makes the smaller angle with the positive $x$-axis. Now we consider two cases, the supporting line of edge $e_m$ intersects the segment $pq$ or not. At first, we suppose that it does not have an intersection with $pq$. In this case, the vertex $m$ should lie on the segment $pt$. For this, we will show that it cannot lie on the segment $tp'$. For a contradiction, suppose that $m$ is on the segment $tp'$. Since the supporting line of $e_m$ does not intersect the segment $pq$, the edge $e_m$ will lie below the line $qm$. Therefore a portion of the edge $e'$ will be invisible for the point $q$ and this is a contradiction. Consequently, the vertex $m$ should be on the segment $pt$. Now we will show that if $m$ lies on the segment $pt$, the points $p$ and $q$ cannot be in a same cell of our decomposition, which is a contradiction.

We assume that the vertex $v$ is one of the two vertices of the endpoints of the edges $e$ or $e'$ which is on the segment $qq$". Since the supporting line of the edge $e_m$ does not have an intersection with the segment $pq$ and the vertex $m$ is on the segment $pt$, the line $e_m$ should lie below the line $vm$. Therefore the other edge of the vertex $m$ should be below the line $vm$ (according to the definition of the edge $e_m$). Hence, the vertex $m$ is a critical vertex for the vertex $v$. We denote the intersection point of the supporting line of $vm$ and the segment $pq$ by $x$ (see Figure 2). Since $p$ and $q$ are inside $P$, the segment $vx$ intersects the polygon, one or more times. If it intersects $P$ at exactly one point, the supporting line of $vm$ is again one of the partition lines and intersects the segment $pq$ and it means that the points $p$ and $q$ are not in a same cell, a contradiction. So we can assume that the segment $vm$ intersects $P$ in at least two points. First we assume that the segment $vx$ intersects polygon $P$ more than once and the segment $vm$ intersects $P$ just once, see Figure 2. Note that the segment $mx$ intersects $P$ at least two times (because the number of intersections must be odd). Let $c$ and $d$ denote these intersections. Consider two hypothetical walkers $C$ and $D$ who traverse the boundary of the polygon $P$ such that they enter into the triangle $mxp$, starting from the points $c$ and $d$ respectively. These two walkers must cross the segment $mx$ or the segment $pm$ for leaving triangle $mxp$. Note that if the walkers hit $px$, then segment $pq$ will not lie completely inside $P$, that is a contradiction. Note that since the shadow region $S(p)$ is started by the ray $\overrightarrow{pp'}$, at most one of the walkers $D$ or $C$ can cross the segment $pm$, thus at least one of them can cross the segment $mx$ $(C)$. The polygonal chain traversed by the walker $C$ in triangle $mxp$ is denoted by $CH(C)$. We rotate the ray $\overrightarrow{mp}$ around $m$ towards inside the triangle $mxp$ until we reach the first vertex of $CH(C)$. It is clear that it is critical for the vertex $m$. Hence in the triangle $mxp$, there exists at least one vertex of $P$ which is critical for $m$. So we can rotate the ray $\overrightarrow{mp}$ around the point $m$ towards inside the triangle $mxp$, until we reach the first "critical" vertex for $m$ in the triangle $mxp$, which is denoted by $u$. See Figure 2. The intersection of the supporting line of $mu$ and the segment $pq$ is denoted by $z$. We distinguish two cases: the polygon $P$ intersects the segment $mz$ or not. If it intersects the segment $mz$, it will not intersect more than once because the vertex $u$ is the first vertex which is reached by rotating $mx$, so there is a part of the ray $\overrightarrow{us}$ which belongs to the partition lines and intersects the segment $pq$ at the point $z$, a contradiction. Now if the polygon $P$ does not intersect the segment $mz$, so the vertices $u$ and $m$ are reflex and critical for each other, then the segment $uz$ belongs to the partition lines and that is a contradiction. Now we can assume that the segment $vm$ intersects $P$ in at least two points, $a$ and $b$. See Figure 3. Consider two hypothetical walkers $A$ and $B$ who traverse the boundary of the polygon $P$ such that they enter into the triangle $vmp$, starting from the points $a$ and $b$ respectively. One of these two walkers should pass over the line $vm$. Because otherwise, the walkers should intersect the line $tm$ or the line $vt$. It is clear that both of them cannot leave the triangle $vmt$ from the line $tm$. Because the ray $\overrightarrow{pp'}$ determines the border of the shadow region $S(p)$. Thus at least one of them $(A)$ should cross the segment $vt$. In this way, the walker $A$ enters into the triangle $vtp'$, and for leaving this triangle it should cross the line $p't$. Because otherwise, it will be an obstacle for the point $q$ (it can only cross the line $qq$"). On the other hand, the walker $B$ cannot cross the line $tm$, because the walker $A$ intersects the line $p't$. The walker $B$ cannot cross the same line, because the ray $\overrightarrow{pp'}$ determines the starting of the shadow region $S(p)$. Also the walker $B$ cannot cross the line $vt$. Because otherwise, it will be an obstacle for $q$ (note that according to our assumption the walker $A$ crosses the line $vt$). So the walker $B$ should intersect the line $vm$. The polygonal chain traversed by the walker $B$ in triangle $vtm$ is denoted by $CH(B)$ . We rotate the ray $\overrightarrow{mt}$ around $m$ towards inside the triangle $vtm$ until we reach the first vertex of $CH(B)$. Obviously, this vertex is critical for the vertex $m$. Then we can assume that there exists always a critical vertex for $m$ in the triangle $vtm$. So we can rotate the ray $\overrightarrow{mt}$ around the point $m$ towards inside the triangle $vtm$, until we reach the first "critical" vertex for $m$ in the triangle $vtm$, which is denoted by $v'$. It is clear the vertices $v'$ and $m$ are critical for each other. The intersection of the ray $\overrightarrow{mv'}$ and the segment $pq$ is denoted by $y$. Note that the segment $mv'$ intersects $P$ at most once (otherwise the vertex $v'$ is not the first critical vertex). We distinguish two cases: the polygon $P$ doesn't intersect the segment $mv'$ or just one intersection occurs. In the first case, it is clear that the vertices $v'$ and $m$ are reflex and critical for each other, so the supporting line of $v'm$ will be one of the partition lines and intersects the segment $pq$ and it means that the points $p$ and $q$ are not in a same cell, a contradiction. In the second case, when the polygon $P$ just has one intersection with the supporting line of $mv'$, there exists a part of the segment $my$ which belongs to a partition line. It can be shown that the points $p$ and $q$ are not in a same cell which is a contradiction (similar to the way described above for the case of the vertex $u$ is a critical vertex for the vertex $m$). According to the above discussion, we have reached a contradiction when the edge $e_m$ does not intersect the segment $pq$, hence the sup-

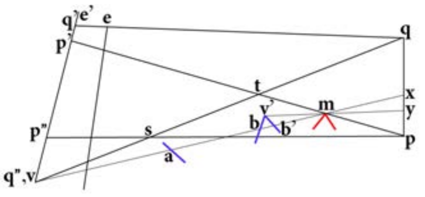


Figure 3: $mv$ intersects $\partial P$ more than once.

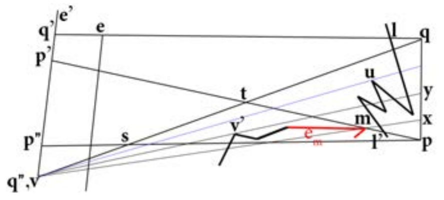


Figure 4: $e_m$ intersects $pq$.

porting line of $e_m$ must intersect the segment $pq$. See Figure 4. Note that the point $q$ is inside the polygon $P$, then the segment $qq'$ must have at least another intersection with the polygon, except at $e$. Also, we know that the point $q'$ is visible for $q$, hence the segment $qq'$ should intersect $P$ at most once, except at $e$. This intersection is denoted by $l$. Now we consider a hypothetical walker $L$ who traverses the boundary of $P$ inside the triangle $qq'q$". This polygonal chain is denoted by $CH(L)$. The walker $L$ should intersect the segment $pp'$ for leaving the quadrilateral $p'q'pq$. This intersection point is denoted by $l'$. We distinguish two cases: the point $l'$ lies on the segment $mp'$ or $pm$. In both cases we will reach a contradiction. First, we suppose the point $l'$ lies on the segment $mp'$. In this case, it is clear that the segment $mq$ cannot intersect the polygon $P$ (otherwise the point $q'$ is invisible for the point $q$ or $P$ is not a simple polygon). In addition, it is easy to show that the vertex $m$ is a reflex vertex (because of the number of intersections between the segment $mq$ and $P$ and $q$ is an interior point of $P$). Thus the supporting line of $e_m$ is one of the partition lines and intersects the segment $pq$ and it means that the points $p$ and $q$ are not in a same cell, a contradiction. In the second case where we consider the point $l'$ lies on the segment $mp$, the vertex $m$ must be on the segment $tp$. Otherwise, since $l'$ is on the segment $mp$, the polygonal chain traversed by walker $L$ from $l$ to $l'$ is intersected by the segment $mq$, so $q$ cannot see whole segment $qq'$. Therefore the edge $e_m$ will be inside the triangle $vtm$ and it cannot cross the segment $tm$ (note that the shadow region $S(p)$ is started by the ray $\overrightarrow{pp'}$), furthermore it cannot cross the segment $tv$ (because $L$ crosses the segment $qq$" once, hence the vertex $q$" is not visible by $q$). Thus the polygonal chain in the triangle $vtm$ crosses the segment $vm$ at least one time. Thereby there exists always one vertex inside the triangle $tvm$. We rotate the ray $\overrightarrow{vt}$ around the point $v$ towards inside the triangle $vtm$, until we reach the first "critical" vertex for $v$ in the triangle $vtm$ (because $v'$ is the first vertex, the polygon $P$ doesn't intersect the segment $qq$" and the vertex $q$ should see $q$"). This vertex is denoted by $v'$. The intersection of the supporting line of $vv'$ and the segment $pq$ is denoted by $y$. If the segment $v'y$ intersects the polygon just once (on $CH(L)$), the supporting line of $v'y$ is one of the partition lines (because vertex $v'$ is a critical vertex for the vertex $v$) and intersects the segment $pq$ and it means that the points $p$ and $q$ are not in a same cell, a contradiction.

Then we suppose that segment $v'y$ intersect $P$ more than once. All of these intersections are inside the triangle $tqp$ (when the ray $\overrightarrow{vt}$ is rotated towards inside the triangle $vtm$, the vertex $v'$ is the first vertex). The segment $qq$" should intersect the polygon $P$ just on $CH(L)$ ($q$ should see $q$"). So if the segment $vy$ intersects the polygon $P$ except on $CH(L)$, then there exists a vertex inside region $S = \triangle tpq \cap \triangle qq"y$. We rotate the ray $\overrightarrow{vq}$ around the vertex $v$ toward in triangle $tpq$ until we reach the first "critical" vertex for $v$ in the region $S$ which is denoted by $u$. The supporting line of $uv$ intersects $P$ just once on $CH(L)$ (otherwise the vertex $v'$ is not the first critical vertex). Thus the supporting line of $uv$ is one of the partition lines and intersects the segment $pq$ and it means that the points $p$ and $q$ are not in a same cell, a contradiction. In all above cases, we showed that there is always a part of the segment $q'q$" which is not visible from $q$. Hence $q$ should have a shadow between $e$ and $e'$.

Also in a similar way, it can be shown that no appear event occurs. $\square$

Due to Theorems 1 and 2, when a 2-modem moves continuously in a cell, neither disappear nor appear event can happen, i. e., the 2-cell decomposition guaranties that the combinatorial representation of the invisible regions of the searcher remains unchanged.

## 3 Conclusion

In this study, we considered a new version of Pursuit-Evasion problem and introduced a new decomposition of a given polygon into convex cells which assures that no evader can move from an invisible region to another one without detecting by the pursuer while the searcher moves inside a cell. Moreover it can be shown that the number of cells in the 2-cell decomposition is $O(n^3)$, but it takes a bit of effort. Also the complexity of the algorithm is the same.

## References

[1] J. Yu and S. M. LaValle. *Shadow information spaces: Combinatorial filters for tracking targets.* IEEE Transactions on Robotics, 28(2):440-456, 2012.

[2] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. *Visibility-based pursuit-evasion in a polygonal environment.* International Journal of Computational Geometry and Applications, 9(5):471-494, 1999.

[3] O. Aichholzer, R. Fabila-Monroy, D. Flores-Pealoza, T. Hackl, C. Huemer, J. Urrutia, and B. Vogtenhuber. *Modem Illumination of Monotone Polygons* Proc. 25th European Workshop on Computational Geometry EuroCG09, pages 167-170, Brussels, Belgium, 2009.

# Euclidean Traveling Salesman Tours through Stochastic Neighborhoods

Pegah Kamousi*　　Subhash Suri †

## 1 Introduction

We consider a variant of the classical traveling salesman problem under a stochastic scenario. Our setting requires planning an optimal tour that visits each of the $n$ regions in the plane, called *neighborhoods*, under the Euclidean metric. The regions are disks whose radii are random variables drawn independently and identically from a known probability distribution, and so a random instance of the problem involves an arbitrary set of disks, with varying radii and an arbitrary overlap pattern. Our problem is to minimize the *expected* length of the tour visiting these disks; the problem is clearly $NP$-hard because it subsumes the classical Euclidean TSP by setting the mean and the variance of the probability distribution to zero. This is a *stochastic* version of the classic traveling salesman problem with neighborhoods (TSPN). This problem, fundamental in its own right, is also motivated by a number of applications such as the data gathering problem in sensor networks: a robotic *data mule* needs to collect data from $n$ geographically distributed wireless sensor nodes whose communication range $r$ is a random variable subject to a number of environmental factors.

In the offline case, we assume that the algorithm knows the input instance at the start of the tour, while in the online case the radii of the disks are revealed only when the tour reaches each disk. We prove the following results in this paper: (1) We can compute a traveling salesman tour through $n$ stochastic disks whose expected length is within factor $O(\log\log n)$ of $\mathbb{E}[L^*]$, the expected optimal, in polynomial time, (2) If the radii of the stochastic disks are revealed online, our algorithm achieves an $O(\log n)$ approximation of $\mathbb{E}[L^*]$, (3) If the disks are (nearly) disjoint when they all appear with the mean radius $\mu$, then the approximation factor improves to $O(1)$ in both the offline and the online cases.

## 2 Related Work

The traveling salesman problem with neighborhoods (TSPN) was first introduced by Arkin and Hassin [1]. The problem is known to be APX-hard when the neighborhoods are general overlapping polygons [2], and therefore the approximation algorithms have focused on either disjoint or "fat" neighborhoods. In particular, if the regions are (nearly) disjoint disks of (nearly) identical size, then there exists a PTAS [3]. Other approximations for special cases can be found in [2, 7, 8]. Without the assumption of (near) disjointness, the approximation results have tended to assume *regions with comparable diameters* [3, 4, 5].

When the regions are neither disjoint nor of roughly the same size, the best approximation ratio known is $O(\log n)$ [4, 6]. In our setting, the stochastic disks can have arbitrarily large radii and overlap in arbitrary ways, and so the prior work does not give an approximation ratio better than $O(\log n)$. When the radii are revealed *online*, no prior work seems to be known. In our stochastic setting, instead of comparing the performance of the algorithm for every single realization, we are interested in the *expected* performance over all the realizations.

## 3 Preliminaries

Let $\mathcal{D} = \{D_1, D_2, \dots, D_n\}$ be a set of $n$ random disks in the plane, where each disk $D_i = (c_i, r_i)$ has a fixed center $c_i$, but its radius $r_i$ is a random variable drawn independently and identically from a probability distribution with mean $\mu$ (we impose some constraints on the probability distribution, which we eliminate from this short abstract). With the disk centers fixed, we may view the set $\mathcal{D}$ as a multivariate random variable (or vector) $\mathcal{R} = (r_1, r_2, \dots, r_n)$. Let $\mathcal{I}_{\mathcal{R}}$ denote the set of all the possible instances (realizations) of the vector $\mathcal{R}$. Each $I \in \mathcal{I}_{\mathcal{R}}$ uniquely identifies a particular instance of our TSP with neighborhoods. The probability distribution of $\mathcal{R}$, denoted $\phi_{\mathcal{R}}$, can be obtained from the marginal distributions of the radii. That is, for an instance $I = (x_1, x_2, \dots, x_n)$, we have

$$\phi_{\mathcal{R}}(I) \;=\; \phi_{\mathcal{R}}(x_1, x_2, \dots, x_n) \;=\; \prod_{i=1}^{n} \phi(x_i), \text{ where}$$

$$\int_{x_1} \cdots \int_{x_n} \phi(x_1) \cdots \phi(x_n)\, dx_1 \cdots dx_n \;=\; 1\,.$$

The expected value of $\mathcal{R}$ is the vector $\mu^{(n)}$, referring to the instance where each of the $n$ disks has the radius equal to the mean value $\mu$. We Let $M$ denote

*Université Libre de Bruxelles, pegah@cs.ucsb.edu
†University of California, Santa Barbara, suri@cs.ucsb.edu

this special instance. The optimal TSPN tour for the instance $M$ is called $\textsc{Opt}(M)$. Let the random variable $L^*$ measure the length of the shortest tour over the sample space $\mathcal{D}$. Then we can write $\mathbb{E}[L^*]$ as

$$\int_0^\infty \cdots \int_0^\infty L^*(x_1,\ldots,x_n)\cdot\phi_{\mathcal{R}}(x_1,\ldots,x_n)\,dx_1\ldots dx_n,$$

where $L^*(I)$ denotes the value of $L^*$ for instance $I$.

Given any polygonal path or cycle $T$, we use $|T|$ to denote its Euclidean length, which is simply the sum of the lengths of its segments. To simplify our presentation, we also assume a fixed start point $s_0$ that lies at least $2\mu$ away from all the disk centers.

We begin with a theorem establishing the importance of the instance $M$, showing that the *optimal of the mean is a good lower bound on the mean of the optimal.*

**Theorem 1** $|\textsc{Opt}(M)| \leq 2\,\mathbb{E}[L^*]$.

## 3.1 The High Level Strategy

All of our stochastic TSP algorithms employ the following three-step strategy: first, we compute a constant factor approximation $T(M)$ for the instance $M$—that is, $|T(M)| = O(|\textsc{Opt}(M)|)$; second, we subdivide $T(M)$ into several *blocks* and assign a subset of the disks to each block; finally, for a random instance $I$, we construct an approximate tour by visiting disks in the block order given by $T(M)$. (We note that simply following $T(M)$ may miss some of the disks in a random instance, since their radii in $I$ could be smaller than the mean value; So, the block order is just a high-level clue about how "subsets" of disks should be visited.)

In the rest of this section, we describe the first (and the simplest) of these three steps, while the other steps are the focus of next sections. Our algorithm for approximating the TSP for the mean radius instance $M$ is based on some classical ideas for approximating the TSP of disks. In particular, if the neighborhoods are convex regions of equal diameter in the plane, then a polynomial algorithm is known for a constant factor approximation of the TSP visiting all the neighborhoods [1, 3]. The approximation algorithm works by choosing a *representative point* in each convex region and finding an almost optimal tour of these points.

In this spirit, consider the instance $M$, which has $n$ (possibly intersecting) disks, each of radius $\mu$. We call a set of vertical lines a *line cover*, if each disk is intersected by at least one of these lines. Such a line cover with the minimum number of lines is easily computed by a simple greedy scan: the first line is chosen to pass through the rightmost point of the leftmost disk; remove all the disks intersected by this line, and repeat until all the disks are covered. We can make two simple observations: first, each disk is intersected by precisely one line in the cover, which we call the *covering line* of this disk; and second, two adjacent lines of the cover are at least $2\mu$ apart. See Figure 1(a) for an illustration. For each disk $D_i$, the point where the covering line of $D_i$ meets its horizontal diameter is selected as its unique *representative point.* Following the algorithm of [1, 3], we then compute a $(1+\epsilon)$-approximate tour of these representative points. Call this tour $T(M)$. Then, by Theorem 1, we have the important result that $T(M) = O(\mathbb{E}[L^*])$.

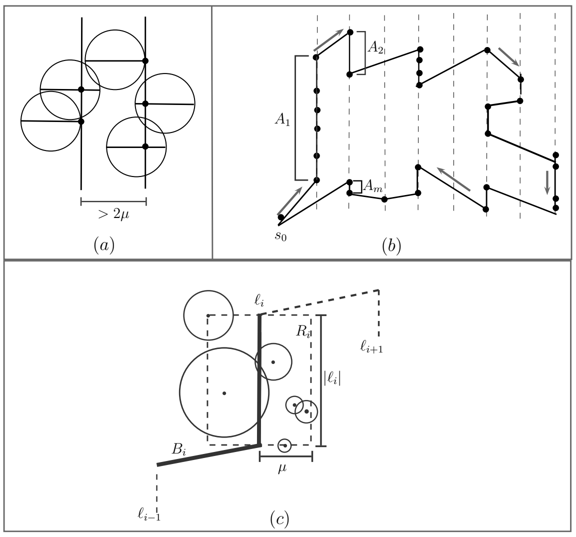


Figure 1: (a) A set of covering lines; (b) a possible structure of $T(M)$ (c) a block $B_i$ and rectangle $R_i$.

Let us fix an orientation of $T(M)$, say, clockwise, and let $A = \{a_1,\ldots,a_n\}$ denote the sequence of representative points of disks in $M$ in the order they are visited by $T(M)$; we assume that the sequence begins with the first disk visited following the initial point $s_0$. Recall that all the $n$ representatives lie on the covering lines, which have a minimum separation of $2\mu$ between them. We now partition the sequence $A$ into chunks of consecutive points $A_1, A_2, \ldots, A_m$, such that each chunk contains points that belong to the same covering line *and* are consecutive along the tour $T(M)$. Without loss of generality, let $A_0$ consist of the singleton initial point $s_0$. See Figure 1(b) for an example.

Let $\ell_i$, for $i = 1, 2, \ldots, m$, be the *line segment* joining the *lowest* and the *highest* (by $y$-coordinate) point in $A_i$. Clearly, $\ell_i$ covers all the points in $A_i$, and thus visits the mean radius disks associated with these points. We will use these chunks $A_i$ to divide the tour $T(M)$ into $m$ blocks $B_1,\ldots,B_m$, where $B_i$ is the portion of $T(M)$ visiting all the points in $A_i$ together with the line segment connecting the last point in $A_{i-1}$ to the first point in $A_i$. That is, $B_i$ is the part of $T(M)$ that starts after its last contact with $\ell_{i-1}$ and ends right after its last contact with $\ell_i$. See Figure 1(c).

Since the minimum distance between $\ell_i$ and $\ell_{i+1}$ is $2\mu$, we can lower bound the length of the block $B_i$ by $2\mu + |\ell_i|$; the initial block $|B_1|$, being an exception, is lower bounded by $\mu + |\ell_1|$, since $s_0$ is at least $\mu$ away from its closest disk. The following observation immediately follows Theorem 1 and the fact that $|T(M)| = \sum_{i=1}^{m} |B_i|$.

**Observation 3.1** $\sum_{i=1}^{m} |B_i| = O(\mathbb{E}[L^*])$.

We say that all the disks covered by $\ell_i$ are *assigned* to the block $B_i$, and these blocks form the desired *partial order* on our input disks: all disks assigned to block $i$ precede any disk assigned to block $j$ if $i < j$.

By construction, the centers of all the disks assigned to block $B_i$ lie within the rectangle $R_i$ of dimensions $|\ell_i| \times 2\mu$, with vertical axis $\ell_i$ (see Figure 1(c)). We will argue that for any random instance, by visiting all the disks of each block in the partial order imposed by blocks, we obtain a tour that achieves a $O(\log\log n)$ factor approximation of the expected optimum.

Our discussion so far applies to the general stochastic TSPN problem: computing the approximately optimal tour for the mean instance $M$, the partial ordering of disks and the block partition all only require knowledge of the mean radius and the disk centers. However, the last key step that computes a good approximation tour for each block $B_i$ crucially depends on whether we know the radii of the random instance beforehand or not. Therefore, the following discussion now separately considers the *offline* and the *online* versions of the problem.

## 4 Stochastic Offline Tour

In this section, we describe an algorithm for visiting the stochastic disks in the offline setting: the salesman knows the disk radii of the given instance before starting the tour. We show how to construct a tour whose expected length is within factor $O(\log\log n)$ of the expected optimal.

In light of the discussion of the previous section, we only need to focus on constructing approximately optimal tours for each block $B_i$, for $i = 1, \dots, m$, because their concatenation leads to an overall tour with length close to $\mathbb{E}[L^*]$. We first recall that the centers of all the disks assigned to $B_i$ lie within the (closed) rectangle $R_i$ with dimensions $|\ell_i| \times 2\mu$, and centered at the midpoint of $\ell_i$. We partition $R_i$ into $2\mu \times 2\mu$ squares; (the last "square" may be a rectangle of width $2\mu$ and height smaller than $2\mu$). We construct the tour separately for each of these squares, visiting the disks *whose centers lie in the square*. The concatenation of these subtours gives the final tour. With this preamble, the next subsection considers the following key problem: given $n$ disks whose centers lie inside a square of side length $2\mu$, construct a tour visiting them. We then explain and analyze the algorithm to combine these subtours in subsection 4.2.

### 4.1 Constructing a Subtour within a Square

Let $\mathcal{D} = \{D_1, \dots, D_n\}$ be a random instance of the stochastic TSPN problem where the centers of all the disks lie inside a square $R$ of dimensions $2\mu \times 2\mu$, where $\mu$ is the mean radius of the disks. We show how to construct a tour visiting these disks with expected length $O(\log\log n)$ times the expected optimal. We begin with an idea used in the work of Elbassioni et. al. [4] for the deterministic TSPN problem on intersecting neighborhoods.

First, let $\mathcal{D}_{\text{in}} \subseteq \mathcal{D}$ denote the set of disks contained in the interior of $R$. If $\mathcal{D}_{\text{in}} = \emptyset$, then the boundary of $R$ visits all the disks, and this is an easy case. Otherwise, $\mathcal{D}_{\text{in}} \neq \emptyset$, and we proceed as follows. We let $N_2(D_i) \subseteq \mathcal{D}$ denote the *2-neighborhood* of disk $D_i = (c_i, r_i)$, which is the set of disks in $\mathcal{D}$ within distance $2r_i$ of $c_i$. That is, $N_2(D_i)$ is the set of disks that intersect a disk of radius $2r_i$ centered at $c_i$. We call the disk $D_i$ the *core* of $N_2(D_i)$. We use the 2-neighborhoods to form a disjoint cover of $\mathcal{D}_{\text{in}}$, by the following iterative algorithm.

Initially, $\mathcal{N} = \emptyset$. Choose the disk $D_i \in \mathcal{D}$ with the smallest radius, and add the 2-neighborhood $N_2(D_i)$ to $\mathcal{N}$, with $D_i$ as its *core*. Remove all the disks of $N_2(D_i)$ from $\mathcal{D}$, and iterate until $\mathcal{D}$ is empty. Clearly, each disk $D_j \in \mathcal{D}_{\text{in}}$ is assigned to $\mathcal{N}$ at some point, and we identify it with the core disk $D_i$ whose 2-neighborhood added $D_j$ to $\mathcal{N}$. Without loss of generality, suppose $D_1, D_2, \dots, D_k$ are the core disks selected by the covering algorithm in this order. Clearly, by the disk selection rule, any two core disks are disjoint, that is, $D_i \cap D_j = \emptyset$, for $1 \leq i, j \leq k$, and the radii are in increasing order, namely, $r_1 \leq r_2 \leq \cdots \leq r_k$.

**Lemma 2** *Let $N'(D_i) \subseteq N_2(D_i)$ be the set of disks added to $\mathcal{N}$ when $D_i$ is chosen as core. Then there is a tour of length at most $O(r_i)$ visiting all the disks of $N'(D_i)$.*

Let $\text{O}\small{\text{PT}}'$ denotes an optimal tour that visits all the disks of $\mathcal{D}_{\text{in}}$. The following key lemma gives a lower bound on $|\text{O}\small{\text{PT}}'|$ for *any* instance of the problem in terms of just the radii of core disks. An analog of this Lemma can also be found in [4].

**Lemma 3** *Let $\{D_1, \dots, D_k\}$, for $k \geq 2$, be the set of core disks whose 2-neighborhoods form the disjoint partition of $\mathcal{D}_{\text{in}}$. Then, $|\text{O}\small{\text{PT}}'| \geq \sum_{i=1}^{k-1} \left(\frac{r_i}{\lceil \log k \rceil}\right)$.*

Finally we have the following theorem.

**Theorem 4** *In polynomial time, we can compute a tour $T(\mathcal{D})$ visiting all the disks of $\mathcal{D}$ such that*

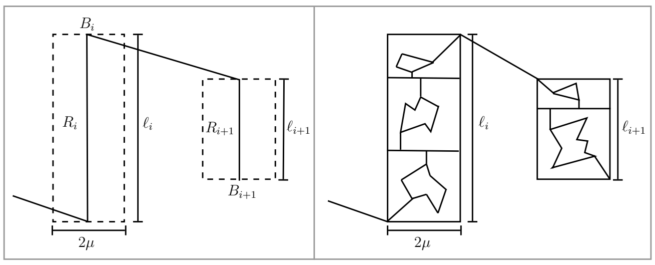


Figure 2: Two blocks $B_i$ and $B_{i+1}$, and the paths replacing them.

$\mathbb{E}[|T(\mathcal{D})|] \leq \mu + O(\log\log n\ \mathbb{E}[|\text{OPT}'|])$, *where* OPT′ *denotes the optimal tour on* $\mathcal{D}_{in}$.

## 4.2 Combining the Subtours

We now stitch together these subtours spanning the disks whose centers lie in $2\mu \times 2\mu$ size squares to construct the final tour. See Figure 2 for illustration. In particular, suppose $S_i = \{R_{i1}, R_{i2}, \ldots\}$ is the partition of the rectangle $R_i$ into these $2\mu \times 2\mu$ squares, and let $T_{ij}$ be the $O(\log\log n)$-optimal tour (obtained using Theorem 4) for the disks whose centers lie in $R_{ij}$, where $R_{ij} \in S_i$. Let $T_i$ be the path obtained by concatenating the tours $T_{ij}$, for $\{j : R_{ij} \in S_i\}$, where $i = 1, \ldots, m$, adding at most $O(|\ell_i| + \mu)$ to the length. Finally, combine the paths $T_i$, for $i = 1, \ldots, m$, to obtain a tour over $\mathcal{D}$, by connecting the boundary of $R_i$ with the boundary of $R_{i+1}$. These connections add at most $\sum_{i=1}^{m} O(B_i)$ to the tour length. (Figure 2 illustrates this construction for blocks $B_i$ and $B_{i+1}$.) It is easy to modify the resulting walk into a traveling salesman tour by the doubling and shortcutting. Let $T(\mathcal{D})$ denote the resulting tour. The following theorem establishes the main result of this section.

**Theorem 5** *In polynomial time, we can compute a tour* $T(\mathcal{D})$ *visiting the set* $\mathcal{D}$ *of stochastic disks, such that* $\mathbb{E}[|T(\mathcal{D})|] = O(\log\log n) \cdot \mathbb{E}[L^*]$.

## 5 Online Tour and Almost Disjoint Disk

We now consider the *online* version of the TSP with stochastic disks. In this case, the salesman learns the radius of each stochastic disk only on arriving at the boundary of the disk—in the data gather application, the disk radius is revealed when the robot is able to communicate with the sensor node. We propose an $O(\log n)$-approximation algorithm for the online version. In the special case, where the mean radii disks are nearly disjoint, we achieve a constant factor approximation, both for the online and the offline setting. (In practice, this is the more likely case.)

Due to lack of space, in this abstract we will only present the main results achieved by the algorithms for the online problem.

**Theorem 6** *Given a set* $\mathcal{D}$ *of random disks whose radii are revealed online, we can compute in polynomial time a tour* $T(\mathcal{D})$ *such that* $\mathbb{E}[|T(\mathcal{D})|] = O(\log n) \cdot \mathbb{E}[L^*]$.

Finally, if the disks are not "too overlapping" in the mean radius instance $M$, we can obtain a simple $O(1)$-approximate tour of $\mathcal{D}$. We say that the set $\mathcal{D}$ has depth $c$ if no point lies in the common intersection of more than $c$ disks, as they appear in $M$. We note that *even with this assumption*, a random instance of the problem may still have arbitrarily large intersection depths. Nevertheless, we can prove the following result.

**Theorem 7** *If the stochastic set* $\mathcal{D}$ *has a constant depth, then we can compute in polynomial time a tour* $T(\mathcal{D})$ *such that* $\mathbb{E}[|T(\mathcal{D})|] = O(1) \cdot \mathbb{E}[L^*]$.

## References

[1] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218, 1995.

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

[3] A. Dumitrescu and M. B. Joseph. Approximation Algorithms for TSP with Neighborhoods in the Plane. *Journal of Algorithms*, 48(1):135 – 159, 2003.

[4] K. M. Elbassioni, A. V. Fishkin, and R. Sitters. On Approximating the TSP with Intersecting Neighborhoods. In *ISAAC*, volume 4288 of *Lecture Notes in Computer Science*, pages 213–222. Springer, 2006.

[5] K. M. Elbassioni, A. V. Fishkin, and R. Sitters. Approximation algorithms for the euclidean traveling salesman problem with discrete and continuous neighborhoods. *Int. J. Comput. Geometry Appl.*, 19(2):173–193, 2009.

[6] C. S. Mata and J. S. B. Mitchell. Approximation Algorithms for Geometric Tour and Network Design Problems. In *Proceedings of the eleventh annual symposium on Computational geometry*, SoCG '95, pages 360–369. ACM, 1995.

[7] J. S. Mitchell. A constant-factor approximation algorithm for tsp with pairwise-disjoint connected neighborhoods in the plane. In *Proceedings of the 2010 annual symposium on Computational geometry*, SoCG 10, pages 183–191, 2010.

[8] J. S. B. Mitchell. A PTAS for TSP with Neighborhoods among Fat Regions in the Plane. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA 07, pages 11–18, 2007.

# Large-volume open sets in normed spaces without integral distances

Sascha Kurz* Valery Mishkin†

## Abstract

We study open sets $\mathcal{P}$ in normed spaces $X$ attaining a large volume while avoiding pairs of points at integral distance. The proposed task is to find sharp inequalities for the maximum possible $d$-dimensional volume. This problem can be viewed as an opposite to known problems on point sets with pairwise integral or rational distances.

## 1 Introduction

For quite some time it was not known whether there exist seven points in the Euclidean plane, no three on a line, no four on a circle, with pairwise integral distances. Kreisel and Kurz [4] found such a set of size 7, but it is unknown if there exists one of size 8.[1] The hunt for those point sets was initiated by Ulam in 1945 by asking for a dense point set in the plane with pairwise rational distances.

Here we study a kind of opposite problem, recently considered by the authors for Euclidean spaces, see [5]: Given a normed space $X$, what is the maximum volume $f(X,n)$ of an open set $\mathcal{P} \subseteq X$ with $n$ connected components without a pair of points at integral distance. We drop some technical assumptions for the normed spaces $X$ and mostly consider the Euclidean spaces $\mathbb{E}^d$ or $\mathbb{R}^d$ equipped with a $p$-norm. In Theorem 5 we state an explicit formula for the Euclidean case $f(\mathbb{E}^d,n)$.

## 2 Basic notation and first observations

We assume that our normed space $X$ admits a measure, which we denote by $\lambda_X$. By $\mathcal{B}_X$ we denote the open ball with diameter one in $X$, i.e. the set of points with distance smaller than $\frac{1}{2}$ from a given center. In the special case $X = (\mathbb{R}^d, \|\cdot\|_p)$ with $\|(x_1,\dots,x_d)\|_X = \|(x_1,\dots,x_d)\|_p := \left(\sum_{i=1}^d |x_i|^p\right)^{\frac{1}{p}}$, where $p \in \mathbb{R}_{>0} \cup \{\infty\}$, we have

$$\lambda_X(\mathcal{B}_X) = \Gamma\left(\frac{1}{p}+1\right)^d / \Gamma\left(\frac{d}{p}+1\right),$$

where $\Gamma$ denotes the famous Gamma function, i.e. the extension of the factorial function. In the Manhattan metric, i.e. $p=1$, the volumes of the resulting cross-polytopes equal $\frac{1}{d!}$ and in the maximum norm, i.e. $p=\infty$, the volumes of the resulting hypercubes equal 1.

At first we observe that the diameter of a connected component $\mathcal{C}$ of a set $\mathcal{P} \subseteq X$ avoiding integral distances is at most 1.[2] Otherwise we can consider two points $u, v \in \mathcal{C}$ having a distance larger than 1 and conclude the existence of a point $w \in \mathcal{C}$ on the curve connecting $u$ with $v$ in $\mathcal{C}$ such that the distance between $u$ and $v$ is exactly 1.

Given two points $u, v \in X$, where $\|v-u\|_X = 1$, we may consider the line $\mathcal{L} := \{u+\alpha(v-u) \mid \alpha \in \mathbb{R}\} \subseteq X$. The restriction of $X$ to $\mathcal{L}$ yields another, one-dimensional, normed space, where we can pose the same question.

We consider the map $\varphi : \mathcal{L} \to [0,1)$, $p \mapsto \alpha \mod 1$ where $p = u+\alpha(v-u) \in \mathcal{L}$. If $\varphi$ is not injective on $\mathcal{P} \cap \mathcal{L}$, the set $\mathcal{P}$ contains a pair of point an integral distance apart. Since the map $\varphi$ is length preserving 'modulo 1', that is, $\|p_1-p_2\|_X \mod 1 = |\varphi(p_1)-\varphi(p_2)|$, its restriction to the connected components of $\mathcal{P} \cap \mathcal{L}$ is length preserving. Thus, we conclude the necessary condition for an open set avoiding integral distances that the length of each intersection with a line is at most 1.

Having those two necessary conditions, i.e. the diameter of each connected component is at most 1 and the length of each line intersection is at most 1, at hand we define $l(X,n)$ as the maximum volume of open sets in $X$ with $n$ connected components, which satisfy the two necessary conditions. We thus have $f(X,n) \le l(X,n) \le n \cdot \lambda_X(\mathcal{B}_X)$ for all $n \in \mathbb{N}$.

In [5] the authors provided an example of a connected open set $U \subseteq \mathbb{R}^d$ such that the intersection of $U$ with each line has a total length of at most 1 but the volume of $U$ is unbounded.

Based on a simple averaging argument, any given upper bound on one of the two introduced maximum

*University of Bayreuth, D-95440 Bayreuth, Germany, sascha.kurz@uni-bayreuth.de

†York University, Toronto ON, M3J1P3 Canada, vmichkin@mathstat.yorku.ca

[1]We remark that, even earlier, Noll and Simmons, see http://www.isthe.com/chongo/tech/math/n-cluster, found those sets in 2006 with the additional property of having integral coordinates, so-called $7_2$-clusters.

[2]We remark that the maximum volume of a set with diameter 1 in $X$ is at most $\lambda_X(\mathcal{B}_X)$, see e.g. [7].

volumes for $n$ components induces an upper bound for $k \geq n$ components in the same normed space $X$:

**Lemma 1** *For each $k \geq n$ we have,*

- $l(X,k) \leq \frac{k}{n} \cdot \Lambda$ *whenever* $l(X,n) \leq \Lambda$*;*
- $f(X,k) \leq \frac{k}{n} \cdot \Lambda$ *whenever* $f(X,n) \leq \Lambda$*.*

For lower bounds we consider the following construction. Given a small constant $0 < \varepsilon < \frac{1}{n}$, we arrange one open ball of diameter $1-(n-1)\varepsilon$ and $n-1$ open balls of diameter $\varepsilon$ each, so that there centers are aligned on a line and that they are non-intersecting. Since everything fits into an open ball of diameter 1 there cannot be a pair of points at integral distance. As $\varepsilon \to 0$ the volume of the constructed set approaches $\lambda_X(\mathcal{B}_X)$, so that we have

$$\lambda_X(\mathcal{B}_X) \leq f(X,n) \leq l(X,n).$$

We then have $f(X,1) \leq l(X,1) = \lambda_X(\mathcal{B}_X)$. The map $\varphi$ shows that equality is also attained for normed spaces $X$ of dimension 1. So, in the following we consider sets consisting of at least two components and normed spaces of dimension at least two.

## 3 Two components

For one component the extremal example was the open ball of diameter 1. By choosing the line through the centers of two balls of diameter 1 we obtain a line intersection of total length two, so that this cannot happen in an integral distance avoiding set. The idea to circumvent this fact is to truncate the open balls in direction of the line connecting the centers so that both components have a width of almost $\frac{1}{2}$, see Figure 1 for the Euclidean plane $\mathbb{E}^2$.

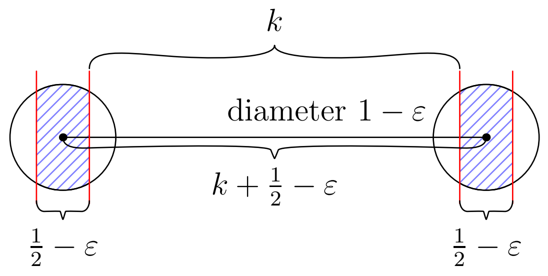

$k$
diameter $1-\varepsilon$
$k+\frac{1}{2}-\varepsilon$
$\frac{1}{2}-\varepsilon$
$\frac{1}{2}-\varepsilon$

Figure 1: Truncated disks – a construction of 2 components in $\mathbb{E}^2$ without integral distances.

The volume $V$ of a convex body $\mathcal{K} \subset \mathbb{E}^d$ with diameter $D$ and minimal width $\omega$ is bounded above by

$$V \leq \lambda_{\mathbb{E}^{d-1}}(\mathcal{B}_{\mathbb{E}^{d-1}}) \cdot D^d \int_0^{\arcsin \frac{\omega}{D}} \cos^d \theta \, \mathrm{d}\theta, \qquad (1)$$

see e.g. [3, Theorem 1]. Equality holds iff $\mathcal{K}$ is the $d$-dimensional spherical symmetric slice with diameter $D$ and minimal width $\omega$. We can easily check that the maximum volume of two $d$-dimensional spherical symmetric slices with diameter 1 each and minimal widths $\omega_1$ and $\omega_2$, respectively, so that $\omega_1 + \omega_2 \leq 1$ is attained at $\omega_1 = \omega_2 = \frac{1}{2}$, independently from the dimension.

Motivated by this fact we generally define $S_X$ to be a spherical symmetric slice with diameter 1 and width $\frac{1}{2}$, i.e. a truncated open ball. In the $d$-dimensional Euclidean case we have

$$\lambda_{\mathbb{E}^d}(S_{\mathbb{E}^d}) = \lambda_{\mathbb{E}^{d-1}}(\mathcal{B}_{\mathbb{E}^{d-1}}) \int_0^{\frac{\pi}{6}} \cos^d \theta \, \mathrm{d}\theta. \qquad (2)$$

The truncated disc in dimension $d=2$ has an area of $\frac{\sqrt{3}}{8} + \frac{\pi}{12} \approx 0.4783$.

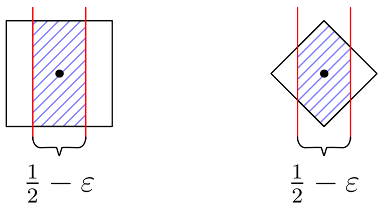


Figure 2: Spherical slices in dimension 2 for the maximum norm, $p=\infty$, and the Manhattan metric, $p=1$.

On the left hand side of Figure 2 we have drawn the spherical slice, i.e. a truncated ball, in dimension 2 for the maximum norm, i.e. $p=\infty$. For general dimension $d$ we have $\lambda_X(S_X) = \frac{1}{2}$. On the right hand side of Figure 2 we have drawn the spherical slice in dimension 2 for the Manhattan metric, i.e. $p=1$. Here we have $\lambda_X(S_X) = \frac{1}{d!} - \frac{1}{2^{d-2}(d-1)!}$.

Since the line through the upper left *corner* of the left component and the lower right *corner* of the right component should have an intersection with the shaded region of total length at most 1, we consider truncated open balls of diameter $1-\varepsilon$ and width $\frac{1}{2}-\varepsilon$, see Figure 1. For some special normed spaces $X$ we can choose $\varepsilon > 0$ and move the centers of the two components sufficiently away from each other such that we can guarantee that no line intersection has a total length of more than 1.

**Lemma 2** *For arbitrary dimension $d \geq 2$ and normed spaces $X = (\mathbb{R}^d, \|\cdot\|_p)$ with $1 < p < \infty$ we have $l(X,2) \geq f(X,2) \geq 2\lambda_X(S_X)$.*

**Proof.** For a given small $\varepsilon > 0$ place a truncated open ball of diameter $1-\varepsilon$ and width $\frac{1}{2}-\varepsilon$ with its center at the origin and a second copy so that the two centers are at distance $k+\frac{1}{2}-\varepsilon$, see Figure 1 for $p=d=2$. Since both components have diameters smaller than 1 there cannot be a pair of points at integral distance within the same component. So let $a = (a_1, \dots, a_d)$ be a point of the left and $b = (b_1, \dots, b_d)$ a point of the right component, where we assume that the centers of the components are moved apart along the first coordinate axis. By construction the distance between $a$ and $b$ is at least $k$. Since $|a_i - b_i| < 1 - \varepsilon$ for all $2 \le i \le d$ and $|a_1 - b_1| < k + 1 - 2\varepsilon$ the distance between $a$ and $b$ is less than

$$\Big((1-\varepsilon)^p \cdot (d-1) + (k+1-2\varepsilon)^p\Big)^{\frac{1}{p}}.$$

By choosing a sufficiently large integer $k$ we can guarantee that this term is at most $k+1$, so that there is no pair of points at integral distance. Finally, we consider the limit as $\varepsilon \to 0$. □

We conjecture that the lower bound from Lemma 2 is sharp for all $p > 1$ and remark that this is true for the Euclidean case $p = 2$ by Inequality (1).

## 4 Relation to finite point sets with pairwise odd integral distances

In this section we restrict the connected components to open balls of diameter $\frac{1}{2}$. We remark that if an integral distance avoiding set contains an open ball of diameter $0 < D < 1$, which fits into one of its components, then the other components can contain open balls of diameter at most $1 - D$. One can easily check that the maximum volume of the entire set $\mathcal{P}$ is attained at $D = \frac{1}{2}$, at least for $p$-norms and dimensions $d \ge 2$. If the set $\mathcal{P}$, i.e. the collection of $n$ open balls of diameter $\frac{1}{2}$, does not contain a pair of points at integral distance, then the mutual distances between centers of different balls have to be elements of $\mathbb{Z} + \frac{1}{2}$. Therefore dilating $\mathcal{P}$ by a factor of 2 yields the set $\mathcal{Q}$ of the centers of the $n$ balls with pairwise odd integral distances.

However, for Euclidean spaces $\mathbb{E}^d$, it is known, see [2], that $|\mathcal{Q}| \le d + 2$, where equality is possible if and only if $d \equiv 2 \pmod{16}$. It would be interesting to determine the maximum number of odd integral distances in other normed spaces.

## 5 Large-volume open sets with diameter and maximum length of line intersections at most one

Assuming that the construction using truncated open balls from Section 3 is best possible or, at the very least, competitive, we can try to arrange $n$ copies of those $S_X$. Since we have to control that each line meets at most two components we cannot arrange the centers on a certain line. On the other hand, the cutting directions, i.e. the directions where we cut of the caps from the open balls, should be almost equal. To meet both requirements, we arrange the centers of the components on a parabola, where each component has diameter $1-\varepsilon$ and width $\frac{1}{2}-\varepsilon$, for a small constant $\varepsilon > 0$, see Figure 3 for an example in $\mathbb{E}^2$.

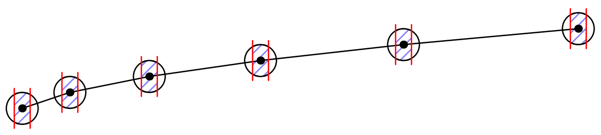


Figure 3: Truncated discs – arranged on a parabola.

**Lemma 3** *For arbitrary dimension $d \ge 2$, $n \ge 2$, and normed spaces $X = (\mathbb{R}^d, \|\cdot\|_p)$ with $p > 1$ we have $l(X, n) \ge n\lambda_X(S_X)$.*

**Proof.** For a given small $\varepsilon > 0$ consider $n$ truncated $d$-dimensional balls $S_X$ of width $\frac{1}{2} - \varepsilon$, where the truncation is oriented in the direction of the $y$-axis, with centers located at $\big(i \cdot k, i \cdot k^2, 0, \dots, 0\big)$ and diameter $1 - \varepsilon$ for $1 \le i \le n$, see Figure 3. For $k$ large, there is no line intersecting three or more components. It remains to check that each line meeting two $d$-dimensional balls centered at $C_1$, $C_2$ has an intersection of length at most $\frac{1}{2}$ with each of the truncated balls. This can be done by performing a similar calculation as in the proof of Lemma 2. Again, we consider the limiting configuration as $\varepsilon \to 0$. □

We conjecture that the lower bound in Lemma 3 is sharp.

## 6 Using results from Diophantine Approximation

In order to modify the construction from the previous section to obtain a lower bound for $f(X, n)$, we use results from Diophantine Approximation and:

**Lemma 4** *Given an odd prime $p$, let $\alpha_j = \frac{\zeta_j - \zeta_{2p-j}}{i}$ for $1 \le j \le \frac{p-1}{2}$, where the $\zeta_j$ are 2pth roots of unity. Then the $\alpha_j$ are irrational and linearly independent over $\mathbb{Q}$.*

A proof, based on a theorem of vanishing sums of roots of unity by Mann [6], is given in [5].

This result can be applied to construct sets avoiding integral distances as follows. We fix an odd prime $p$ with $p \ge n$. For each integer $k \ge 2$ and each $\frac{1}{4} > \varepsilon > 0$ we consider a regular $p$-gon $P$ with side lengths $2k \cdot \sin\!\left(\frac{\pi}{p}\right)$, i.e. with circumradius $k$. At $n$ arbitrarily chosen vertices of the $p$-gon $P$ we place the centers of $d$-dimensional open balls with diameter $1 - \varepsilon$. Since the diameter of each of the $n$ components is less than 1 there is no pair of points at integral distance inside one of these $n$ components. For each pair of centers $c_1$ and $c_2$, we cut off the corresponding two components such that each component has a width of $\frac{1}{2} - \varepsilon$ in that direction, see Figure 4 for an example with $n = p = 5$.

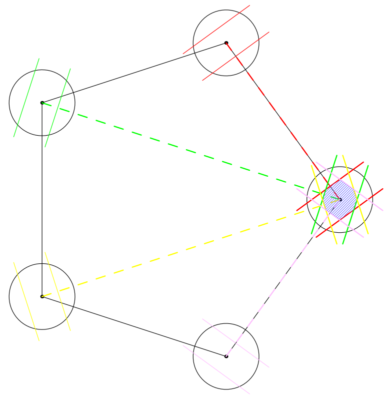


Figure 4: $p$-gon construction: Integral distance avoiding point set for $d = 2$ and $p = n = 5$.

Next we consider two points $a$ and $b$ from different components. We denote by $\alpha$ the distance of the centers of the corresponding components. From the triangle inequality we conclude

$$\alpha - \left(\frac{1-2\varepsilon}{2}\right) < \text{dist}(a,b) < \alpha + \left(\frac{1-2\varepsilon}{2}\right).$$

Since the occurring distances $\alpha$ are given by $2k\sin\left(\frac{j\pi}{p}\right)$ for $1 \le j \le \frac{p-1}{2}$ we look for a solution of the following system of inequalities

$$\left\{2k \cdot \sin\left(\frac{j\pi}{p}\right) - \frac{1}{2} + \varepsilon\right\} \le 2\varepsilon \qquad (3)$$

with $k \in \mathbb{N}$, where $\{\beta\}$ denotes the positive fractional part of a real number $\beta$, i.e. there exists an integer $l$ with $\beta = l + \{\beta\}$ and $0 \le \{\beta\} < 1$. By Lemma 4 the factors $2\sin\left(\frac{j\pi}{p}\right)$ are irrational and linearly independent over $\mathbb{Q}$, so by Weyl's Theorem [8] the systems admit a solution for all $k$. (Actually we only use the denseness result, which Weyl himself attributed to Kronecker.) We call the just described construction the $p$-gon construction.

These ingredients we provided in more detail in [5] enable us to establish the conjectured exact values of the function $f\Big((\mathbb{E}^d, \|\cdot\|_2), n\Big)$:

**Theorem 5** *For all $n, d \ge 2$ we have*

$$f\Big(\mathbb{E}^d, n\Big) = n \cdot \lambda_{\mathbb{E}^d}(S_{\mathbb{E}^d}).$$

**Proof.** (Sketch)
For a given number $n$ of components we choose an increasing sequence of primes $n < p_1 < p_2 < \dots$. For each $i = 1, 2, \dots$ we consider the $p$-gon construction with $p = p_i$ and $n$ neighbored vertices. If the scaling factor $k$ tends to infinity the $d$-dimensional volume of the resulting sequence of integral distance avoiding sets tends to the volume of the following set: Let $\mathcal{P}_i$ arose as follows. Place an open ball of diameter 1 at $n$ neighbored vertices of a regular $p_i$-gon with radius $2n^2$. Cut off caps in the direction of the lines connecting each pair of centers so that the components have a width of $\frac{1}{2}$ in that direction. In order to estimate the volume of $\mathcal{P}_i$ we consider another set $\mathcal{T}_i$, which arises as follows. Place an open ball of diameter $1 - \Delta_i$ at $n$ neighbored vertices of a regular $p_i$-gon with radius $2n^2$. Cut off caps in the direction of the $x$-axis so that the components have a width of $\frac{1}{2} - \Delta_i$. One can suitably choose $\Delta_i$ so that $\mathcal{T}_i$ is contained in $\mathcal{P}_i$. Since the centers of the components of $\mathcal{P}_i$ tend to be aligned, as $i$ increases, $\Delta_i$ tends to 0 as $i$ approaches infinity. We thus have $\lim_{i\to\infty} \lambda_{\mathbb{E}^d}(\mathcal{T}_i) = n \cdot \lambda_{\mathbb{E}^d}(S_{\mathbb{E}^d})$. $\square$

## 7 Conclusion

We have proposed the question for the maximum volume of an open set $\mathcal{P}$ consisting of $n$ components in an arbitrary normed space $X$ avoiding integral distances. For the Euclidean plane those sets need to have upper density 0, see [1]. Theorem 5 proves a conjecture stated in [5] and some of the concepts have been transferred to more general spaces. Nevertheless, many problems remain unsolved and provoke further research.

## References

[1] H. Furstenberg, Y. Katznelson, and B. Weiss, *Ergodic theory and configurations in sets of positive density*, Mathematics of Ramsey theory, Coll. Pap. Symp. Graph Theory, Prague/Czech., Algorithms Comb. 5, 184–198, 1990.

[2] R.L. Graham, B.L. Rothschild, and E.G. Straus, *Are there $n+2$ points in $E^n$ with odd integral distances?*, Amer. Math. Mon. **81** (1974), 21–25.

[3] M.A. Hernández Cifre, G. Salinas, and S. Segura Gomis, *Two optimization problems for convex bodies in the n-dimensional space*, Beiträge Algebra Geom. **45** (2004), no. 2, 549–555.

[4] T. Kreisel and S. Kurz, *There are integral heptagons, no three points on a line, no four on a circle*, Discrete Comput. Geom. **39** (2008), no. 4, 786–790.

[5] S. Kurz and V. Mishkin, *Open sets avoiding integral distances*, (submitted), http://arxiv.org/abs/1204.0403.

[6] H.B. Mann, *On linear relations between roots of unity*, Mathematika, Lond. **12** (1965), 107–117.

[7] M.S. Mel'nikov, *Dependence of volume and diameter of sets in an n-dimensional banach space*, Uspekhi Mat. Nauk **18** (1963), no. 4(112), 165–170.

[8] H. Weyl, *Über die Gleichverteilung von Zahlen mod. Eins*, Math. Ann. **77** (1916), 313–352.

# Clear Unit-Distance Graphs[*]

Marc van Kreveld[§] Maarten Löffler[§] Frank Staals[§]

## Abstract

We introduce a variation of unit-distance graphs which we call *clear unit-distance graphs*. They require the pairwise distances of the representing points to be either exactly 1 or not close to 1. We discuss properties and applications of clear unit-distance graphs.

## 1 Introduction

Unit-distance graphs are embedded graphs, usually in $\mathbb{R}^2$, with the property that there is an edge between two vertices if and only if their distance is exactly 1. Unit-distance graphs can be traced back to Erdös [3], who asked the now-famous open question of how many edges a unit-distance graph with $n$ vertices can have. Since then, many deep observations about the class of unit-distance graphs have been made [1, 2, 5].

Unit-distance graphs appear in many applications, but we focus here on the role they play in certain games and puzzles. Figure 1 shows a maze puzzle realized by a set of a holes in a metal plate, and a movable ring that has a small part of the perimeter missing. The missing part allows the ring to navigate from hole to hole, but only between holes at unit distance.

[*]M. Löffler and F. Staals were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.021.123 and 612.001.022, respectively.

[§]Department of Information and Computing Sciences, Universiteit Utrecht, The Netherlands, {m.j.vankreveld|m.loffler|f.staals}@uu.nl

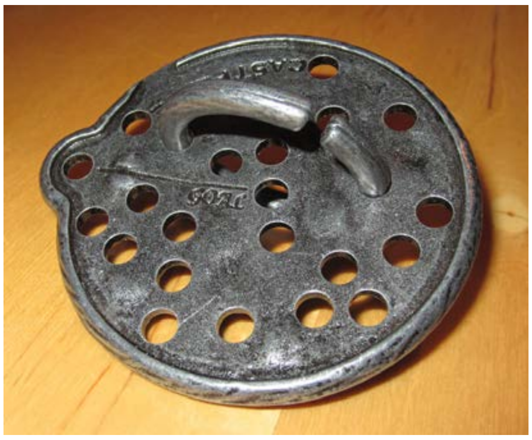


Figure 1: A maze represented as the graph of a clear unit distance point set. The goal is to get the ring through the two holes connected by a line segment (the text reads "GOAL").

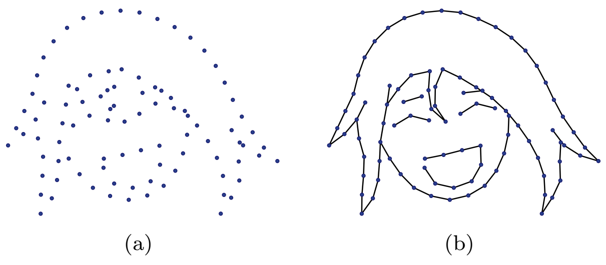


Figure 2: (a) A set of points, forming a unite-the-dots puzzle. (b) The drawing induced by the points as a clear unit distance graph, with $\varepsilon = 0.1$.

As another example, we propose a new type of drawing puzzle. The puzzle *connect-the-dots* is a well-known activity for young children, where the goal is to create a drawing by connecting a series of numbered dots in the given order. The resulting curve usually completes a simple picture. However, connecting the dots by means of numbered dots suffers from some profound disadvantages. The puzzle only allows for a single polyline, which decreases the possible complexity of the drawing. Usually this is circumvented by pre-drawing some parts of the drawing. In addition, the printed numbers do not improve the visual appearance of the completed drawing. Finally, the use of numbers is not particularly challenging. We therefore propose an adapted version of this puzzle which is both more challenging and allows not just polyline drawings, without the use of numbering. In this puzzle, which we call *unite-the-dots*, two dots should be connected by a line segment if the distance between these two points is equal to a pre-defined distance. Without loss of generality we assume this distance to be 1, hence the drawing resulting from a unite-the-dots puzzle corresponds to a unit-distance graph. An example is shown in Figure 2.

**Clear unit-distance graphs.** In practice, it is hard to distinguish points at unit distance, and points at almost unit distance. This motivates studying a new class of graphs, the *clear unit-distance graphs*, in which points are either at unit distance, or a distance significantly different from 1. Clear unit-distance graphs can be seen as a variation on the unit-distance graph that is useful in practical situations like the ones just described. We will model the minimum required deviation from the unit distance by an additive parameter $\varepsilon$. Moreover, in the applications mentioned above, it is also undesirable if pairs of points are too close. For unite-the-dots, the disk-shaped drawings of points must be disjoint, and for the mechanical mazes the holes should not merge into bigger holes. For simplicity we use the same parameter $\varepsilon$ to specify how far two distinct points should be apart. Hence, for a given value $\varepsilon$, the allowed distances between points lie in the range $[\varepsilon, 1-\varepsilon] \cup [1] \cup [1+\varepsilon, \infty)$.

**Results and Organization.** Section 2 formally introduces clear unit-distance graphs and Section 3 investigates several properties of these graphs. Section 4 describes the unite-the-dots problem in more detail and briefly discusses how these properties may be used to obtain efficient algorithms for automatically generating unite-the-dots puzzles.

## 2 Definitions

An *$\varepsilon$-distinguishable unit-distance point set* is a set of points in the plane with the property that every pair $(p, q)$ of points has a distance $d(p, q)$ that is either exactly $1$, or at least $\varepsilon$ and at most $1-\varepsilon$, or at least $1+\varepsilon$.[1] An $\varepsilon$-distinguishable unit-distance point set induces an *$\varepsilon$-distinguishable unit-distance drawing*, by connecting all points at unit distance with an edge. A graph $G$ is an *$\varepsilon$-distinguishable unit-distance graph* if it has an *$\varepsilon$-distinguishable unit-distance drawing* whose points correspond to the vertices of $G$. For brevity we will write $(1, \varepsilon)$-point set, $(1, \varepsilon)$-graph, and $(1, \varepsilon)$-drawing.

A *clear unit-distance graph* is an $\varepsilon$-distinguishable unit-distance graph for some constant $\varepsilon > 0$.

## 3 Properties of clear unit-distance graphs

In this section, we investigate some properties of $(1, \varepsilon)$-point sets and graphs. We assume $0 < \varepsilon < 1$ is a given constant, and write $\gamma = 1/\varepsilon$.

### 3.1 Density bounds

**Observation 1** *Let $G$ be an $(1, \varepsilon)$-graph, and $R \subset \mathbb{R}^2$ a region of constant diameter. There can be at most $O(\gamma^2)$ vertices of any $(1, \varepsilon)$-drawing of $G$ in $R$.*

[1]While the extra condition that inter-point distance must be at least $\varepsilon$ is natural from the point of view of the applications, the conceptually cleaner definition which only requires distances to be either $1$ or at least $\varepsilon$ different from $1$ may also be of theoretical interest. We argue that the extra condition does not influence the results too much, since any two points at distance less than $\varepsilon$ must have exactly the same neighbors in the (relaxed) $(1, \varepsilon)$-graph. We defer a more thorough discussion of this issue to the full paper.

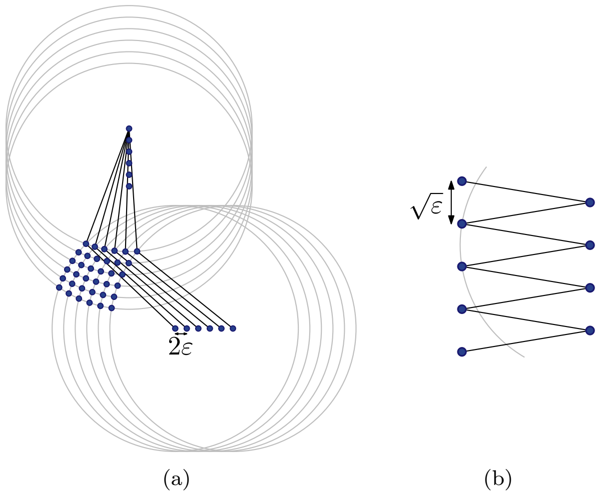


Figure 3: (a) A connected graph may have $\Omega(\gamma^2)$ vertices in a constant-diameter area. (Not all edges are shown to avoid visual clutter.) (b) A zig-zag path must have distance $\sqrt{\varepsilon}$ between points to ensure a distance of at least $\varepsilon$ from a point to the unit circle centered at the points on the other side.

This upper bound follows directly from the requirement that inter-point distances are at least $\varepsilon$. More interestingly, the bound can actually be achieved, even if the graph is required to be connected:

**Lemma 2** *There exist a connected $(1, \varepsilon)$-graph $G$, a $(1, \varepsilon)$-drawing $D$ of $G$ and a region of constant diameter $R$, such that there are $\Omega(\gamma^2)$ vertices of $D$ in $R$.*

**Proof.** Note that placing points on the vertices of a regular $\varepsilon$-spaced $\gamma/4$ by $\gamma/4$ grid results in a valid $(1, \varepsilon)$-drawing, since all inter-point distances are at least $\varepsilon$ and clearly smaller than $1-\varepsilon$. To make the graph connected, we slightly modify the grid and place the points on the intersections of two sets of $\gamma/4$ circles, whose centers lie $2\varepsilon$-spaced on a horizontal and a vertical line, see Figure 3(a). $\square$

The construction above relies on high-degree vertices to work. More interesting is the question of how dense a connected $(1, \varepsilon)$-graph with maximum vertex degree $d$ can be. Already for the case $d = 2$ (i.e., for paths) this appears to be a challenging question. We provide a lower bound of $\Omega(\sqrt{\gamma})$:

**Lemma 3** *There exist a connected $(1, \varepsilon)$-graph $G$ of maximum degree $2$, a $(1, \varepsilon)$-drawing $D$ of $G$ and a region of constant diameter $R$, such that there are $\Omega(\sqrt{\gamma})$ vertices of $D$ in $R$.*

**Proof.** We take two sets of $\sqrt{\gamma}/2$ points, each lying $\sqrt{\varepsilon}$-spaced on a vertical line, such that the first two points on the first line both lie at distance $1$ to the first point on the second line. Then the induced

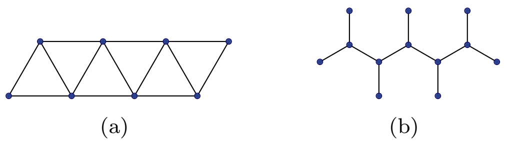


Figure 4: (a) A fixed triangle strip forces a large diameter. (b) For $\varepsilon = \sqrt{3}-1$, a claw graph has a fixed drawing, which can be replicated to get a linear diameter.

unit-distance graph is a path. Furthermore, the distance between any pair of points not at distance $1$ is at distance more than $1 + \varepsilon$ if they lie on different lines or less than $1/2$ if they lie on the same line, see Figure 3(b). □

### 3.2 Crossing number bounds

The number of crossings in the example of Figure 3(a) is $\Theta(\gamma^4)$, showing that a constant-diameter region can have as many crossings.

**Lemma 4** *Any $(1, \varepsilon)$-drawing has $O(\gamma^{16/3})$ crossings in a constant-diameter region.*

**Proof.** Only $O(\gamma^2)$ points can be within unit distance of a constant-diameter region because these points must lie in a (slightly larger) constant-diameter region themselves. These points realize at most $O(\gamma^{8/3})$ unit distances which are the edges (using the upper bound for the unit-distance problem by Spencer, Szemerédi and Trotter [4]). □

### 3.3 Diameter bounds

We now proceed to bound the (geometric) diameter of $(1, \varepsilon)$-drawings. Clearly, since all connected point pairs are at unit distance, no drawing of a connected $(1, \varepsilon)$-graph can have diameter larger than $n$. We show that $(1, \varepsilon)$-graphs exist whose drawings necessarily need this diameter:

**Lemma 5** *A connected $(1, \varepsilon)$-graph $G$ of maximum degree $4$ exists such that any $(1, \varepsilon)$-drawing of $G$ has diameter $\Omega(n)$, for any $0 < \varepsilon \leq \sqrt{3} - 1$.*

**Proof.** A rigid strip of triangles is a clear unit distance graph. See Figure 4(a). □

For some specific values of $\varepsilon$, we can show that even trees may require a linear diameter:

**Lemma 6** *A $(1, \varepsilon)$-tree $G$ of maximum degree $3$ exists such that any $(1, \varepsilon)$-drawing of $G$ has diameter $\Omega(n)$, for $\varepsilon = \sqrt{3} - 1$.*

**Proof.** Let $G$ be a caterpillar graph where all internal nodes have degree $3$ (i.e., a path turned into a tree by adding a leaf to every internal path node). Now, the three incident edges of any internal node must make $120°$ angles with each other; otherwise, two of them would be at a distance less than $\sqrt{3} = 1 + \varepsilon$ to each other (note that they are not connected in $G$ so they cannot be at distance $1$, and they also cannot be at a distance bigger than $\varepsilon$ and smaller than $1 - \varepsilon$ since $\varepsilon > 1/2$). It remains to argue that the embedding has a purely zigzagging backbone. But this is obvious, since any deviation would place six points in regular hexagonal position, creating a cycle. □

Clearly, as a direct consequence of Observation 1, a $(1, \varepsilon)$-drawing must have diameter at least $\Omega(\sqrt{n\varepsilon})$. Conversely, we show that there are graphs for which every drawing has this diameter:

**Lemma 7** *A connected $(1, \varepsilon)$-graph $G$ exists such that any $(1, \varepsilon)$-drawing of $G$ has diameter $O(\sqrt{n\varepsilon})$.*

**Proof.** We construct a graph $G$ consisting of $O(\varepsilon^2 n)$ copies of the construction in Figure 3(a), linked to a $c\sqrt{n\varepsilon}$ by $c\sqrt{n\varepsilon}$ grid. Clearly, the grid ensures that any drawing has diameter $O(\sqrt{n\varepsilon})$, and by choosing $c$ sufficiently large we make sure that if the grid is drawn regularly, there is enough room in its faces for the $O(\gamma^2)$ points without interfering with the grid points themselves. □

## 4 Unite-the-dots

Unite-the-dots puzzles are a variation of connect-the-dots (a.k.a. follow-the-dots), where a set of numbered points is given, and a polyline must be drawn that connects them in the right order. Unite-the-dots does not use numbers to annotate points. Instead, two points are connected if and only if they are at exactly the right (unit) distance. Unit-distance drawings are the output for a given set of points, and clear unit-distance point sets are suitable as the input for unite-the-dots puzzles. The puzzle can be solved with the help of a small coin or short stick.

In this section we study the problem of converting a line drawing into a clear unit-distance point set whose clear unit-distance drawing resembles the line drawing. Let $C$ be a curve between two points $p, q$ at unit distance. We say that the line segment $\overline{pq}$ *u-models* $C$ with respect to a parameter $\delta \geq 0$ if the length of $C$ is at most $1 + \delta$, and $C$ is fully inside the intersection of the radius-$1$ disks centered at $p$ and $q$. By extension, we also say that the points $p, q$ u-model $C$.

More generally, let $C$ be any curve. Denote the subcurve between any two points $p, q \in C$ by $C(p, q)$. Let $p_1, \ldots, p_k$ be a set of $k$ points on $C$ and ordered along it. Then we say that $p_1, \ldots, p_k$ u-model $C$ if (i) $p_1$ and $p_k$ coincide with the two endpoints of $C$,

(ii) points $p_i, p_{i+1}$ are at unit distance for all $1 \leq i \leq k-1$, (iii) points $p_i, p_{i+1}$ u-model $C(p_i, p_{i+1})$ for all $1 \leq i \leq k-1$, and (iv) no other pair of points is at unit distance. To be suitable as a $(1, \varepsilon)$-point set, we need to strengthen the last condition: (iv) every other pair of points is at distance $\geq \varepsilon$ and $\leq 1-\varepsilon$ or $\geq 1+\varepsilon$.

Even more generally, let $\mathcal{C} = \{C_1, \ldots, C_h\}$ be a collection of curves. A $(1, \varepsilon)$-set of points $P$ u-models $\mathcal{C}$ if and only if $P$ is a $(1, \varepsilon)$-point set, for every curve, a subset of the points in $P$ u-models it, and no pair of points of $P$ lies at distance $1$ unless they u-model a piece of some curve in $\mathcal{C}$. Intuitively, this means that the corresponding $(1, \varepsilon)$-drawing resembles $\mathcal{C}$. Furthermore, we require that $P$ be minimal: no subset of $P$ should also u-model $\mathcal{C}$. This condition ensures that $P$ has no isolated points in its $(1, \varepsilon)$-drawing.

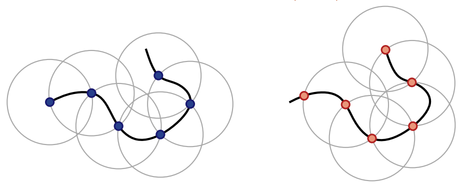


Figure 5: A curve with points chosen from the one end or the other end.

Most curves, even most line segments, are not u-modeled by any point set. Since we must choose one point of $P$ at the endpoint of the curve, and the next point must be at unit distance, it would be a coincidence if a point (the last point) coincides with the other endpoint. For example, only integer-length line segments can be u-modeled. To overcome this caveat we will allow one piece of each curve to be not u-modeled. This may be an end piece or an interior piece, but we would like it to be short. One could for instance start at one end of $C$, choose a point in $P$, and then incrementally choose the next point where the curve leaves the unit disk around the previously chosen point, until the remaining part of $C$ lies fully inside the last disk, see Figure 5. We would then have to check whether the chosen points $p_1, \ldots, p_k$ u-model $C(p_1, p_k)$ and form a $(1, \varepsilon)$-point set. Similarly, we could start at the other end, or start at both ends and leave a part in the middle.

Given a drawing, represented by a set of curves, we wish to compute the dots that make a unite-the-dots puzzle, along with any curve pieces that are not u-modeled by the dots. We observe by the u-modeling definition with parameter $\delta$ that the number of points in $P$ is always linear in the total length of the curves in the input.

Suppose we are given a collection $\mathcal{C}$ of $h$ curves and parameters $\varepsilon$ and $\delta$, we can decide whether a $(1, \varepsilon)$-point set exists whose $(1, \varepsilon)$-drawing resembles $\mathcal{C}$ with the exception of at most one short ending piece per curve (which would be pre-drawn). This is done as follows. For each curve $C_i$, generate the two sets of points $P_i$ and $Q_i$ as in Figure 5. They are associated with the TRUE and FALSE states of a variable $x_i$. By examining $P_i$ and $Q_i$ separately we can decide if $x_i$ can be TRUE or FALSE at all. By taking pairs of points of different curves, say a point of $Q_i$ and a point of $P_j$, we test their distance to decide if they can be together in a $(1, \varepsilon)$-point set. If not, we make a clause $(x_i \vee \overline{x}_j)$. This approach allows us to solve the problem using 2-SAT in $O(n^2)$ time, where $n$ is the total length of all curves (and also the number of points in a unite-the-dots puzzle, if one exists).

Using packing and algorithmic ideas we can improve the bound to $O(n \log n)$. The dependency on $\varepsilon$ is quartic, which can be derived from our results in Section 3. We can use the same approach when we allow to pre-draw at most one short interior curve piece. Minimizing the total length of the pieces that must be pre-drawn is NP-hard. We show these results in the full paper.

## 5 Conclusions

We introduced clear unit-distance graphs and drawings as a way to model situations where it is important to clearly distinguish unit-distance point pairs from other point pairs. We made several observations about the properties of clear unit-distance graphs. We expect that many of our bounds can be improved. It would be of particular interest to improve the density upper bound for paths: there is a substantial gap between the $\Omega(\sqrt{\gamma})$ lower bound and the $O(\gamma^2)$ upper bound, and an improved upper bound would immediately imply a better $\varepsilon$-dependency for our unite-the-dots algorithm.

**Acknowledgment.** The authors thank Martin Kodde for inspiring discussions in the early stages of this work.

## References

[1] S. Cabello, E. D. Demaine, and G. Rote. Planar embeddings of graphs with specified edge lengths. *J. Graph Algorithms Appl.*, 11(1):259–276, 2007.

[2] K. B. Chilakamarri. Unit distance graphs in rational $n$-space. *Discr. Math.*, 69:213–218, 1988.

[3] P. Erdös. On sets of distances of $n$ points. *The American Mathematical Monthly*, 53(5):248–250, 1946.

[4] J. Spencer, E. Szemerédi, and W. Trotter. Unit distances in the Euclidean plane. In B. Bollobas, editor, *Graph Theory and Combinatorics (Proc. Cambridge Conf. on Combinatorics)*, pages 293–308. Academic Press, 1984.

[5] A. Žitnik, B. Horvat, and T. Pisanski. All generalized Petersen graphs are unit-distance graphs, 2010. IMFM preprints 1109.

# Extending partial representations of proper and unit interval graphs

Pavel Klavík[*][‖] Jan Kratochvíl[*][‖] Yota Otachi[†] Ignaz Rutter[‡] Toshiki Saitoh[§]
Maria Saumell[¶][‖][**] Tomáš Vyskočil[*][‖]

## Abstract

A recently introduced problem of extending partial interval representations asks, for an interval graph with some intervals pre-drawn by the input, whether it is possible to extend this partial representation to a representation of the entire graph. In this paper, we give a linear-time algorithm for extending proper interval representations and an almost quadratic-time algorithm for extending unit interval representations.

## 1 Introduction

Geometric intersection graphs, and in particular intersection graphs of objects in the plane, have gained a lot of interest for their practical motivations, algorithmic applications, and interesting theoretical properties. Undoubtedly the oldest and the most studied among them are *interval graphs*, i.e., intersection graphs of intervals on the real line. They were introduced by Hájos [7] in the 1950's and the first polynomial-time recognition algorithm appeared already in the early 1960's [6]. For overviews of interval graphs and other intersection-defined classes, see [13].

Only recently, the following very natural generalization of the recognition problem has been considered [12]. The input of the partial representation extension problem consists of a graph and a part of the representation and it asks whether it is possible to extend this partial representation to the entire graph. Klavík et al. [12] give a quadratic-time algorithm for the class of interval graphs and a cubic-time algorithm for the class of proper interval graphs. Polynomial-time algorithms are given for interval graphs [1, 11], and for function and permutation graphs [8]. Most of

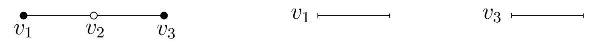


Figure 1: Partial representation that can be extended to a proper representation, but not to a unit one.

the cases of chordal graphs as subtrees-in-tree intersection graphs are hard to extend [10].

In this paper, we give a linear-time algorithm for proper interval graphs and an almost quadratic-time algorithm for unit interval graphs.

**Problem Description.** For a graph $G$, an *interval representation* $\mathcal{R}$ is a collection of intervals $\{R_u : u \in V(G)\}$ such that $R_u \cap R_v \neq \emptyset$ if and only if $uv \in E(G)$. For an interval $R_v$, we denote its left and right endpoint by $\ell_v$ and $r_v$, respectively. For numbered vertices $v_1, \dots, v_n$, we denote these endpoints by $\ell_i$ and $r_i$. A graph is an *interval graph* if it has an interval representation.

We consider two subclasses of interval graphs. An interval representation is called *proper* if no interval is a proper subset of another interval. An interval representation is called *unit* if the length of each interval is one. The class of *proper interval graphs* (PROPER INT) consists of all interval graphs having proper interval representations, whereas the class of *unit interval graphs* (UNIT INT) consists of all interval graphs having unit interval representations.

The *recognition* problem of a class $\mathcal{C}$ asks whether an input graph belongs to $\mathcal{C}$; that is, whether it has a representation by the specific type of sets $R_u$. A *partial representation* $\mathcal{R}'$ of $G$ is a representation of an induced subgraph $G'$ of $G$. A vertex in $V(G')$ is called *pre-drawn*. A representation $\mathcal{R}$ *extends* $\mathcal{R}'$ if $R_u = R'_u$ for each $u \in V(G')$. Here we deal with the problem Partial Representation Extension of $\mathcal{C}$, RepExt($\mathcal{C}$) for short. The input of the problem is a graph $G$ with a partial representation $\mathcal{R}'$, and the problem asks whether $G$ has a representation $\mathcal{R}$ that extends $\mathcal{R}'$.

**Our results.** It is well-known that PROPER INT = UNIT INT. However, partial representation extension behaves differently for these two classes (see Figure 1 for an example). In [12], the RepExt(PROPER INT) problem is solved in time $\mathcal{O}(nm)$, where $n$ is the number of vertices and $m$ the number of edges of the input graph. We show:

[*]Department of Applied Mathematics, Charles University, {klavik,honza,whisky}@kam.mff.cuni.cz

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

[‡]Faculty of Informatics, Karlsruhe Institute of Technology, rutter@kit.edu

[§]Graduate School of Engineering, Kobe University, saitoh@eedept.kobe-u.ac.jp

[¶]Département d'Informatique, Université Libre de Bruxelles, maria.saumell.m@gmail.com

[‖]Supported by ESF EuroGIGA project GraDR as GAČR GIG/11/E023.

[**]Supported by ESF EuroGIGA project ComPoSe as F.R.S.-FNRS - EUROGIGA NR 13604.

**Theorem 1** *The* RepExt(PROPER INT) *problem can be solved in time* $\mathcal{O}(n+m)$.

As for RepExt(UNIT INT), we deal with a more general problem called *bounded representation* of unit interval graphs, BoundRep for short. An input of BoundRep gives a graph $G$ and $b$ constraints of the form $\ell_i \geq \text{lbound}(v_i)$, and $\ell_i \leq \text{ubound}(v_i)$. The problem asks whether $G$ has a unit interval representation respecting the bounds. In general, this problem is NP-complete; see the full version [9] of this paper.

In each representation of a graph with $c$ components, these components are ordered from left to right. Let us denote this ordering $\blacktriangleleft$. On the other hand, let $D(r)$ denote the complexity of dividing numbers of length $r$ in binary—the best known algorithm achieves $D(r) = \mathcal{O}(r \log r 2^{\log^* r})$ [5].

**Theorem 2** *The* BoundRep *problem with a prescribed ordering* $\blacktriangleleft$ *can be solved in time* $\mathcal{O}(n^2 + nD(r))$*, where* $r$ *is the size of the input describing bound constraints.*

We obtain the following corollary:

**Corollary 3** *The* RepExt(UNIT INT) *problem can be solved in time* $\mathcal{O}(n^2+nD(r))$*, where* $r$ *is the size of the input describing positions of pre-drawn intervals.*

Our algorithms also construct the required representations. Proofs and details that have been omitted in this version can be found in [9].

## 2 Preliminaries and RepExt(PROPER INT)

We reserve $n$ for the number of vertices and $m$ for the number of edges of $G$. We denote the sets of vertices and edges by $V(G)$ and $E(G)$ respectively. For a vertex $v$, we define $N[v] = \{x, vx \in E(G)\} \cup \{v\}$. Here we assume that $G$ contains no two vertices $u$ and $v$ such that $N[u] = N[v]$. See [9] for the case where such vertices appear in $G$.

**Unique Ordering.** It is well-known that in each proper interval representation, intervals are uniquely ordered from left to right. This ordering $<$ is the order of the left endpoints and at the same time the order of the right endpoints.

**Lemma 4 (from [4])** *For a connected proper/unit interval graph, the left-to-right ordering* $<$ *is uniquely determined up to complete reversal.*

Such an ordering can be computed in linear time [3]. Using this unique ordering, extendible instances of RepExt(PROPER INT) can be simply characterized, which yields the linear-time algorithm of Theorem 1; see [9] for details.

**Representations in $\varepsilon$-grids.** Given $\varepsilon = \frac{1}{K}$, where $K$ is an integer, the $\varepsilon$-grid is the set of points $\{k\varepsilon : k \in \mathbb{Z}\}$. For a given instance of BoundRep, we ask which value of $\varepsilon$ ensures that we can construct a representation having all endpoints on the $\varepsilon$-grid. For the standard unit interval graph representation problem a grid of size $\frac{1}{n}$ is sufficient [3]. In the case of BoundRep, consider all values $\text{lbound}(v_i)$ and $\text{ubound}(v_i)$ distinct from $-\infty, +\infty$, and express them as irreducible fractions $\frac{p_1}{q_1}, \frac{p_2}{q_2}, \cdots, \frac{p_b}{q_b}$. Then we define:

$$\varepsilon' := \frac{1}{\text{lcm}(q_1, q_2, \dots, q_b)}, \qquad \text{and} \qquad \varepsilon := \frac{\varepsilon'}{n}. \quad (1)$$

**Lemma 5** *If there exists a valid representation* $\mathcal{R}'$ *for an input of the problem* BoundRep*, there exists a valid representation* $\mathcal{R}$ *in which all intervals have endpoints on the* $\varepsilon$*-grid, where* $\varepsilon$ *is defined by (1).*

## 3 LP Algorithm for BoundRep with Prescribed $\blacktriangleleft$

We process components $C_1 \blacktriangleleft C_2 \blacktriangleleft \cdots \blacktriangleleft C_c$ from left to right. For each component $C_t$, we calculate the ordering $<$ of Lemma 4 and its reversal.

When processing $C_t$, we solve one linear program for each of the two orderings. Let $v_1 < \cdots < v_k$ be one such ordering. We denote the right-most endpoint of a representation of $C_t$ by $E_t$ (we also define $E_0 = -\infty$). Additionally, we redefine all lower bounds as $\text{lbound}(v_i) = \max\{\text{lbound}(v_i), E_{t-1} + \varepsilon\}$. The linear program has variables $\ell_1, \dots, \ell_k$. Let $\varepsilon$ be as in (1). We minimize the value of $E_t = \ell_k + 1$ subject to:

$$\begin{aligned}
\ell_i &\leq \ell_{i+1}, && \forall i = 1, \dots, k-1, && (2)\\
\ell_i &\geq \text{lbound}(v_i), && \forall i = 1, \dots, k, && (3)\\
\ell_i &\leq \text{ubound}(v_i), && \forall i = 1, \dots, k, && (4)\\
\ell_i &\geq \ell_j - 1, && \forall v_iv_j \in E, v_i < v_j, && (5)\\
\ell_i + \varepsilon &\leq \ell_j - 1, && \forall v_iv_j \notin E, v_i < v_j. && (6)
\end{aligned}$$

We solve the same linear program for the other ordering of the vertices of $C_t$. If none of the two programs is feasible, we report that no bounded representation exists. If at least one of them is feasible, we take the solution minimizing $E_t$.

**Proposition 6** *The* BoundRep *problem can be solved in polynomial time.*

This linear program can be solved in time $\mathcal{O}(k^2r + kD(r))$, using the Bellman-Ford algorithm [2, Chapter 24.4]. In the next section we improve the time complexity in the case of BoundRep to $\mathcal{O}(k^2 + kD(r))$.

## 4 Shifting Algorithm for BoundRep with Prescribed $\blacktriangleleft$

The goal of this section is to prove Theorem 2. We solve the linear program described in Section 3 by a combinatorial algorithm based on shifting of intervals.

### 4.1 Preliminaries

First, we suppose that the unit interval graph is connected, and that one left-to-right ordering $<$ of the intervals is prescribed.

Let $\mathfrak{Rep}$ denote the set of all $\varepsilon$-grid representations in the ordering $<$ satisfying the lower bounds (3) (but not necessarily the upper bounds (4)). Clearly, this set is non-empty. There is a natural partial ordering of these representations: For $\mathcal{R}, \mathcal{R}' \in \mathfrak{Rep}$, we say that $\mathcal{R} \le \mathcal{R}'$ if and only if $\ell_i \le \ell'_i$ for every interval $v_i \in V(G)$. The poset $(\mathfrak{Rep}, \le)$ is a (meet)-semilattice:

**Lemma 7** *Every non-empty $S \subseteq \mathfrak{Rep}$ has an infimum* $\inf(S)$.

We call the infimum $\inf(\mathfrak{Rep})$ the *left-most representation.* Clearly, if this representation satisfies the upper bound constraints, then it is an optimal solution of the linear program. On the other hand, it can be proved that there exists a representation $\mathcal{R}'$ satisfying both lower and upper bound constraints if and only if the left-most representation satisfies the upper bound constraints. Therefore, we can solve the linear program by constructing the left-most representation.

Suppose that we construct some initial $\varepsilon$-grid representation that is not the left-most representation. We transform this initial representation in $\mathfrak{Rep}$ into the left-most representation by applying the *left-shifting operation*, which shifts one interval of the representation by $\varepsilon$ to the left such that this shift maintains the correctness of the representation.

**Proposition 8** *For $\varepsilon = \frac{1}{K}$ and $K \ge \frac{n}{2}$, an $\varepsilon$-grid representation $\mathcal{R} \in \mathfrak{Rep}$ is the left-most representation if and only if it is not possible to shift any single interval to the left by $\varepsilon$ while maintaining correctness of the representation.*

An interval $v_i$ is called *fixed* if it is in the left-most position, i.e., $\ell_i = \min\{\ell'_i : \mathcal{R}' \in \mathfrak{Rep}\}$. A representation is the left-most representation if and only if every interval is fixed.

An interval $v_i$, having $\ell_i > \text{lbound}(v_i)$, can be shifted to the left by $\varepsilon$ if it does not make the representation incorrect, and the incorrectness can be obtained in two ways. First, there could be some interval $v_j$ such that $v_j < v_i$, $v_iv_j \notin E(G)$, and $\ell_j + 1 + \varepsilon = \ell_i$; we call $v_j$ a *left obstruction* of $v_i$. Second, there could be some interval $v_j$ such that $v_i < v_j$, $v_iv_j \in E(G)$, and $\ell_i + 1 = \ell_j$; then we call $v_j$ a *right obstruction* of $v_i$. Each vertex has at most one obstruction of each type, and these obstructions are always the same: If $v_i$ has a left obstruction, it is the first non-neighbor of $v_i$ on the left. If $v_i$ has a right obstruction, it is the right-most neighbor of $v_i$.

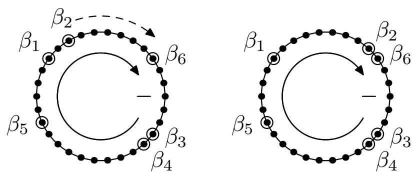


Figure 2: In the position cycle on the left, we shift $\beta_2$ in clockwise direction towards $\beta_6$, which gives a new representation with position cycle on the right. After left-shifting, $v_6$ is not necessarily an obstruction of $v_2$.

For each interval in some $\varepsilon$-grid representation with $\varepsilon = \frac{1}{K}$, we can write its position in the form $\ell_i = \alpha_i + \beta_i\varepsilon$, where $\alpha_i \in \mathbb{Z}$, $\beta_i \in \mathbb{Z}_K$. We can depict $\mathbb{Z}_K = \{0, \dots, K-1\}$ as a cycle with $K$ vertices where the value decreases clockwise. The value $\beta_i$ assigns to each interval $v_i$ one vertex of the cycle. Together with placed $v_i$'s, we call this the *position cycle.*

The position cycle allows us to visualize and work with left-shifting very intuitively. When an interval $v_i$ is shifted by $\varepsilon$ to the left, $\beta_i$ is cyclically decreased by one, so it is moving clockwise along the cycle. If $v_j$ is the left obstruction of $v_i$, then $\beta_j = \beta_i - 1$; if it is its right obstruction, then $\beta_i = \beta_j$. So in both cases $\beta_j$ is very close to $\beta_i$. See Figure 2 for an example.

### 4.2 The Shifting Algorithm

The shifting algorithm solves the linear program of Section 3 in time $\mathcal{O}(k^2 + kD(r))$, where $k$ is the number of vertices of the component and $r$ is the size of the input describing bounds of the component.

The algorithm works in three basic steps:

1. Construct an initial $\varepsilon$-grid representation (in the ordering $<$) having $\ell_i \ge \text{lbound}(v_i)$ for all intervals, using the algorithm of Corneil et al. [3].
2. Shift intervals to the left while maintaining correctness of the representation until the left-most representation is constructed, using Proposition 8.
3. Check whether the left-most representation satisfies the upper bounds. If so, this representation satisfies all bound constraints and solves the linear program of Section 3. Otherwise, no representation satisfying all bound constraints exists, and thus the linear program has no solution.

Since $\varepsilon$ can be very small, we do not shift the intervals on the $\varepsilon$-grid. Instead, we position the intervals on a larger $\Delta$-grid, $\Delta = \frac{1}{n^2}$, and shift them there. When some interval becomes fixed, it is removed from the position cycle for the $\Delta$-grid, and its precise position on the $\varepsilon$-grid is calculated. Working on the $\Delta$-grid allows to reduce the time complexity from $\mathcal{O}(k^2 D(r))$ to $\mathcal{O}(k^2 + kD(r))$. See [9] for details.

We shift unfixed intervals by using gaps in the position cycle: When we shift interval $v_i$ from $\ell_i$ to $\ell'_i$, we decrease $\beta_i$ to $\beta_\ell + 1$, where $\beta_\ell$ is the first $\beta_j$ we encounter when we move clockwise from $\beta_i$. We also check whether this shift is valid with respect to fixed intervals and the lower bound constraint. The interval $v_i$ can become fixed in two ways: Either $\ell'_i \leq \text{lbound}(v_i)$ or there is some fixed obstruction $v_j$ to which $v_i$ is shifted. This can be checked in $\mathcal{O}(1)$ time. If $v_i$ becomes fixed, it is removed from the position cycle and its position on the $\varepsilon$-grid is calculated.

We start with an initial $\Delta$-grid representation satisfying all lower bounds such that $\ell_i \leq \text{lbound}(v_i)+\Delta$ for at least one $v_i$. Then every interval can be shifted in total by distance at most $\mathcal{O}(k)$ from the initial position, since the component is connected. To obtain the initial representation, we use the algorithm in [3].

The shifting of unfixed intervals proceeds in two phases. The first phase creates one big gap by clustering all $\beta_i$'s in one part of the cycle. To do so, we shift intervals in the order given by the position cycle. We obtain one big gap of size at least $n(n-1)$. In the second phase, we use this big gap to shift intervals one by one, which also moves the cluster along the position cycle. In both phases, if some interval becomes fixed, it is removed from the position cycle. The second phase finishes when each interval becomes fixed and the left-most representation is constructed.

Now we are ready to prove Theorem 2:

**Proof.** [Theorem 2, sketch] We process the components $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$ from left to right, and for each component we solve two linear programs by constructing the left-most representation. We use the shifting algorithm described in this section. By Proposition 8, the algorithm stops when each interval is fixed, and it indeed constructs the left-most representation. As already argued, for this representation it is sufficient to check the upper bounds.

Concerning complexity, each interval is shifted by distance at most $k$. The first phase performs $\mathcal{O}(k)$ shifts. In the second phase, each interval is shifted by at least $\frac{n-1}{n}$ unless it becomes fixed. So in total, the second phase performs $\mathcal{O}(k^2)$ shifts. Each shift can be implemented in time $\mathcal{O}(1)$ unless the interval becomes fixed. We need additional time $\mathcal{O}(kD(r))$ for precomputation and to compute exact positions on the $\varepsilon$-grid every time an interval becomes fixed. Thus the total time per component is $\mathcal{O}(k^2 + kD(r))$ and we get $\mathcal{O}(n^2 + nD(r))$ for the entire graph. $\square$

## 5 RepExt(UNIT INT)

The RepExt(UNIT INT) problem can be solved using Theorem 2:

**Proof.** [Corollary 3] A connected component $C$ of $G$ is called *located* if it contains at least one pre-drawn interval, and *unlocated* otherwise. Unlocated components can be placed far to the right and we can deal with them using a standard recognition algorithm.

Concerning located components $C_1, \dots, C_c$, they have to be ordered from left to right according to the left-to-right ordering of the pre-drawn intervals (otherwise the problem has no solution). This gives the required ordering $\blacktriangleleft$. We straightforwardly construct the instance of BoundRep with this $\blacktriangleleft$ as follows. For each pre-drawn interval $v_i$ at position $\ell_i$, we set $\text{lbound}(v_i) = \text{ubound}(v_i) = \ell_i$. For the rest of intervals, we set no bounds. Clearly, this instance of BoundRep is equivalent to the original RepExt(UNIT INT) problem. And we can solve it in time $\mathcal{O}(n^2 + nD(r))$ using Theorem 2. $\square$

## References

[1] T. Bläsius, I. Rutter. *Simultaneous PQ-ordering with applications to constrained embedding problems.* Proc. SODA'13, pp. 1030-1043.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. *Introduction to Algorithms, 3rd edition.* The MIT Press, 2009.

[3] D.G. Corneil, H. Kim, S. Natarajan, S. Olariu, A.P. Sprague. *Simple linear time recognition of unit interval graphs.* Inform. Process. Lett. 55(2):99-104, 1995.

[4] X. Deng, P. Hell, J. Huang. *Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs.* SIAM J. Comput. 25(2):390-403, 1996.

[5] M. Fürer. *Faster integer multiplication.* SIAM J. Comput. 39(3):979-1005, 2009.

[6] P.C. Gilmore, A.J. Hoffman. *A characterization of comparability graphs and of interval graphs.* Can. J. Math. 16:539-548, 1964.

[7] G. Hajós. *Über eine Art von Graphen.* Internationale Mathematische Nachrichten 11:65, 1957.

[8] P. Klavík, J. Kratochvíl, T. Krawczyk, B. Walczak. *Extending partial representations of function graphs and permutation graphs.* Proc. ESA'12, pp. 671-682.

[9] P. Klavík, J. Kratochvíl, Y. Otachi, I. Rutter, T. Saitoh, M. Saumell, T. Vyskočil. *Extending partial representations of proper and unit interval graphs.* `arxiv:1207.6960` (2012).

[10] P. Klavík, J. Kratochvíl, Y. Otachi, T. Saitoh. *Hardness of partial representation extension for chordal graphs.* Proc. ISAAC'12, pp. 444-454.

[11] P. Klavík, J. Kratochvíl, Y. Otachi, T. Saitoh, T. Vyskočil. *Linear-time algorithm for partial representation extension of interval graphs.* In preparation.

[12] P. Klavík, J. Kratochvíl, T. Vyskočil. *Extending partial representations of interval graphs.* Proc. TAMC'11, pp. 276-285.

[13] J.P. Spinrad. *Efficient Graph Representations.* Field Institute Monographs, 2003.

# Augmentability of Geometric Matching and Needlework

Tillmann Miltzow[*]

**Abstract**

Given $2n$ points in the plane, it is well-known that there always exists a perfect straight-line non-crossing matching. We show that it is NP-complete to decide if a partial matching can be augmented to a perfect one, via a reduction from 1-in-3-SAT. This result also holds for bichromatic matchings. From there by an easy reduction scheme we can also show that the same augmentability problem is NP-complete for planar 2-regular, 3-regular, 4-regular and 5-regular graphs.

**Introduction** Matchings appear in everyday life and are long studied mathematical objects. They give rise to many natural mathematical questions and have many applications.

It is well-known that any point set in general position in the plane with an even number of points admits a perfect non-crossing straight line matching. [2] Consider the matching with smallest total edge-length; it is necessarily non-crossing.

For related augmenting problems we refer to a recent survey from Hurtado and Cs. D. Tóth [1]. After we present the proof of our main Theorem 1, we will present applications in the Paragraph Needlework.

**Definitions** To avoid confusions we repeat some definitions. A *planar straight line graph* (PSLG) is a geometric graph; the vertices are points embedded in the plane and the edges are non-crossing line segments. A PSLG is a *partial geometric matching* if each vertex is incident to at most one edge. In a *perfect geometric matching* every vertex is incident to exactly one edge. If the underlying point set is colored red and blue, we say a PSLG is *bichromatic* if no two vertices with the same color have a common edge. Here, we say that a PSLG $G = (V, E)$ augments PSLG $G' = (V', E')$ if they have the same vertex set and $E' \subseteq E$. We always talk exclusively about straight line embeddings. We say that two points $p, q$ in the plane *see each other* If the line segment $pq$ doesn't cross any other obstacle. We will denote red and blue vertices by $\bullet$ and $\circ$. A Graph is $l$-regular if every vertex has degree exactly $l$. The problem of 1-in-3-SAT is to decide whether there is an assignment of the variables of a boolean formula such that exactly one literal is true in every clause. Every clause consists of exactly 3 literals for such formulas. For a graph class $\mathcal{G}$ the geometric augmentability problem is to decide whether a given input PSLG $G$ can be augmented to belong to $\mathcal{G}$.

[*]Institute of Computer Science, Freie Universität Berlin, Germany. t.m@fu-berlin.de

## Matchings

**Theorem 1 (Augmenting Matchings)** *It is NP-complete to decide whether a partial geometric matching can be augmented to a perfect one, both for the monochromatic and bichromatic case. The problem remains NP-complete even if there are no two vertices of the same color that can see each other.*

**Proof.** We prove the bichromatic case first. It can be checked in quadratic time whether a given set of edges augments a matching and has no crossings. Thus the decision problem lies in NP. We describe gadgets and then describe how to encode an instance of 1-in-3-SAT in an instance of our augmentability problem using these gadgets [4]. The construction not surprisingly consists of *variable gadget*s and *clause gadget*s. We will use *lane*s to transport the information from the variable gadgets to the clause gadgets. Lanes might have to cross each other on their way (One could avoid crossings, but this would make the construction more complicated.) from the variable gadget to the clause gadget and at other points, which will become apparent later. For this purpose *junction*s are introduced. To encode the negation of a variable, we construct *multiplier*s, which also serve to split a lane in two. This is necessary as our variable gadgets only emit two lanes initially. Thus, we can create as many lanes as we want with the multiplier to transport the information from a certain variable. This is necessary if a variable appears in more than one clause of the original formula. We have the problem that some lanes are not used. We let all of them go into a common *basin*, where they can be taken care of easily. See Figure 1 and 3. All of our gadgets consist of the edges and free vertices arranged in the plane. For the clarity of the drawings we did not draw the edges separately, but let them appear as one PSLG. This is not problematic as one could perturb each edge slightly such that they do not overlap nor cross and still all the visibility properties remain.

The lane(Figure 1.1) consists of a tube bounded by matching edges and unused points. The tube is piked with matching edges to guarantee that each free vertex has exactly two possible neighbors he could be

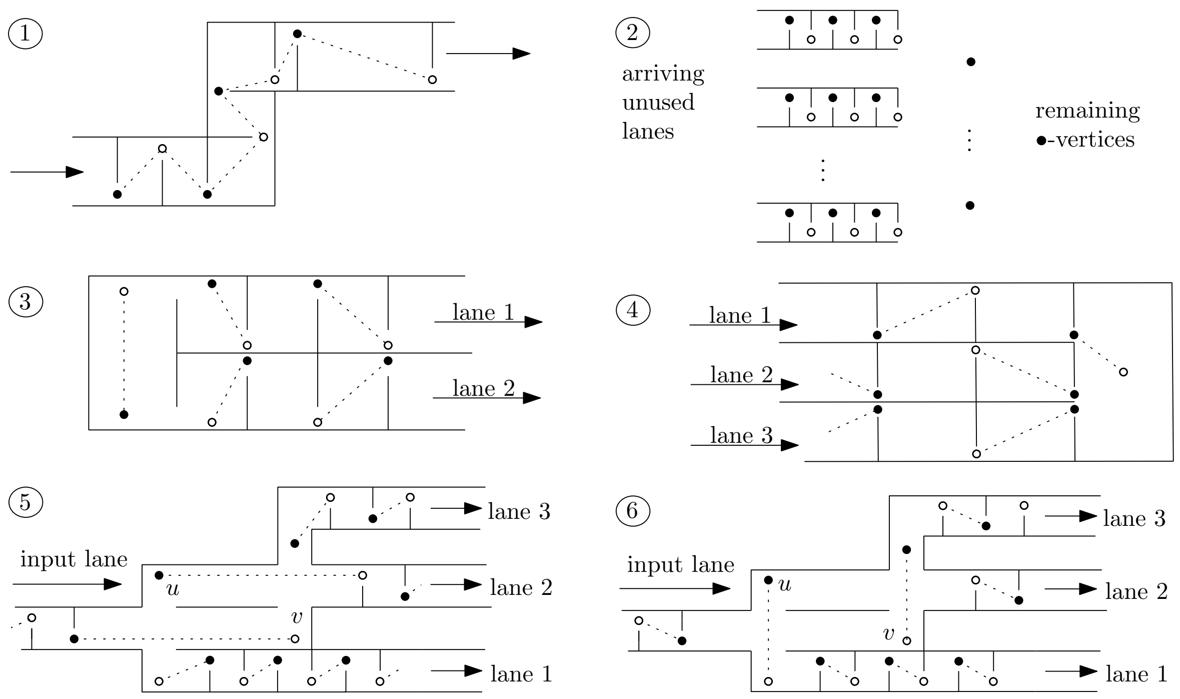


Figure 1

connected to. Thus any matching edge determines the matching of all edges in the lane and there are exactly two ways to match all the free vertices. Also we attach to a lane a direction as we say it starts at a variable gadget and ends in a clause gadget or the basin; maybe bypassing several other gadgets in between.

A lane is defined to *transports the information* ***true*** if the orientation of the edge is ●○(with respect to the direction of the lane) and false if it is ○●. Note that lanes do not need to be straight, but can have any kind of bends.

The variable gadget(Figure 1.3) consists of 2 free-vertices at the start of 2 lanes. We say that a variable is set to `true` if the two free vertices are matched to one another, and `false` otherwise. The upper lane transports the information $x$ and the lower one transports the information $\neg x$ by our convention.

The basin(Figure 1.2) only gathers all the lanes to a common location and place sufficiently many ●-vertices next to these lanes to serve them (i.e. connect the ○-vertices, that are not connected yet.). It holds # (●-vertices in the basin) = # (○-vertices) − #(●-vertices in the remaining construction).

The clause gadget(Figure 1.4) consists of exactly one free ○-vertex, which is exposed to three arriving lanes. The ○-vertex can only be matched to exactly one of the three lanes. Thus all free vertices can be matched iff exactly one of the lanes transports the information `true`.

The multiplier(Figure 1.5&1.6) accepts one arriving

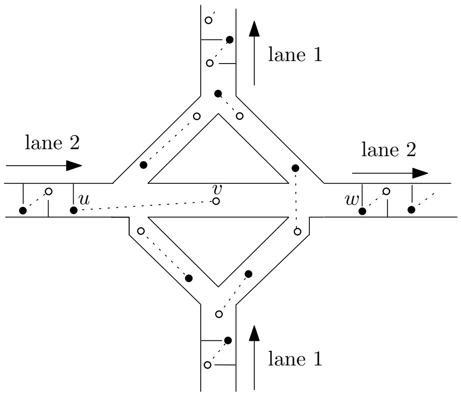


Figure 3

lane and emits 3 leaving ones. Lane 2 and 3 carry the same information as the arriving lane and lane 1 will transport opposite information. Observe that the vertex $v$ in Figure 1 5 can only be adjacent to two vertices. To which is determined by the information of the arriving lane. This in turn determines what $u$ is connected to and will decide the edges for the three emitting lanes.

The junction is depicted in Figure 3 together with one possible assignments of the incoming lanes. Depending on lane 2 the vertex $v$ must be matched with $u$ or $w$. This implies that either the left tunnel or the right tunnel is blocked by an edge. This is no problem when lane 1 carries the information `true`, as lane 1 will not cross lane 2. When lane 1 carries information

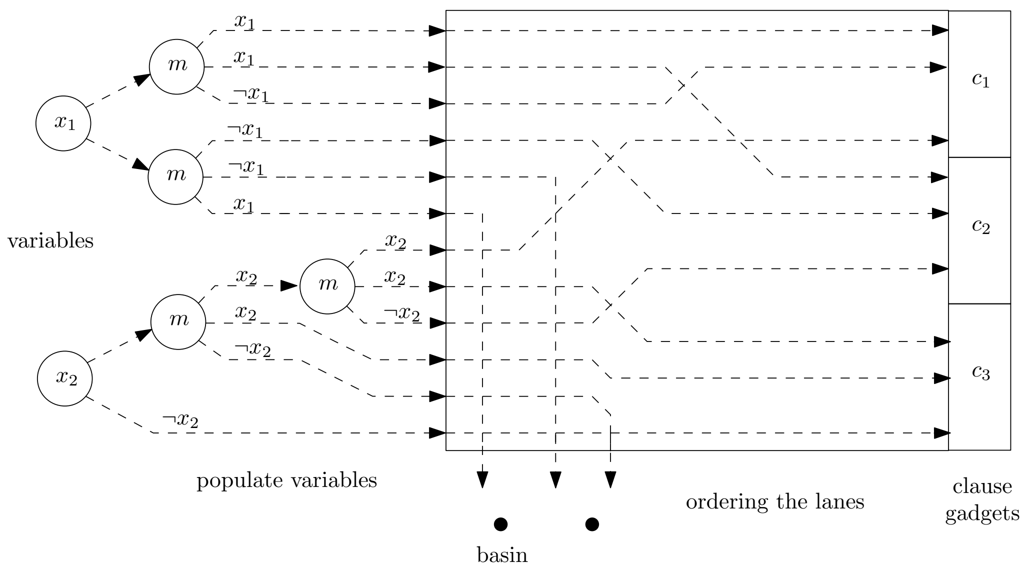


Figure 2

`false` lane 1 uses the tunnel that is not blocked by the edge connected to $v$.

The whole construction is depicted in Figure 2. Let $\varphi$ be an instance of 1-in-3-SAT, we explain how to construct a partial geometric matching $M(\varphi)$. The construction consist of 5 parts. The first part consists merely of variable gadgets. The second part populates the lanes from the variables using the multiplier. This is done till each literal is at least as many times present as it is used in the clauses. The third part is the most important one, here the literals are connected to the corresponding clauses. Note that we have at most a quadratic number of crossings. At the bottom of the third part is the fourth part, the basin, which takes care of overmuch lanes. The last part are the clause gadgets, where the lanes end. It is clear, that this construction can be made in polynomial time. If there is an assignment to the variables such that $\varphi$ is satisfied, we match the variable gadget accordingly and transport these information over the lanes and each clause gadget will be satisfied. Thus it is possible to augment the matching of $M(\varphi)$ to a perfect one. On the other hand if $M(\varphi)$ can be augmented to a perfect matching we get an assignment from the variable gadget, which satisfies each clause.

We will not match vertices of the same color as this is not allowed. Thus inserting tiny segments that block the visibilities of vertices of the same color and no other visibilities doesn't change the complexity of the problem.

As we cannot connect vertices with the same color, ignoring the color restriction does not change the complexity of the problem either. ☺

**Needlework** We show how to apply Theorem 1 to get NP-hardness of other augmentability problems. In the following we consider only graph classes[1] $\mathcal{G}$ that contain only planar graphs and have the following natural property:

$$G_1, G_2 \in \mathcal{G} \Leftrightarrow G_1 \sqcup G_2 \in \mathcal{G}$$

For instance, connected graphs or graphs with bounded diameter do not satisfy this property. We cannot apply our technique to these classes.

A PSLG $N_{\mathcal{G}} \notin \mathcal{G}$ is called an *one pointed needle* for graph class $\mathcal{G}$ if the following conditions are met:

- There exists a distinguished vertex $v \in N_{\mathcal{G}}$ on the outer face of $N_{\mathcal{G}}$.
- When two copies of $N_{\mathcal{G}}$ are connected with an edge the resulting PSLG $K$ lies in $\mathcal{G}$.
- Other edges added to $K$ immediately exclude the option that an augmentation of $K$ is in $\mathcal{G}$.

The PSLG depicted in Figure 4 a) is an one pointed needle for planar 3-regular graphs.

**Theorem 2** *Let $N$ be a* one pointed needle *of the graph class $\mathcal{G}$ then the geometric augmentability problem for $\mathcal{G}$ is NP-hard.*

**Proof.** We make a reduction from the problem in Theorem 1. Let $M$ be a partial monochromatic matching. Replace every vertex $v$ by a small rotated copy of the one pointed needle $N$. The edges of $M$ remain where they were. We denote the resulting PSLG

[1]We consider finite graphs with $V(G) \subset \mathbb{N}$ and therefore think of graph classes as sets.

by $G(M)$. It is clear that $G(M)$ can be constructed in polynomial time and we observe that $M$ can be augmented to a perfect matching iff each needle in $G(M)$ can be matched to another one. ☺

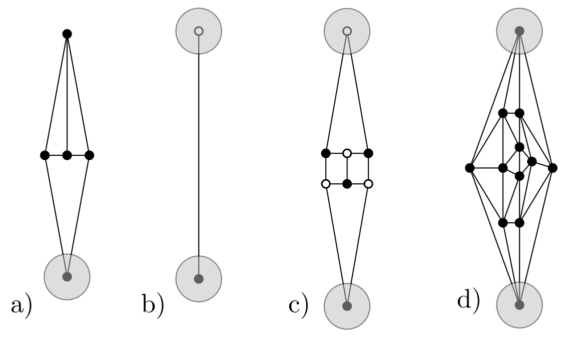


Figure 4

As an application we get: It is NP-complete to decide whether a PSLG can be augmented to a cubic one. Alexander Pilz proved the stronger statement, that the problem remains NP-complete if the input graph is connected [3]. Unfortunately 2- and 4-regular planar graphs admit no one pointed needle, due to parity issues.

A PSLG $N_{\mathcal{G}}$ is called *bichromatic two pointed needle* if the following holds:

- $N_{\mathcal{G}}$ has two distinguished vertices $v$ and $w$ on its outer face, with largest and smallest $y$-coordinate respectively.
- The $v$ and $w$ cannot see each other and we color them blue and red respectively.
- Let $M$ be a geometric perfect bichromatic matching of the distinguished vertices of some copies of $N_{\mathcal{G}}$. Then these copies together with $M$ is in $\mathcal{G}$.
- Any other edge added excludes immediately membership in $\mathcal{G}$.

The PSLGs in Figure 4 b) and c) are examples of bichromatic two pointed needles for bichromatic 2-regular and 3-regular graphs, respectively.

**Theorem 3** *Let $N$ a* bichromatic two pointed needle *of the graph class $\mathcal{G}$ then the geometric augmentability problem of $\mathcal{G}$ is NP-hard.*

**Proof.** Let $M$ be a partial bichromatic matching, without two vertices of the same color visible to one another. Replace every $\bullet$-vertex by a tiny copy of $N$ directed *upwards* and every $\circ$-vertex by such a copy directed *downwards*. Denote the result by $G(M)$. Remember that no two vertices of the same color see each other in $M$. Hence every top vertex must be matched to another top vertex. Therefore augmentations of $M$ can be translated to augmentations of $G(M)$ and vice versa. ☺

A PSLG $N_{\mathcal{G}}$ is a *monochromatic two pointed needle* for a graph class $\mathcal{G}$ if it is a bichromatic one with the exception that, edges between $v$ and $w$ do not necessarily destroy membership in $\mathcal{G}$. The PSLG in Figure 4 d) is a monochromatic two pointed needle for 5-regular planar graphs.

**Theorem 4** *Let $N$ be a* monochromatic two pointed needle *of the graph class $\mathcal{G}$. Then the geometric augmentability problem of $\mathcal{G}$ is NP-hard.*

**Proof.** The proof is exactly the same as in the previous Theorem 3, except that we prevent $vw$-edges by tiny copies of some $G \in \mathcal{G}$. ☺

**Theorem 5** *The geometric augmentability problem is NP-complete for planar, $i$-regular graphs ($i = 2, 3, 4, 5$).*

**Proof.** Membership for these graph classes can be tested in polynomial time. Thus the problem lies in NP. We construct a monochromatic two pointed needle. Take any member of one of these classes and remove an edge $(vw)$. The vertices $v$ and $w$ are the distinguished vertices. Find a straight line embedding, where $v$ and $w$ have largest and smallest $y$-coordinate and cannot see each other. Apply Theorem 4. ☺

We think that needles for many more graph classes can be found easily and the presented reduction scheme is far from being exploited.

**Acknowledgments** The author thanks Andrei Asinowski for his help for improving the presentation and helpful discussions. I also thank Günter Rote for his guidance and support. At last thanks goes to an anonymous reviewer and Joe Mitchell for their advice regarding the presentation.

## References

[1] Ferran Hurtado and Csaba D. Tóth. Plane geometric graph augmentation: a generic perspective. *In Thirty Essays on Geometric Graph Theory (J. Pach, ed.), vol. 29 of Algorithms and Combinatorics*, vol. 29:327–354, 2013.

[2] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane - a survey. *Discrete and Computational Geometry. The Goodman-Pollack Festschrift vol. 25 of Algorithms and Combinatorics*, 25:551 – 570, 2003.

[3] Alexander Pilz. Augmentability to cubic graphs. In *28th European Workshop on Computational Geometry*, 2012.

[4] Michael Sipser. *Introduction to the theory of computation*, volume 2. PWS Publishing Company, 1997.

# Quasi-Parallel Segments and Characterization of Unique Bichromatic Matchings[*]

Andrei Asinowski[†]
Tillmann Miltzow[‡]
Günter Rote[§]

## Abstract

Given $n$ red and $n$ blue points in general position in the plane, it is well-known that there is a perfect matching formed by non-crossing line segments. We characterize the bichromatic point sets which admit exactly one non-crossing matching. We give several geometric descriptions of such sets, and find an $O(n \log n)$ algorithm that checks whether a given bichromatic set has this property.

## 1 Introduction

**Basic notation and preliminary results.** A *bichromatic $(n + n)$ point set $F$* is a set of $n$ blue points and $n$ red points in the plane. We assume that the points of $F$ are in general position, this is, no three points lie on the same line. A *perfect bichromatic straight-line matching* of $F$ is a set of $n$ non-crossing segments that connect points of $F$ so that each segment has one blue and one red endpoint, each blue point is connected to exactly one red point, and vice versa. Such matchings are also known in the literature as *BR-matchings*. However, we denote the colors blue and red by $\circ$ and $\bullet$.

It is well known that any $F$ has at least one BR-matching. One way to construct such a matching is to use recursively the Ham-Sandwich Theorem. In this paper, we characterize bichromatic sets with a *unique* BR-matching. In what follows, $M$ denotes a given BR-matching of $F$. The segments in $M$ are considered directed from the $\circ$-end to the $\bullet$-end. For $A \in M$, the line that contains $A$ is denoted by $g(A)$, and it is considered directed consistently with $A$. For two directed segments $A$ and $B$ for which the lines $g(A)$ and $g(B)$ do not cross, we say that the segments (resp., the lines) are *parallel* if they have the same orientation; otherwise we call them *antiparallel*. If we delete inner points of $A$ from $g(A)$, we obtain two closed *outer rays*: the *$\circ$-ray* and the *$\bullet$-ray*, according to the endpoint of $A$ that belongs to the ray.

The boundary of the convex hull of $F$ will be denoted by $\partial\mathrm{CH}(F)$. Consider the circular sequence of colors of the points of $\partial\mathrm{CH}(F)$; a *color interval* is a maximal subsequence of this circular sequence that consists of points of the same color. In the point set in Fig. 1a, $\partial\mathrm{CH}(F)$ has four color intervals: two $\circ$-intervals (of sizes 1 and 2) and two $\bullet$-intervals (of sizes 2 and 3).

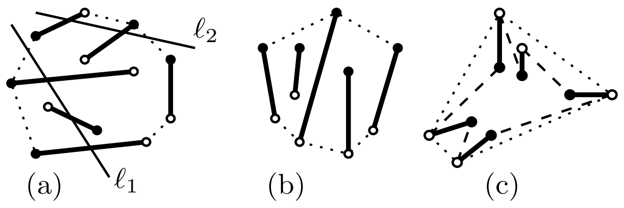


Figure 1: (a) A matching with chromatic cuts; (b) A matching of type L; (c) A matching of type C. (Another matching for the same point set is indicated by dashed lines.)

A *chromatic cut* of $M$ is a line $\ell$ that crosses two segments of $M$ such that their $\bullet$-ends are on different sides of $\ell$. ($\ell$ can as well cross other segments of $M$.) For example, the lines $\ell_1$ and $\ell_2$ in Fig. 1a are chromatic cuts. The matchings in Fig. 1b–c have no chromatic cuts. Matchings without chromatic cuts will play a central role in our work. We specify two types of BR-matchings without chromatic cuts. A *matching of linear type* (or, for shortness, *matching of type L*) is a BR-matching without a chromatic cut such that $\partial\mathrm{CH}(F)$ consists of exactly two color intervals (both necessarily of size at least 2). A *matching of circular type* (or *matching of type C*) is a BR-matching without a chromatic cut such that all points of $\partial\mathrm{CH}(F)$ have the same color. Fig. 1b–c shows matchings of types L and C. We shall prove in Lemma 6 that *any* BR-matching without chromatic cuts belongs to one of these types. Aloupis et al. [1, Lemma 9] proved that if a BR-matching $M$ has a chromatic cut, then there exists a *compatible* BR-matching $M' \neq M$, which means that the union of $M$ and $M'$ is a non-crossing

[*]Supported by by the ESF EUROCORES programme EuroGIGA, CRP ComPoSe, Deutsche Forschungsgemeinschaft (DFG), under grant FE 340/9-1.

[†]Institute of Computer Science, Freie Universität Berlin, Germany. asinowski@gmail.com

[‡]Institute of Computer Science, Freie Universität Berlin, Germany. t.miltzow@gmail.com

[§]Institut für Informatik, Freie Universität Berlin, Germany. rote@inf.fu-berlin.de

set of segments. Thus, *having no chromatic cut is a necessary condition for $M$ being the unique BR-matching of $F$.* However, the matching in Fig. 1c shows that this condition is not sufficient.

**Main results.** We prove the following results.

**Theorem 1** *Let $M$ be a BR-matching without a chromatic cut. Then:*

1. *$M$ is either of type L or of type C.*
2. *If $M$ is of type L, then it is unique.*
3. *If $M$ is of type C, then it is not unique, and there are at least two more matchings.*

Together with the aforementioned necessary condition, this means that $M$ is a unique BR-matching for its point set if and only if it is a matching of type L. Therefore, we study such matchings in more details; see Theorem 2.

**Definition 1** *For any two segments $A, B \in M$, $A \neq B$, we define the relation $A \lhd B$ if $B$ is contained in the right half-plane bounded by $g(A)$ and $A$ is contained in the left half-plane bounded by $g(B)$.*

**Definition 2** *A BR-matching $M$ is called* quasi-parallel *if there exists a directed line $\ell$ such that the following conditions hold:*

- *No segment is perpendicular to $\ell$.*
- *For any $A \in M$, the direction of its projection on $\ell$ (as usual, from $\circ$ to $\bullet$) coincides with the direction of $\ell$.*
- *For any non-parallel $A, B \in M$, the projection of the intersection point of $g(A)$ and $g(B)$ on $\ell$ doesn't lie in the convex hull of the projections of $A$ and $B$ on $\ell$.*

Quasi-parallel segments were introduced as a generalization of parallel segments by Rote [4, 5] in the context of a dynamic programming algorithm for some instances of the traveling salesman problem. In that work, the segments were uncolored; thus, our definition is a "colored" version of the original one. Fig. 2 shows an example of quasi-parallel matching, with horizontal $\ell$.

**Theorem 2** *Let $M$ be a BR-matching of $F$. Then the following conditions are equivalent:*

1. *$M$ is the only BR-matching of $F$.*
2. *$M$ is a matching of type L.*

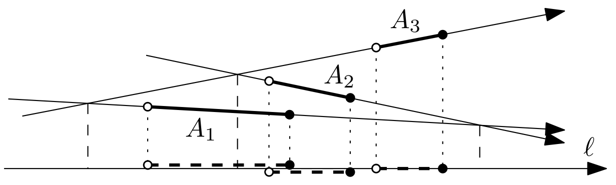


Figure 2: A quasi-parallel matching.

3. *$\partial CH(F)$ consists of two color intervals of length at least 2; and for any $A, B \in M$ ($A \neq B$), either $g(A)$ and $g(B)$ are parallel (including orientation), or their intersection point belongs to the outer rays of the same color.*
4. *The relation $\lhd$ is a linear order on $M$.*
5. *No subset of segments forms one of the three forbidden configurations from Fig. 3.*
6. *$M$ is quasi-parallel.*

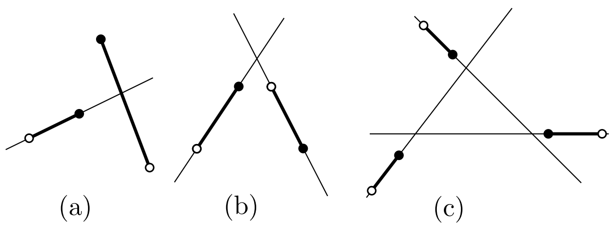


Figure 3: Forbidden patterns for quasi-parallel matchings.

Moreover, if $M$ satisfies any of the above conditions, then any submatching of $M$ satisfies the above conditions. This follows from the fact that conditions 4–6 directly imply that they hold for all subsets. (This is most trivial for condition 5.)

**Related work.** Monochromatic and bichromatic straight-line matchings have been intensively studied in the recent years.

One direction is *geometric augmentation*. Given a matching, one wants to determine whether it is possible to add segments in order to get a bigger matching with a certain property, under what conditions can this be done, how many segments one has to add, etc. See Hurtado and Cs. D. Tóth [3] for a recent survey.

The *bichromatic compatible matching graph* of $F$ has as its node set the BR-matchings of $F$. Two BR-matchings are joined by an edge if they are compatible. Aloupis et al. [1] proved that the bichromatic compatible matching graph is always connected.

**Chromatic cuts.** We start with a simple geometric description of BR-matchings that admit a chromatic cut.

**Lemma 3** *Let $M$ be a BR-matching. $M$ has no chromatic cut if and only if no subset of segments forms one of the two forbidden patterns from Fig. 3(a)–(b).*

**Lemma 4** *Let $M$ be a BR-matching. $M$ has a chromatic cut if and only if there exists a balanced line that crosses a segment of $M$.*

(A line $\ell$ is a *balanced line* if in each open half-plane determined by $\ell$, the number of $\bullet$-points is equal to the number of $\circ$-points.)

**Corollary 5** *Let $M$ be a BR-matching of $F$ with a chromatic cut. Then there is a different matching $M' \neq M$.*

*Remark.* As mentioned in the introduction, Corollary 5 follows from the stronger statement of [1, Lemma 9]: the existence of a compatible matching $M' \neq M$. The full version of our paper gives a simpler alternative proof.

**Lemma 6** *Let $M$ be a BR-matching of $F$ that has no chromatic cut. Then*

- *either all points of $\partial CH(F)$ have the same color,*
- *or the points of $\partial CH(F)$ form two color intervals of size at least 2.*

## 2 Quasi-Parallel, or Linear, Matchings

In this section we sketch the proof of Theorem 2. Conditions 3, 4, 5 are different geometric characterizations of quasi-parallel (or linear) matchings; their proofs are omitted here. We want to give one proof that quasi-parallel matchings are unique. This proof not the shortest possible, but it is probably the most intuitive one. We assume that the equivalence of 2–6 is already established.

Consider a set $V = \{v_1, \ldots, v_m\}$ of pairwise non-crossing unbounded Jordan curves ("ropes"). They partition the plane into $m+1$ connected regions. We assume that they are numbered in such a way that in going from $v_i$ to $v_j$ ($j > i+1$), one has to cross $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$. We will think of them as "vertical" curves that are numbered from left to right. Consider another set $G = \{g_1, \ldots, g_n\}$ of pairwise non-crossing Jordan arcs, such that every curve $g_k$ has its endpoints on two different vertical curves $v_i$ and $v_j$ ($j > i$), has exactly one intersection point each with $v_i, v_{i+1}, v_{i+2}, \ldots, v_j$, and no intersection with the other curves. See Figure 4a for an example. We say that the curves $V \cup G$ form a *partial (combinatorial) grid.*

**Lemma 7** *[The Fishnet Lemma] The "horizontal" arcs $g_k$ of a partial combinatorial grid $V \cup G$ can*

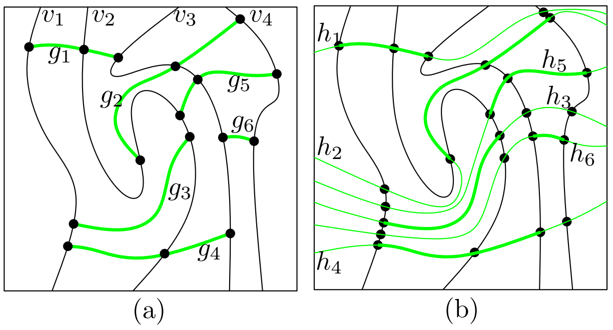


Figure 4: (a) A partial grid. (b) Extension to a full fishnet grid of ropes.

*be extended to pairwise non-crossing unbounded Jordan arcs $h_k$ in such a way that the curves $H = \{h_1, \ldots, h_n\}$ together with $V$ form a full combinatorial grid $V \cup H$: Each "horizontal" curve $h_k$ crosses each "vertical" curve $v_i$ exactly once. See Figure 4b.*

**Theorem 8** *Let $M$ be a matching of type L on the point set $F$. Then $M$ is the only matching of $F$.*

Remark: This statement was proven earlier for compatible matchings [4, Lemma 2] and follows from the connectedness of the bichromatic compatibility graph.

**Proof.** (Sketch) Given a quasi-parallel matching $M$ it is easy to construct a set of Jordan curves $V$ as in Lemma 7 by considering the line arrangement formed by the segments. At each intersection point, the curves switch from one line to the other, and after a slight deformation in the vicinity of the intersections, they become noncrossing. Now assume there is another matching $M'$; $M$ and $M'$ form at least one alternating cycle. Let $G = \{g_1, \ldots, g_k\}$ be the segments of $M'$ in the order when one would traverse this cycle. Clearly, $V, G$ satisfy the condition of the Fishnet Lemma and thus can be extended to a full combinatorial grid. Observe $g_1, \ldots, g_k$ also must be in this order, because (w.l.o.g.) $h_1$ is above $h_2$, $h_2$ above $h_3$, and so on. This means that $g_1$ and $g_k$ are maximally apart, $g_k$ being below $g_1$. So they cannot be connected to the same segment in $M$. ↯ ☺

## 3 Circular Matchings

**Theorem 9** *BR-matchings $M$ of type C have at least two disjoint BR-matchings compatible to $M$.*

**Proof.** (Sketch) Let $M$ be a BR-matching of type C. Since it is not of type L there must be one of the forbidden subconfigurations of Figure 3. Since $M$ has no chromatic cut, there must be 3 segments in $M$ as in Figure 3(a). The triangle bounded by the $\bullet$-rays (w.l.o.g.) is empty of segments. The rest of the plane is partitioned into three convex regions, each defined by a pair of segments as in Figure 5(a). All segments in a region $Q_i$ together with the two defining segments are of type L. Thus in each region there is an alternating path from the •-point of the left bounding segment to the ○-point of the right bounding edge and vice versa. The union of the three paths forms an alternating polygon and thus we have found two different compatible BR-matchings $M'$ and $M''$ depending on which paths we chose. ☺

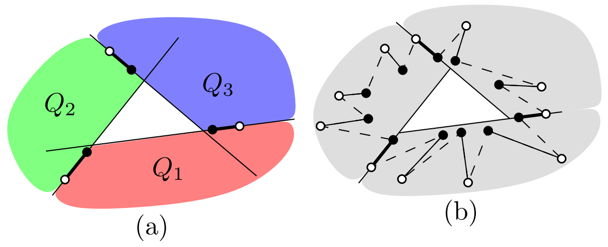


Figure 5: Illustrations to Theorem 9

## 4 Other Results and Remarks

**Parallelizability.** As we showed in Theorem 2, quasi-parallel matchings generalize *parallel matchings* (which consist of parallel segments) in the sense that they are exactly the BR-matchings for which the relation $\lhd$ is a linear order. Therefore, it is natural to ask whether all order types (determined by orientations of triples of points) of bichromatic point sets with a unique BR-matching are realizable by endpoints of a parallel matching.

We construct an example that shows that the answer to this question is negative. The construction is based on the following observation.

**Observation 1** *Let $A, B, C$ be three vertical segments such that $A \lhd B \lhd C$. Denote by $a_1, b_1, c_1$ the upper ends, and by $a_2, b_2, c_2$ the lower ends of the corresponding segments. If the triple $a_1, b_1, c_2$ is oriented counterclockwise, and the triple $a_2, b_2, c_1$ clockwise, then $B$ is shorter than $A$.*

Now, the construction goes as follows. Refer to Figure 6: Consider first three pairs of parallel lines with directions, say, 0°, 60°, and 120°, and three equal vertical segments $A_0, B_0, C_0$ as shown in Figure 6a. Change slightly the slopes of the lines so that each pair will intersect as indicated schematically in Figure 6b, and form the three corresponding segments $A, B, C$. Add vertical segments in the wedges formed by these pairs, as shown in the right part. These six segments form a quasi-parallel matching $M$. Now, assume that there exists a parallel matching $M'$ with corresponding endpoints of the same order type. Denote by $A', B', C'$ the segments of $M'$ that correspond in $M'$ to $A, B, C$. Then, according to Observation 1, $A'$ is longer than $B'$, $B'$ is longer than $C'$, and $C'$ is longer than $A'$. This is a contradiction.

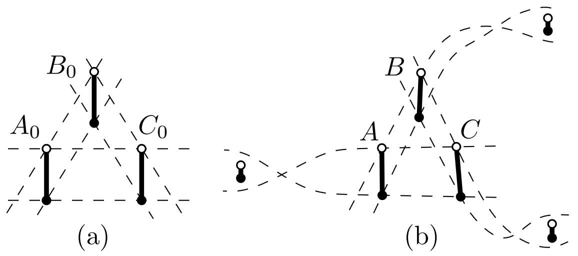


Figure 6: Construction of a non-parallelizable quasi-parallel matching.

**Algorithm.** Given a point set $F$ we can compute a BR-matching $M$ in $O(n \log n)$ time. Now we pretend that $\lhd$ is a linear order of $M$, and sort $M$ by $\lhd$, in $O(n \log n)$ time. If this fails at any stage we know that $\lhd$ is not linear and, therefore, $F$ admits more than one matching. Finally we add the segments one by one in the order given by $\lhd$. By updating the convex hull, and repeating the process in reverse order, one can check in linear time if $\lhd$ is indeed a linear order.

**Theorem 10** *It can be checked in $O(n \log n)$ time whether a bichromatic $(n + n)$-set has a unique non-crossing BR-matching.*

**Acknowledgments.** We thank Michael Payne, Lothar Narins and Veit Wiechert for helpful discussions. An extended version of this abstract [2] contains more details and further results about this topic.

## References

[1] G. Aloupis, L. Barba, S. Langerman, and D. L. Souvaine. Bichromatic compatible matchings. In T. Chan and R. Klein, editors, *Proc. 29th Ann. Symp. Comput. Geometry*, SoCG'13. ACM Press, June 2013, to appear. Preprint arXiv:1207.2375.

[2] A. Asinowski, T. Miltzow, G. Rote. Quasi-parallel segments and characterization of unique bichromatic matchings. arXiv:1302.4400, Feb. 2013.

[3] F. Hurtado and Cs. D. Tóth. Plane geometric graph augmentation: a generic perspective. In J. Pach, ed., *Thirty Essays on Geometric Graph Theory*, Algorithms and Combinatorics, vol. **29**, pp. 327–354. Springer, 2013.

[4] G. Rote. *Two solvable cases of the traveling salesman problem.* PhD thesis, Technische Univ. Graz, 1988.

[5] G. Rote. The $N$-line traveling salesman problem. *Networks* **22** (1992), 91–108.

# Hierarchical Flows with an Application to Image Matching

Stefan Funke [*] Sabine Storandt[†]

**Abstract**

We study the task of computing a matching between two images by formulating it as an instance of the minimum-cost flow problem. Here, we use a cycle cancelling algorithm to find the optimal flow. To reduce the practical runtime, we propose a hierarchical scheme in which the images are first scaled down and then the optimal solution for the smaller problem is used as a starting point for the higher resolution. Our experiments reveal a significant reduction of negative cycle detection operations using our hierarchical approach.

## 1 Introduction

A typical task in image analysis and compression is to find corresponding parts in two somewhat similar images A and B with dimension $n \times m$. One approach to tackle the problem is computing a matching between the two images on pixel level (separated for the red, green and blue color channel) and post-process the family of correspondences for the final result, see Figure 1 for an example. So the color value $val$ of every pixel $a_{ij}$ in image A can be understood as a supply, analogously every value of a pixel $b_{ij}$ in image B as a demand. Given some cost matrix indicating how expensive it is to shift 'mass' from a certain pixel in image A to a pixel in image B, the goal is to fulfil as much of the demand as possible while minimizing the total costs. This can be formulated as an instance of the minimum-cost flow problem on a carefully constructed network. One common method to compute the optimal flow is the so called cycle cancelling approach, which allows to input an initial flow and decreases the costs incrementally until an optimal solution is found. Intuitively, if we scale down the images slightly, the solution for this smaller problem should be close to the optimal solution for the unscaled versions. Therefore we will present a hierarchical approach, that speeds up the cycle cancelling algorithm by identifying good initial flows based on optimal solutions for lower resolution versions.

[*]FMI, Universität Stuttgart, 70569 Stuttgart, Germany, stefan.funke@fmi.uni-stuttgart.de

[†]Institut für Informatik, Albert-Ludwigs-Universität Freiburg, 79110 Freiburg, Germany, storandt@informatik.uni-freiburg.de

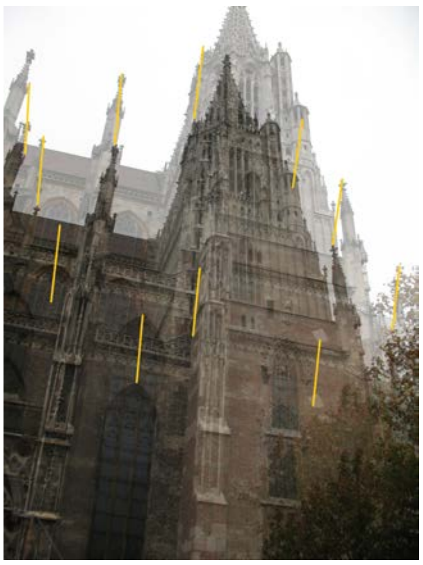


Figure 1: Overlay of two similar images along with some correspondences on pixel level.

## 2 Preliminaries

### 2.1 The Minimum-Cost Flow Problem

In the minimum-cost flow problem, we are given a graph or network $G(V, E)$ with the edges having certain capacities $c : E \to \mathbb{R}^+$ and costs $p : E \to \mathbb{R}$. Moreover we have distinguished source and target vertices $s, t \in V$; the goal is to send $U \in \mathbb{R}^+$ units of flow from $s$ to $t$ such that the edge capacities are respected and the total costs are minimized.
There exist a variety of algorithms which solve this problem optimally, e.g. based on linear programming or combinatorial approaches, see [1] for an overview. Several algorithms involve the construction of the so called residual network. Here, given some feasible initial flow $F'$, a reverse edge is added for each edge $(u, v) \in E$ which transports flow according to $F'$. This edge $(v, u)$ receives a capacity that equals the amount of flow send through $(u, v)$, so $c(v, u) = f(u, v)$, and costs $p(v, u) = -p(u, v)$. The flow $F'$ is a minimum-cost flow if and only if the residual network does not contain any negative cycles. Such a negative cycle can be 'cancelled' by sending the amount of flow through it that is determined by the minimal capacity among the participating edges. So algorithms that repeatedly identify and cancel negative cycles provide optimal solutions for the minimum-cost flow problem.

## 2.2 Cycle Cancelling

There are various approaches to detect negative cycles in (residual) flow networks. In this paper we will concentrate on the Robust Dijkstra (RD) method [2], while our hierarchical flow construction scheme allows for plugging in an arbitrary cycle cancelling algorithm.

RD is an $\mathcal{O}(n)$-pass algorithm expanding the idea of Dijkstra's algorithm. Here we also assign to every node $v$ a distance label $d(v)$ (initially 0), to which we refer from now on as the node's potential. Likewise, we attach predecessor labels $prev(v)$ (initially *null*). In contrast to Dijkstra, we furthermore relate to every node its potential difference $\Delta(v)$ (initially 0). Initially all nodes are stored in a max-heap $Q$ sorted by their $\Delta$-values. To assure that every node is examined at most once in every pass, the nodes get divided in the following into $Q$ and a queue $S$. In every step, we extract the node $v$ with maximal $\Delta$ from $Q$ and check for all edges $(v, w)$ if $d(v) + c(v, w) < d(w)$. If this is the case, we replace $d(w)$ with the new potential $d'(w) = d(v)+c(v, w)$ and set $\Delta(w) = d(w) - d'(w), prev(w) = v$. Obviously $\Delta$ always stays non-negative while $d$ strictly decreases. If $w$ has already be examined in the current pass, its pushed into $S$, otherwise into $Q$ (if it was pushed there before, we perform *decrease key*). If $Q$ runs empty, we swap the content of $S$ and $Q$ and heapify the latter. So if both $Q$ and $S$ become empty, the graph does not contain any negative cycles and RD terminates. Otherwise a negative cycle in the graph equals a cycle in the tree based on the *prev* tags of the nodes, which we call dependency tree/forest. It can be identified by backtracking after each change of a *prev* tag.

## 3 Constructing the Flow-Graph

Given the two images A and B with dimensions $n \times m$, we want to construct an instance of the minimum cost flow problem for an arbitrary cost matrix. For that purpose, we create a node for every pixel in each of the images. To allow pixels of image A to be matched to arbitrary pixels in image B via respective flows, we add edges from every pixel $a_{ij}$ in image A to all pixels $b_{kl}$ in image B with unlimited capacity and costs $d(a_{ij}, b_{kl})$ for some distance metric. Moreover we add a source $s$ and a target $t$. We connect $s$ to every node $a_{ij}$ by an edge with costs of zero and a capacity which equals the respective color value *val* of pixel $a_{ij}$ in the image. Analogously we connect all nodes $b_{kl}$ to $t$, resulting in a total number of $n^2m^2 + 2nm$ edges. Let $M_A, M_B$ denote the total mass (i.e. the summed color values) of image A and B, then we aim to send $M = \min(M_A, M_B)$ units of flow from $s$ to $t$.

## 4 The Basic Hierarchical Algorithm

We now present a hierarchical approach that works for arbitrary cost matrices. It starts by scaling down both of the input images by a certain factor and solving the according minimum-cost flow problem optimally. Then this solution is used an initial flow for the next problem in the hierarchy with a smaller scaling factor, see Algorithm 1 for the pseudo-code.

**Algorithm 1**: Hierarchical Flow Algorithm

```
   Input: A[n][m], B[n][m], s (initial scaling factor)
   Output: List of optimal flows $F^*$
1  begin
2      $A_s \leftarrow scale(A, 2^s)$;
3      $B_s \leftarrow scale(B, 2^s)$;
4      $F \leftarrow findInitialSolution(A_s, B_s)$;
5      $F^* \leftarrow findOptimalFlow(A_s, B_s, F)$;
6      while $2^s > 1$ do
7          $s \leftarrow s - 1$;
8          $A_s \leftarrow scale(A, 2^s)$;
9          $B_s \leftarrow scale(B, 2^s)$;
10         $F \leftarrow lift(A_s, B_s, 2^s, F^*)$;
11         $F^* \leftarrow findOptimalFlow(A_s, B_s, F)$;
12     return $F^*$;
13 end
```

Here, *scale* takes a matrix $X[n][m]$ and an integer $t$ as input, with $t$ being a divisor of $n$ and $m$. The output is the matrix $X'[n/t][m/t]$ with $x'_{i,j} = \sum_{l=it}^{it+t-1} \sum_{k=jt}^{jt+t-1} x_{l,k}$.

The procedure *findInitialSolution* computes a greedy flow of $M$ units according to the well known initial solution for the transportation problem.

*findOptimalFlow* is an arbitrary cycle-cancelling algorithm. Given an initial flow $F$ of the required amount, it first creates the residual graph $G'$ of $G$ according to $F$ and then invokes the cycle cancelling process until cost optimality is achieved.

The function *lift* takes the optimal solution of hierarchical level $s$ and transforms it to a solution on level $s-1$. Let $f^* = (i, j, i', j', u) \in F^*$ denote a flow from $a_{ij} \in A_{s-1}$ to $b_{i'j'} \in B_{s-1}$ of $u$ units. In $A_s$ and $B_s$, there are now four (sub)nodes corresponding to the (super)nodes $a_{ij}$ and $b_{i'j'}$ respectively. If $u$ is less than $val(a_{ij})$ or $val(b_{i'j'})$ and more than one entry of a subnode is greater than zero, it is not definite how to transform the flow to the next higher level. Therefore *lift* uses a greedy approach; it starts with the upper left subnodes $a_{2i,2j}, b_{2i',2j}$ and adds a flow between them with $u' = \min(val(a_{2i,2j}), val(b_{2i',2j'}), u)$ units. Then we reset $u$ to $u-u'$ and analogously decrease the values of $a_{2i,2j}$ and $b_{2i',2j'}$. If $u = 0$ the flow is completely lifted, otherwise it yields either $val(2i, 2j) = 0$ or $val(2i', 2j') = 0$. As soon as the value of a subnode turns zero, we move on to the next one belonging to the same supernode and proceed as before. We stop as soon as $u$ runs zero (which obviously must always be possible).

## 5 An improved Approach for the $L_1$ Metric

### 5.1 Modifying the Graph Structure

For image matching applications the $L_1$ metric often provides reasonable costs, because it counts the number of pixels the 'mass' has to be shifted. Luckily, using the $L_1$ metric, one can find a more efficient graph model. This new model is based on the fact, that a flow under the $L_1$ metric can be decomposed into miniflows of length one without changing the overall costs. Moreover this decomposition can be done such that at first there are only horizontal miniflows going to the left or right neighbour, followed by only vertical ones.
Therefore a new graph $G^*$ can be created as follows: The node set and edges adjacent to $s$ and $t$ stay unmodified. Every node in A gets an edge to its left and its right neighbour (if it exists), each with unbounded capacity and cost 1. Analogously every node in B is now connected with its neighbour above and below, also with unbounded capacity and cost 1. Moreover every node in A has one edge to its equivalent in B with the same coordinates. These edges have zero costs and are unbounded as well. Obviously every flow in G can also be represented as a flow in $G^*$ with equal costs. But now the out-degree of a node of A/B in $G^*$ is three at maximum (six in the residual graph) and therefore the total number of edges is $O(nm)$ only.

But this change of the graph structure also requires the modification of the *lift* procedure: In the residual graph of $G^*$ negative cycles are possible that use only edges between nodes of A/B. Such a circle could not occur in the residual network of $G$. Furthermore an initial set of flows for *findOptimalFlow* is only feasible iff every flow can be divided in a horizontal, a static (flows between nodes in A and B with the same coordinates) and a vertical component in that order and no supply is overused and no demand oversupplied. To take care of these new conditions, *lift* proceeds in five phases:
**1.** Decomposing all input flows in miniflows of length one and storing them divided in horizontal, static and vertical flows in matrices: A positive entry in the horizontal matrix equals a flow to the right neighbour with mass according to the entry, a negative entry means flow to the left. Analogously a positive entry in the vertical flow matrix means flow down, a negative entry flow up. In the static flow matrix there are positive entries only.
**2.** Upgrading all horizontal flows: Here, we parse greedily through the source subnodes and shift the minimum mass of the belonging entry and the total flow mass $U$ horizontally until a subnode of the target is reached. Then, we update $U$ and the demand/supply values accordingly. We proceed until $U = 0$. To make sure that the mass of the source node is always greater or equal to $U$, we first go through each row from the left to right and consider all miniflows going right and afterwards vice versa.
**3.** Upgrading static flows where the mass of the source node is smaller or equal to the one of the target node: Because these are flows from A to B the general *lift* algorithm can be used here.
**4.** Upgrading vertical flows: First we convert the miniflows back to long flows in a greedily fashion. These long flows are again flows between A and B, because the only possible sources are such supernodes in A with their mass greater than the mass of the equivalent supernode in B. So general *lift* can construct a family of flows between the respective subnodes. Then we update the vertical flow matrix by subtracting the flow mass from all involved entries. This phase is finished when all entries in the vertical flow matrix are zero.
**5.** Upgrading the remaining static flows.

**Lemma 1** *The* lift *procedure always returns a feasible flow of $M$ units for the next hierarchical level.*

**Proof:** *lift* always gets a flow of $M$ units as an input and upgrades all flows to the next level, so the total flow is preserved.
For the general *lift* algorithm the feasibility is obvious. In the special case of the $L_1$ metric phase 2 clearly does not lead to a violation of feasibility. In phase 3 updating those static flows does not overfill any demands in B, because there are no horizontal flows between supernodes left and so if there is still too little mass in a supernode in A, horizontal flows between the belonging subnodes cannot change these lack. In phases 4 and 5 only flows between A and B are created and all demands get fulfilled if possible. So all in all *lift* only creates flows that can be divided first in a horizontal, followed by a static and a vertical component. All flows between supernodes are upgraded and only flows between subnodes get added and therefore the resulting flow does not hurt any supplies or demands.■

### 5.2 Improving the Robust Dijkstra Method

The overall runtime of our approach depends strongly on the RD method. Therefore we now introduce some modifications of RD, that make it much more efficient while running on $G^*$.

**Decreasing the initial size of Q**

**Lemma 2** *In $G^*$ every negative cycle using non-negative edges can be detected by starting RD from a negative source.*

**Proof:** In $G^*$ edges with negative costs exist only inside a layer. So negative cycles consisting of negative edges only can occur exclusively between neighbours in one layer and so they can easily be cancelled. Given a node $v$, one can find a negative cycle by starting RD at it. If $v$ is not a negative source, either there is an incoming negative edge or no outgoing negative edge. The latter can not be true, because RD would stop immediately in that case and no negative cycle would be detected. But if $v$ has a negative incoming edge $(u, v)$, the same negative cycle could be detected starting at $u$. Because the cycle can not consist of edges with negative cost only, there always has to be a negative source one can start from.■

This cuts the initial size of $Q$ to at most half the number of nodes.

**Reusing the dependency forest**

Especially if RD detects only a short cycle, the overhead of constructing the dependency forest can be huge. Therefore it is more efficient to keep the parts of the forest that are not affected by the cycle cancelling for the next iteration.
To realize this approach, we set the initial potentials $d(v) = \infty \ \forall v \in V$. Moreover we identify all negative sources, set their potentials to zero and push them into $Q$. Then we run conventional RD (until a cycle is detected) with the slight change that if we decrease the potential of $v$ from $d(v)$ to $d'(v)$, we set $\Delta(v) = min\{d(v) - d'(v), -d'(v)\}$. If a cycle is found, we identify the node of the cycle with the lowest tree depth. After the cancelling of the cycle we delete the whole subtree $T_v$ beneath $v$ by setting $prev(w) = null$, $d(w) = \infty$, $\Delta(w) = 0 \ \forall w \in T_v$. Moreover we check for new negative sources originated by the cycle cancelling. These nodes again get a potential of zero and are pushed in $Q$ or $S$. Furthermore we also push nodes in $Q$ or $S$ that are not in $T_v$ themselves, but have adjacent edges to nodes in $T_v$. Then again we use RD as described before to identify the next negative cycle. If $Q$ and $S$ become empty, RD terminates.

**Lemma 3** *After termination of modified RD an optimal flow is retrieved.*

**Proof:** We claim that $\forall v \notin Q, S : d(v) + c(v, w) \geq d(w) \ \forall (v, w) \in E$ (loop invariant). This is certainly fulfilled after the initialization, because here all nodes not in $Q$ or $S$ have potential $d(v) = \infty$. For conventional RD it yields: As long as a node $v$ is not added to $Q$ its potential stays the same. If a node is pushed into $Q$ or $S$, its potential decreased strictly. If a node is removed from $Q$ all potentials of adjacent nodes got updated and so the loop invariant is true. If a cycle is detected, the nodes not in $Q$ or $S$ are the ones with potentials $d(v) = \infty$ or nodes that were not in $Q$ or $S$ before and do not have adjacent edges to nodes in the subtree affected by the cycle. So for the latter the loop invariant was true before the cycle deletion and neither their potentials nor the potentials of their neighbours did change. So all in all the loop invariant is fulfilled after each pass and so after termination there are no negative cycles left. Therefore modified RD finds a minimum-cost flow in $G^*$.■

| | s=0 | s=1 | s=2 | s=6 |
|---|---|---|---|---|
| 128x128 | 285.702 | 12.491 | 12.512 | 12.247 |
| 64x64 | | 58.781 | 4.305 | 4.047 |
| 32x32 | | | 7.578 | 983 |
| 16x16 | | | | 204 |
| 8x8 | | | | 48 |
| 4x4 | | | | 17 |
| 2x2 | | | | 2 |
| total | 285.702 | 71.272 | 24.395 | 17.548 |

Table 1: Number of cycle cancelling operations for different resolutions dependent on the initial scaling factor $s$.

## 6 Experimental Results

We implemented our hierarchical flow scheme in C++ and tested our approach on real-world images as well as artificial input. We observed a significant reduction of cycle cancelling operations for each of the inputs when using our hierarchical approach. In Table 1 we picked one real-world example with dimensions 128x128 and solved the problem without scaling (s=0) as well as with scaling levels s=1,2 and 6. The total number of cycle cancelling operations obviously decreases with higher $s$. But more importantly the number of these operations on high resolution images gets significantly smaller. As detecting negative cycles is on average more expensive in the larger image versions, reducing their number has a large impact on the runtime. In this example we could decrease the runtime from 77m33s for s=0 to 3m36s (of which 3m17s were spend on the highest resolution) for s=6.

## References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1 edition, Feb. 1993.

[2] B. Cherkassky and A. Goldberg. Negative-cycle detection algorithms. In J. Diaz and M. Serna, editors, *Algorithms ESA '96*, volume 1136 of *Lecture Notes in Computer Science*, pages 349–363. Springer Berlin Heidelberg, 1996.

# Two-Sided Boundary Labeling with Adjacent Sides

Philipp Kindermann[*] Benjamin Niedermann[†] Ignaz Rutter[‡] Marcus Schaefer[‡] André Schulz[§]
Alexander Wolff[†]

## Abstract

In the *Boundary Labeling* problem, we are given a set of $n$ points, referred to as *sites*, inside an axis-parallel rectangle $R$, and a set of $n$ pairwise disjoint rectangular labels that are attached to $R$ from the outside. The task is to connect the sites to the labels by non-intersecting polygonal paths, so-called *leaders*.

In this paper, we study the *Two-Sided Boundary Labeling with Adjacent Sides* problem, with labels lying on two adjacent sides of the enclosing rectangle. We restrict ourselves to rectilinear leaders with at most one bend. We present a polynomial-time algorithm that computes a crossing-free leader layout if one exists. So far, such an algorithm has only been known for the simpler cases that labels lie on one side or on two opposite sides of $R$ (where a crossing-free solution always exists).

## 1 Introduction

Label placement is an important problem in cartography and, more generally, information visualization. Features such as points, lines, and regions in maps, diagrams, and technical drawings often have to be labeled so that users understand better what they see. The general label-placement problem is NP-hard [6], which explains why labeling a map manually is a tedious task that has been estimated to take 50% of total map production time [5].

Boundary labeling can be seen as a graph-drawing problem where the class of graphs to be drawn is restricted to matchings.

**Problem statement.** Following Bekos et al. [2], we define the Boundary Labeling problem as follows. We are given an axis-parallel rectangle $R = [0, W] \times [0, H]$, which is called the *enclosing rectangle*, a set $P = \{p_1, \ldots, p_n\} \subset R$ of $n$ points, called *sites*, within the rectangle $R$, and a set $L$ of $m \leq n$ axis-parallel rectangles $\ell_1, \ldots, \ell_m$, called *labels*, that lie in the complement of $R$ and touch the boundary of $R$. No two labels overlap. We denote an instance of the problem by the triplet $(R, L, P)$. A *solution* to the problem is a set of $m$ curves $c_1, \ldots, c_m$, called *leaders*, that connect sites to labels such that the leaders a) produce a matching between the labels and (a subset of) the sites, b) are contained inside $R$, and c) touch the associated labels on the boundary of $R$.

A solution is *planar* if the leaders do not intersect. Note that we do not prescribe which site connects to which label. The endpoint of a leader at a label is called a *port*. We distinguish two incarnations of the Boundary Labeling problem: either the position of the ports on the boundary of $R$ is fixed and part of the input, or the ports *slide*, i.e., their exact location is not prescribed.

We restrict our solutions to *po-leaders*, that is, starting at a site, the first line segment of a leader is parallel ($p$) to the side of $R$ containing the label it leads to, and the second line segment is orthogonal ($o$) to that side. Bekos et al. [1, Fig. 12] observed that not every instance (with $m = n$) admits a planar solution with *po*-leaders where all sites are labeled.

**Previous work.** For *po*-labeling, Bekos et al. [2] gave a simple quadratic-time algorithm for the one-sided case that, in a first pass, produces a labeling of minimum total leader length by matching sites and ports from bottom to top. In a second pass, their algorithm removes all intersections without increasing the total leader length. This result was improved by Benkert et al. [3] who gave an $O(n \log n)$-time algorithm for the same objective function and an $O(n^3)$-time algorithm for a very general class of objective functions, including, for example, bend minimization. They extend the latter result to the two-sided case (with labels on opposite sides of $R$), resulting in an $O(n^8)$-time algorithm. For the special case of leader-length minimization, Bekos et al. [2] gave a simple dynamic program running in $O(n^2)$ time. All these algorithms work both for fixed and sliding ports.

**Our contribution.** We investigate the problem Two-Sided Boundary Labeling with Adjacent Sides where all labels lie on two *adjacent* sides of $R$,

[*]Lehrstuhl für Informatik I, Universität Würzburg, Germany. WWW: www1.informatik.uni-wuerzburg.de/en/staff
[†]Fakultät für Informatik, Karlsruher Institut für Technologie (KIT), Germany. Email: {rutter, benjamin.niedermann}@kit.edu
[‡]College of Computing and Digital Media, DePaul University, Chicago, IL, USA. Email: mschaefer@cs.depaul.edu
[§]Institut für Mathematische Logik und Grundlagenforschung, Universität Münster, Germany. Email: andre.schulz@uni-muenster.de

for example, on the top and right side. Note that point data often comes in a coordinate system; then it is natural to have labels on adjacent sides (for example, opposite the coordinate axes). We argue that this problem is more difficult than the case where labels lie on opposite sides, which has been studied before: with labels on opposite sides, (a) there is always a solution where all sites are labeled (if $m = n$) and (b) a feasible solution can be obtained by considering two instances of the one-sided case.

Our result is an algorithm that, given an instance with $n$ labels and $n$ sites, decides whether a planar solution exists where all sites are labeled and, if yes, computes a layout of the leaders (see Section 3). We use dynamic programming to "guess" a partition of the sites into the two sets that are connected to the leaders on the top side and on the right side. The algorithm runs in $O(n^2)$ time and uses $O(n)$ space.

**Notation.** We call the labels that lie on the right (top) side of $R$ *right (top) labels.* The *type* of a label refers to the side of $R$ on which it is located. The *type* of a leader (or a site) is simply the type of its label. We assume that no two sites lie on the same horizontal or vertical line, and no site lies on a horizontal or vertical line through a port or an edge of a label.

For a solution $\mathcal{L}$ of a boundary labeling problem, we define the total length of all leaders in $\mathcal{L}$ by $\text{length}(\mathcal{L})$.

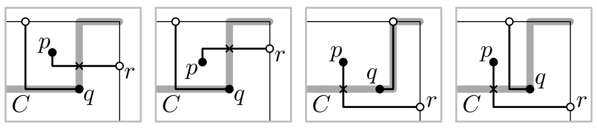
## 2 Structure of Planar Solutions

In this section, we attack our problem presenting a series of structural results of increasing strength. For simplicity, we assume fixed ports. For sliding ports, we can simply fix all ports to the bottom-left corner of their corresponding labels. First we show that we can split a planar two-sided solution into two one-sided solutions by constructing an $xy$-monotone, rectilinear curve from the top-right to the bottom-left corner of $R$. Afterwards, we provide a necessary and sufficient criterion to decide whether for a given separation there exists a planar solution. This will form the basis of our dynamic programming algorithm, which we present in the next section.

**Definition 1** *We call an $xy$-monotone, rectilinear curve connecting the top-right to the bottom-left corner of $R$ an* $xy$-separating curve. *We say that a planar solution to* Two-Sided Boundary Labeling with Adjacent Sides *is* $xy$-separated *if and only if there exists an $xy$-separating curve $C$ such that*
*a) the top sites and all their leaders lie on or above $C$*
*b) the right sites and all their leaders lie below $C$.*

It is not hard to see that a planar solution is not $xy$-separated if there exists a site $p$ that is labeled to the right side and a site $q$ that is labeled to the top side with $x(p) < x(q)$ and $y(p) > y(q)$. There are exactly four patterns in a possible planar solution that satisfy this condition. Moreover, these patterns are the only ones that can violate $xy$-separability.

**Lemma 1** *A planar solution is $xy$-separated iff it does not contain any of the following patterns P1–P4*



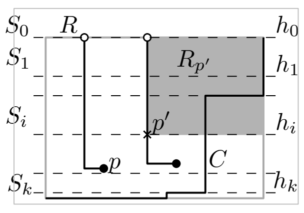
(P1) (P2) (P3) (P4)

Next, we claim that any planar solution can be transformed into an $xy$-separated planar solution. Our proof shows that each of the four patterns of Lemma 1 can be resolved by rerouting leaders such that no crossings arise and the leader length decreases. We cannot present the proof due to space constraints.

**Proposition 1** *If there exists a planar solution $\mathcal{L}$ to* Two-Sided Boundary Labeling with Adjacent Sides, *then there exists an $xy$-separated planar solution $\mathcal{L}'$ with* $\text{length}(\mathcal{L}') \leq \text{length}(\mathcal{L})$.

Since every solvable instance of Two-Sided Boundary Labeling with Adjacent Sides admits an $xy$-separated planar solution, it suffices to search for such a solution. Moreover, an $xy$-separated planar solution that minimizes the total leader length is a solution of minimum length. In Lemma 2 we provide a necessary and sufficient criterion to decide whether, for a given $xy$-monotone curve $C$, there is a planar solution that is separated by $C$. We denote the region of $R$ above $C$ by $R_{\text{T}}$ and the region of $R$ below $C$ by $R_{\text{R}}$. For each horizontal segment of $C$ consider the horizontal line through the segment. We denote the parts of these lines within $R$ by $h_1, \ldots, h_k$, respectively. Further let $h_0$ be the top edge of $R$. The line segments $h_1, \ldots, h_k$ partition $R_{\text{T}}$ into $k$ strips, which we denote by $S_1, \ldots, S_k$ from top to bottom, such that each strip $S_i$ is bounded by $h_i$ from below for $i = 1, \ldots, k$; see Fig. 1a. Additionally, we define $S_0$ to be the empty strip that coincides with $h_0$. Note that this strip cannot contain any site of $P$. For any point $p$ on one of the horizontal lines $h_i$, we define the rectangle $R_p$, spanned by the top-right corner of $R$ and $p$. We define $R_p$ such that it is closed but does *not* contain its top-left corner. In particular, we consider the port of a top label as contained in $R_p$, only if it is not the upper left corner.

A rectangle $R_p$ is *valid* if the number of sites of $P$ above $C$ that belong to $R_p$ is at least as large as the number of ports on the top side of $R_p$. The central idea is that the sites of $P$ inside a valid rectangle $R_p$ can be connected to labels on the top side of the valid


(a)

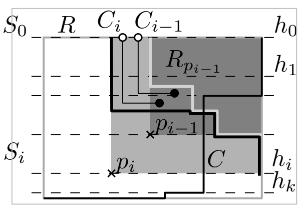

(b)

Fig. 1: The strip condition. a) The horizontal segments of $C$ partition the strips $S_0, S_1, \ldots, S_k$. b) Constructing a planar labeling from a sequence of valid rectangles.

rectangle by leaders that are completely contained inside the rectangle. We are now ready to present the *strip condition.*

**Condition 1** *The* strip condition *of strip $S_i$ is satisfied if there exists a point $p_i \in h_i \cap R_\mathrm{T}$, such that $R_{p_i}$ is valid.*

We now prove that, for a given $xy$-montone curve $C$ going from the top-right corner to the bottom-left corner of $R$, there exists a planar solution in $R_\mathrm{T}$ for the top labels if and only if $C$ satisfies the strip condition for all strips $S_0, \ldots, S_k$ in $R_\mathrm{T}$.

**Lemma 2** *Let $C$ be an $xy$-monotone curve from the top-right corner of $R$ to the bottom-left corner of $R$. Let $P' \subseteq P$ be the sites that are in $R_\mathrm{T}$. There is a planar solution that uses all top labels of $R$ to label sites in $P'$ in such a way that all leaders are in $R_\mathrm{T}$ if and only if each horizontal strip $S_i$, as defined above, satisfies the strip condition.*

**Proof.** To show that the conditions are necessary, let $\mathcal{L}$ be a planar solution for which all top leaders are above $C$. Consider strip $S_i$, which is bounded from below by line $h_i, 0 \leq i \leq k$. If there is no site of $P'$ below $h_i$, rectangle $R_p$ is clearly valid, where $p$ is the intersection of $h_i$ with the left side of $R$ , and thus the strip condition is satisfied. Hence, assume that there is a site $p \in P'$ that is labeled by a top label, and is in strip $S_j$ with $j > i$; see Fig. 1a. Then, the vertical segment of this leader crosses $h_i$ in $R_\mathrm{T}$. Let $p'$ denote the rightmost such crossing of a leader of a site in $P'$ with $h_i$. We claim that $R_{p'}$ is valid. To see this, observe that all sites of $P'$ top-right of $p'$ are contained in $R_{p'}$. Since no leader may cross the vertical segments defining $p'$, the region $R_{p'} \cap R_\mathrm{T}$ contains the same number of sites as $R_{p'}$ contains ports on its top side, i.e., $R_{p'}$ is valid.

Conversely, we show that if the conditions are satisfied, then a corresponding planar solution exists. For $i = 0, \ldots, k-1$, let $p'_i$ denote the rightmost point of $h_i \cap R_\mathrm{T}$, such that $R_{p'_i}$ is valid. We define $p_i$ to be the point on $h_i \cap R_\mathrm{T}$, whose $x$-coordinate is $\min_{j \leq i}\{x(p'_j)\}$. Note that $R_{p_i}$ is a valid rectangle, as, by definition, it completely contains some valid rectangle $R_{p'_j}$ with $x(p'_j) = x(p_i)$. Also by definition the sequence formed by the points $p_i$ has decreasing $x$-coordinates, i.e., the $R_{p_i}$ grow to the left; see Fig. 1b.

We can prove inductively that, for each $i = 0, \ldots, k$, there is a planar labeling $\mathcal{L}_i$ that matches the labels on the top side of $R_{p_i}$ to sites contained in $R_{p_i}$, in such a way that there exists an $xy$-monotone curve $C_i$ from the upper-left corner of $R_{p_i}$ to its lower right corner that separates the labeled sites from the unlabeled sites without intersecting any leaders. Then $\mathcal{L}_k$ is the claimed labeling. □

A similar strip condition (with vertical strips) can be obtained for the right region $R_\mathrm{R}$ of a partitioned instance. The characterization is completely symmetric.

## 3 The Algorithm

Now we describe how to find an $xy$-monotone chain $C$ that satisfies the strip conditions. For that purpose we only consider $xy$-monotone chains that lie on the dual of the grid induced by the sites and ports of the given instance. When traversing this grid from grid point to grid point, we either pass a site (*site event*) or a port (*port event*). By passing a site, we decide if the site is connected to the top or to the right side. In the following, we describe a dynamic program that finds an $xy$-separating chain in $O(n^3)$ time.

Let there be $m_\mathrm{R}$ ports on the right side of $R$ and $m_\mathrm{T}$ ports on the top side of $R$, then the grid has size $[n + m_\mathrm{T} + 2] \times [n + m_\mathrm{R} + 2]$. We define the grid points as $G(x, y)$, with $G(0, 0)$ being the bottom-left and $r := G(n+m_\mathrm{T}+2, n+m_\mathrm{R}+2)$ being the top-right corner of $R$. Further, we define $G_x(s) := x(G(s, 0))$ and $G_y(t) := y(G(0, t))$.

An entry in the table of our dynamic program is described by three values. The first two values are $s$ and $t$, which give the position of the current search for the curve $C$. The interpretation is that the entry encodes the possible $xy$-monotone curves from $r$ to $p_C := G(s, t)$; see Fig. 2. The remaining value $u$ denotes the number of sites above $C$ in the rectangle spanned by $r$ and $p_C$. Note that it suffices to store $u$, as the number of sites below the curve $C$ can directly be derived from $u$ and all sites that are contained in the rectangle spanned by $r$ and $p_C$. We denote the first values describing the positions of the curves by the vec-

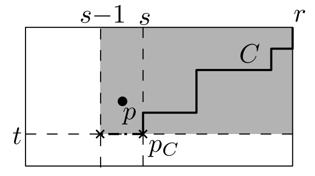


Fig. 2: Step of the dynamic program where $p$ enters the rectangle spanned by $r$ and $G(s-1, t)$.

tor $\mathbf{c} = (s,t)$. Our goal is to compute a table $T[\mathbf{c},u]$, such that $T[\mathbf{c},u] = \texttt{true}$ if and only if there exists an $xy$-monotone chain $C$, such that the following conditions hold. (i) Curve $C$ starts at $r$ and ends at $p_C$. (ii) Inside the rectangle spanned by $r$ and $p_C$, there are $u$ sites of $P$ above $C$. (iii) For each strip in the two regions $R_\mathrm{T}$ and $R_\mathrm{R}$ defined by $C$ the strip condition holds.

It follows from these conditions, Proposition 1 and Lemma 2 that the instance admits a planar solution if and only if $T[(0,0),u] = \texttt{true}$ for some $u$.

Let us now proceed to describe how to compute the table. Initially, we set $\mathbf{c} = (n + m_\mathrm{T} + 2, n + m_\mathrm{R} + 2)$. We initialize the first entry $T[\mathbf{c}, 0]$ with `true`. The remaining entries are initialized with `false`.

Let $\mathbf{c} := (s,t)$ be the current grid point we checked as endpoint for $C$. Based on the table $T[\mathbf{c},\cdot]$ we then compute the entries $T[\mathbf{c} - \mathbf{\Delta c},\cdot]$ where the vector $\mathbf{\Delta c} = (\Delta s, \Delta t)$ is chosen in such a way that exactly one of both entries $\Delta s, \Delta t \in \{0,1\}$ has value 1. We classify such steps, depending on whether we cross a site or a port. We give a full description for $\mathbf{\Delta c} = (1,0)$, i.e, we decrease $s$ by 1. The other case is completely symmetric. Assume $T[\mathbf{c},u] = \texttt{true}$. We distinguish two cases, based on whether we cross a site or a port.

**Case 1:** Going from $s$ to $s-1$ is a site event, i.e., there is a site $p$ with $G_x(s) > x(p) > G_x(s-1)$. Note that, by our general position assumption and by the definition of the coordinates, the site $p$ is unique. If $y(p) > G_y(t)$, then $p$ enters the rectangle spanned by $G(s-1,t)$ and $r$, and it is located above $C$. We thus set $T[\mathbf{c} - \mathbf{\Delta c}, u+1] = \texttt{true}$. Otherwise we set $T[\mathbf{c} - \mathbf{\Delta c}, u] = \texttt{true}$. Note that the strip conditions remain satisfied since we do not decrease the number of sites in any region.

**Case 2:** Going from $s$ to $s-1$ is a port event, i.e., there is a label $\ell$ on the top side, whose port is between $G_x(s-1)$ and $G_x(s)$. Thus, the region above $C$ contains one more label. We therefore check the strip condition for the strip above the horizontal line through $G(s-1,t)$. If it is satisfied, we set $T[\mathbf{c} - \mathbf{\Delta c}, u] = \texttt{true}$.

This immediately gives us a polynomial-time algorithm for Two-Sided Boundary Labeling with Adjacent Sides. The running time crucially relies on the number of strip conditions that need to be checked. We show that after a $O(n^2)$ preprocessing phase, such queries can be answered in $O(1)$ time.

To implement the test of the strip conditions, we use a table $B_\mathrm{T}$, which stores in the position $B_\mathrm{T}[s,t]$ how large a deficit of top sites to the right can be compensated by sites above and to the left of $G(s,t)$. To compute this matrix, we use a simple dynamic program, which calculates the entries of $B_\mathrm{T}$ by going from the left to the right side. Once we have computed this matrix, it is possible to query the strip condition in the dynamic program that computes $T$ in $O(1)$ time. The table can be clearly filled out in $O(n^2)$ time. A similar matrix $B_\mathrm{R}$ can be computed for the vertical strips. Altogether, this yields an algorithm for Two-Sided Boundary Labeling with Adjacent Sides that runs in $O(n^3)$ time and uses $O(n^3)$ space. However, the entries of each row and column of $T$ depend only on the previous row and column, which allows us to reduce the storage requirement to $O(n^2)$. Using Hirschberg's algorithm [4], we can still backtrack the dynamic program and find a solution corresponding to an entry in the last cell in the same running time.

**Theorem 3** Two-Sided Boundary Labeling with Adjacent Sides *can be solved in $O(n^3)$ time using $O(n^2)$ space.*

In order to increase the performance of our algorithm, we can reduce the dimension of the table $T$ by 1. For any search position $\mathbf{c}$, the possible values of $u$, for which $T[\mathbf{c},u] = \texttt{true}$ form an interval. Thus, we only need to store the boundaries of the $u$-interval. Further, we can compute the tables $B_\mathrm{T}$ and $B_\mathrm{R}$ backwards, i.e., in the direction of the dynamic program, by precomputing the entries of $B_\mathrm{T}$ and $B_\mathrm{R}$ on the top and right side. Using Hirschberg's algorithm, this reduces the running time to $O(n^2)$ and the space to $O(n)$.

**Theorem 4** Two-Sided Boundary Labeling with Adjacent Sides *can be solved in $O(n^2)$ time using $O(n)$ space.*

## References

[1] Michael A. Bekos, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Area-feature boundary labeling. *Comput. J.*, 53(6):827–841, 2010. 1

[2] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Comput. Geom. Theory Appl.*, 36(3):215–236, 2007. 1

[3] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009. 1

[4] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, 18(6):341–343, 1975. 4

[5] Joel L. Morrison. Computer technology and cartographic change. In D.R.F. Taylor, editor, *The Computer in Contemporary Cartography.* Johns Hopkins University Press, 1980. 1

[6] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13:21–47, 1999. 1

# Trajectory-Based Dynamic Map Labeling

Andreas Gemsa*† Benjamin Niedermann* Martin Nöllenburg*†

## Abstract

In this paper we introduce *trajectory-based labeling*, a new aspect of dynamic map labeling where a movement trajectory for the map viewport is given. We define a general labeling model and study the active range maximization problem in this model. The problem is $\mathcal{NP}$-complete and we present a practical ILP formulation for the general case. For the restricted case that no more than $k$ labels can be active at any time, we give a polynomial-time algorithm.

## 1 Introduction

In contrast to traditional static maps, dynamic digital maps support continuous movement of the map viewport based on panning, rotation, or zooming. In this paper, we take a trajectory-based view on map labeling. In many applications, e.g., car navigation, a movement trajectory is known in advance and it becomes interesting to optimize the visualization of the map locally along this trajectory.

Selecting and placing a maximum number of non-overlapping labels for various map features is an important cartographic problem. Labels are usually modeled as rectangles and the objective in a static map is to find a maximum (possibly weighted) independent set of labels. This is known as $\mathcal{NP}$-complete [6] and there are approximation algorithms and PTAS's in different labeling models, e.g. [1, 5].

With the increasing popularity of interactive dynamic maps, e.g., as digital globes or on mobile devices, the static labeling problem has been translated into a dynamic setting. Due to the temporal dimension of the animations occurring during map movement, it is necessary to define a notion of temporal consistency or coherence as to avoid distracting effects such as jumping or flickering labels [2]. Previously, consistent labeling has been studied from a global perspective under continuous zooming [3] and continuous rotation [7]. In practice, however, an individual map user is typically interested only in a specific part of a map and it is thus often more important to optimize the labeling locally for a certain trajectory of the map viewport than globally for the whole map.

*Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany. Email: {lastname}@kit.edu

†received financial support by the Concept for the Future of KIT within the framework of the German Excellence Initiative.

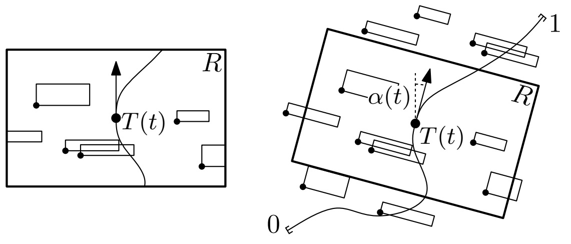


Figure 1: Illustration of the viewport moving along a trajectory. Left the user's view and right a general view of the map and the viewport.

We introduce a new and versatile trajectory-based model for consistent map labeling, which we apply to point feature labeling for a viewport that moves and rotates along a differentiable trajectory in a fixed-scale base map in a forward-facing way. It is no surprise that maximizing the number of visible labels integrated over time in our model is $\mathcal{NP}$-complete. We present a practical ILP model for the general unrestricted case and a polynomial-time algorithm for the restricted case that no more than $k$ labels are active at any time for some constant $k$. We note that limiting the number of active labels is of interest in particular for dynamic maps on small-screen devices.

## 2 Trajectory-Based Labeling Model

Let $M$ be a labeled north-facing and fixed-scale map, i.e., a set of points $P = \{p_1, \dots, p_n\}$ in the plane together with a corresponding set $L = \{\ell_1, \dots, \ell_n\}$ of labels. Each label $\ell_i$ is represented by an axis-aligned rectangle of individual width and height. We call the point $p_i$ the *anchor* of the label $\ell_i$. Here we assume that each label has an arbitrary but fixed position relative to its anchor, e.g., with its lower left corner coinciding with the anchor. The *viewport* $R$ is an arbitrarily oriented rectangle of fixed size that defines the currently visible part of $M$ on the map screen. The viewport follows a trajectory that is given by a continuous differentiable function $T\colon [0,1] \to \mathbb{R}^2$. For an example see Figure 1. More precisely, we describe the viewport by a function $V\colon [0,1] \to \mathbb{R}^2 \times [0, 2\pi]$. The interpretation of $V(t) = (c, \alpha)$ is that at time $t$ the center of the rectangle $R$ is located at $c$ and $R$ is rotated clockwise by the angle $\alpha$. Since $R$ moves along $T$ we define $V(t) = (T(t), \alpha(t))$, where $\alpha(t)$ denotes the direction of $T$ at time $t$. We sometimes refer to $R$ at time $t$ as $V(t)$. To ensure good readability, we require that the labels are always aligned with the viewport axes as the viewport changes its orientation, i.e., they rotate around their anchors by the same angle $\alpha(t)$, see Figure 1. We denote the rotated label rectangle of $\ell$ at time $t$ by $\ell(t)$.

We say that a label $\ell$ is *present* at time $t$, if $V(t) \cap \ell(t) \neq \emptyset$. As we consider the rectangles $\ell(t)$ and $V(t)$ to be closed, we can describe the points in time for which $\ell$ is present by closed intervals. We define for each label $\ell$ the set $\Psi_\ell$ that describes all disjoint subintervals of $[0,1]$ for which $\ell$ is present, thus $\Psi_\ell = \{[a,b] \mid [a,b] \subseteq [0,1]$ is maximal so that $\ell$ is present at all $t \in [a,b]\}$. Further, we define the disjoint union $\Psi = \{([a,b],\ell) \mid [a,b] \in \Psi_\ell$ and $\ell \in L\}$ of all $\Psi_\ell$. We abbreviate $([a,b],\ell) \in \Psi$ by $[a,b]_\ell$ and call $[a,b]_\ell \in \Psi$ a *presence interval* of $\ell$.

Two labels $\ell$ and $\ell'$ are in *conflict* with each other at time $t$ if $\ell(t) \cap \ell'(t) \neq \emptyset$. If $\ell(t) \cap \ell'(t) \cap V(t) \neq \emptyset$ we say that the conflict is *present*. As in [7] we can describe the occurrences of conflicts between two labels $\ell, \ell' \in L$ by a set of closed intervals: $C_{\ell,\ell'} = \{[a,b] \subseteq [0,1] \mid [a,b]$ is maximal and $\ell$ and $\ell'$ are in conflict at all $t \in [a,b]\}$. We define the disjoint union $C = \{([a,b],\ell,\ell') \mid [a,b] \in C_{\ell,\ell'}$ and $\ell,\ell' \in L\}$ of all $C_{\ell,\ell'}$. We abbreviate $([a,b],\ell,\ell') \in C$ as $[a,b]_{\ell,\ell'}$ and call it a *conflict interval* of $\ell$ and $\ell'$.

The tuple $(P, L, \Psi, C)$ is called an *instance* of trajectory-based labeling. Note that the essential information of $T$ is implicitly given by $\Psi$ and $C$. If we assume that the trajectory $T$ is a continuous, differentiable chain of $m$ circular arcs we can compute $\Psi$ in $O(m \cdot n)$ time and $C$ in $O(n^2)$ time using simple geometric observations. We omit the details due to space restrictions.

Next we define the *activity* of labels, i.e., when to actually display which of the present labels on screen. We restrict ourselves to closed and disjoint intervals describing the activity of a label $\ell$ and define the set $\Phi_\ell = \{[a,b] \subseteq [0,1] \mid \ell$ is active at all $t \in [a,b]\}$, as well as the disjoint union $\Phi = \{([a,b],\ell) \mid [a,b] \in \Phi_\ell$ and $\ell \in L\}$ of all $\Phi_\ell$. We abbreviate $([a,b],\ell) \in \Phi$ with $[a,b]_\ell$ and call $[a,b]_\ell \in \Phi$ an *active interval* of $\ell$.

It remains to define an *activity model* restricting $\Phi$ in order to obtain a reasonable labeling without label overlaps or inconsistent dynamic behavior like label flickering. Here we propose three activity models AM1, AM2, AM3 with increasing flexibility. All three activity models guarantee consistency and share the following three restrictions: (1) a label can only be active at time $t$ if it is present at time $t$, (2) to avoid flickering each presence interval contains at most one active interval, and (3) if two labels are in conflict at a time $t$, then at most one of them may be active at time $t$ to avoid overlapping labels.

What distinguishes the three models are the possible points in time when labels can become active or

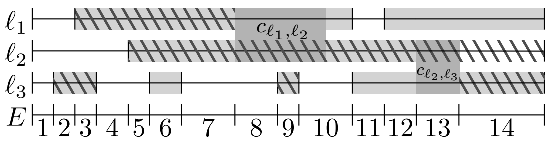


Figure 2: The light gray intervals show presence intervals, the hatched intervals active intervals and the dark gray intervals conflicts between labels.

inactive. The first and most restrictive activity model AM1 demands that each label $\ell$ is either active for a complete presence interval $[a,b]_\ell \in \Psi$ or never active in $[a,b]_\ell$. The second activity model AM2 allows an active interval of a label $\ell$ to end earlier than the corresponding presence intervals if there is a *witness label* $\ell'$ for that, i.e., an active interval for $\ell$ may end at time $c$ if there is a starting conflict interval $[c,d]_{\ell,\ell'}$ and the conflicting label $\ell'$ is active at $c$. However, AM2 still requires every active interval to begin with the corresponding presence interval. The third activity model AM3 extends AM2 by also relaxing the restriction regarding the start of active intervals. An active interval for a label $\ell$ may start at time $c$ if a present conflict $[a,c]_{\ell,\ell'}$ involving $\ell$ and an active witness label $\ell'$ ends at time $c$. In this model active intervals may begin later and end earlier than their corresponding presence intervals if there is a visible reason for the map user to do so, namely the start or end of a conflict with an active witness label.

A common objective in both static and dynamic map labeling is to maximize the number of labeled points. In our setting, we want to maximize $\sum_{[a,b]_\ell \in \Phi} w([a,b]_\ell)$, where $w([a,b]_\ell)$ is the weight of the active interval $[a,b]_\ell$, e.g., $w([a,b]_\ell) = b - a$. This corresponds to displaying a maximum number of labels integrated over the time interval $[0,1]$. Given an instance $(P, L, \Psi, C)$, then with respect to one of the three activity models we want to find an activity $\Phi$ that maximizes $\sum_{[a,b]_\ell \in \Phi} w([a,b]_\ell)$; we call this optimization problem GeneralMaxTotal. If we require, that at any time $t$ at most $k$ labels are active for a constant $k$, we call it $k$-RestrictedMaxTotal.

## 3 Solving GeneralMaxTotal

We observe that GeneralMaxTotal is $\mathcal{NP}$-complete by a straight-forward reduction from the $\mathcal{NP}$-complete maximum independent set of rectangles problem [6]. We simply interpret the set of rectangles as a set of labels, choose a short vertical trajectory $T$ and a viewport $R$ that always contains all labels.

**Theorem 1** GeneralMaxTotal *is $\mathcal{NP}$-complete.*

Since we are still interested in finding an optimal solution for GeneralMaxTotal we have developed integer linear programming (ILP) formulations for all three activity models. Due to space constraints we only present the ILP for AM3. It is then straightforward to adapt this ILP for AM1 and AM2.

We define $E$ to be the totally ordered set of the interval boundaries of all presence and all conflict intervals and additionally include 0 and 1; see Figure 2. We call each interval $[c, d]$ between two consecutive elements $c$ and $d$ in $E$ an *atomic segment* and denote the $i$-th atomic segment of $E$ by $E(i)$.

For each label $\ell$ and for each $E(i)$ we introduce three binary variables $x_i^\ell$, $b_i^\ell$ and $e_i^\ell$. If the label is active during $E(i)$ the variable $x_i^\ell$ is set to 1, otherwise $x_i^\ell = 0$. The variable $b_i^\ell$ is set to 1 if and only if $E(i)$ is the first atomic segment of an active interval of $\ell$, and analogously $e_i^\ell = 1$ if and only if $E(i)$ is the last atomic segment of an active interval of $\ell$.

Now we are ready to introduce the constraints of the ILP. In order to ensure property (1) we set (A) $b_i^\ell = e_i^\ell = x_i^\ell = 0$ for all segments $E(i)$ that are not contained in any presence interval of $\ell$. To ensure property (2), we require for every presence interval $[a, b]_\ell \in \Psi$ that (B) $\sum_{j\in J} b_j^\ell \le 1$ and $\sum_{j\in J} e_j^\ell \le 1$, where $J = \{j \mid E(j) \subseteq [a, b]\}$. Next, for every conflict interval $[c, d]_{\ell,\ell'} \in C$ and every $E(i) \subseteq [c, d]$ we require (C) $x_i^\ell + x_i^{\ell'} \le 1$ so that at most one of both labels can be active during segment $E(i)$ (property (3)).

The desired meaning of the variables $x_i^\ell$, $b_i^\ell$ and $e_i^\ell$ is obtained as follows. For all labels $\ell \in L$ we introduce the constraint $b_1^\ell = x_1^\ell$ for the first segment $E(1)$, and for the remaining segments $E(i)$, $i > 1$, we introduce the constraints (D) $x_{i-1}^\ell + b_i^\ell = x_i^\ell + e_{i-1}^\ell$. This means that if $\ell$ is active during $E(i-1)$ ($x_{i-1}^\ell = 1$), then it must either be active during $E(i)$ ($x_i^\ell = 1$) or the active interval ends with $E(i-1)$ ($e_{i-1}^\ell = 1$), and if $\ell$ is active during $E(i)$ ($x_i^\ell = 1$) then it must be active during $E(i-1)$ ($x_{i-1}^\ell = 1$) or the active interval begins with $E(i)$ ($b_i^\ell = 1$).

To model the admissible boundaries of the active intervals according to AM3 we consider for all segments $E(j) \subset [a, b]_\ell$ which are neither the first nor the last atomic segments in $[a, b]_\ell$ the following two constraints. Let $c_{\ell,\ell_1}, \dots, c_{\ell,\ell_k} \in C$ be all conflicts of $\ell$ that contain $E(j-1)$ but not $E(j)$, i.e., the conflicts end with $E(j-1)$. We require (E) $b_j^\ell \le \sum_{i=1}^k x_{j-1}^{\ell_i}$, where $b_j^\ell = 0$ if no such conflict exists. Condition (E) ensures that at least one label of $\ell_1, \dots, \ell_k$ is active during $E(j-1)$ if $\ell$ becomes active with $E(j)$. Analogously, let $c_{\ell,\ell_1}, \dots, c_{\ell,\ell_k} \in C$ be all conflicts of $\ell$ that contain $E(j+1)$ but not $E(j)$, i.e., the conflicts begin with $E(j+1)$. We require (F) $e_j^\ell \le \sum_{i=1}^k x_{j+1}^{\ell_i}$, where $e_j^\ell = 0$ if no such conflict exists.

The objective function is $\sum_{\ell\in L} \sum_{i=1}^{|E|-1} x_i^\ell \cdot w(E(i))$.

Obviously, $|E| \le |\Psi| + |C| + 2$. We obtain $O((|\Psi| + |C|) \cdot |L|)$ variables and $O((|\Psi| + |C|) \cdot |L|)$ constraints.

We have evaluated all three models on the street map of the city center of Karlsruhe, Germany with over 2000 labels. Using randomly chosen trajectories based on shortest paths in the road network we could show that the ILPs provide practically useful running times. Using a map scale of 1:2000 we could compute $\Phi$ in less than a second for a vast majority of trajectories. For a scale of 1:4000, the running times for AM1-3 are below five seconds, again excluding a few outliers.

## 4 Solving $k$-RestrictedMaxTotal

In this section we show that the problem $k$-RestrictedMaxTotal can be solved in polynomial time. Due to space constraints we only give a description of our algorithms for AM1; they can be extended to AM2 by applying some non-trivial modifications. Solving $k$-RestrictedMaxTotal is related to finding a maximum cardinality $k$-colorable subset of $n$ intervals in interval graphs. This can be done in polynomial time in both $n$ and $k$ [4]. However, we have to consider additional constraints due to conflicts between labels, which makes our problem more difficult. In the following, we first give algorithms that solve $k$-RestrictedMaxTotal for $k = 1$, and $k = 2$, and then sketch how to extend this to any constant $k > 2$.

Note that for the case that at most one label can be active ($k = 1$) conflicts between labels do not matter. Thus, it is sufficient to find an independent subset of $\Psi$ of maximum weight. This is equivalent to finding a maximum weight independent set on interval graphs, which can be done in $O(|\Psi|)$ time [8], given that the intervals are sorted. We denote this algorithm by $\mathcal{A}_1$.

Before we give our polynomial-time algorithm that solves 2-RestrictedMaxTotal we require some definitions. We say two presence intervals $[a, b]_\ell$ and $[c, d]_{\ell'}$ *are in conflict* if there is a conflict $[f, g]_{\ell,\ell'} \in C$ with $[f, g]_{\ell,\ell'} \cap [a, b]_\ell \cap [c, d]_{\ell'} \ne \emptyset$. We assume that the intervals of $\Psi = \{I_1, \dots, I_m\}$ are sorted in non-decreasing order by their left endpoints. We call a tuple of two presence intervals $(I_i, I_j)$, $I_i, I_j \in \Psi$ a *separating pair* if $i < j$, $I_i$ and $I_j$ overlap and are not in conflict with each other. Further, a separating pair $(I_p, I_q)$ is smaller than another separating pair $(I_i, I_j)$ if and only if $p < i$ or $p = i, q < j$. We denote the set of all separating pairs by $S$.

We observe that a separating pair $(I_i, I_j)$ contained in any solution of 2-RestrictedMaxTotal splits the set of presence intervals into two independent subsets. Specifically, a left (right) subset $L[i, j]$ ($R[i, j]$) that contains only intervals which lie completely to the left (right) of the intersection of $I_i$ and $I_j$ and are neither in conflict with $I_i$ nor $I_j$; see Figure 3.

We are now ready to describe our dynamic programming algorithm. For ease of notation we add two dummy separating pairs to $\Psi$. One pair $(I_{-1}, I_0)$ with presence intervals strictly to the left and one pair $(I_{m+1}, I_{m+2})$ with presence intervals

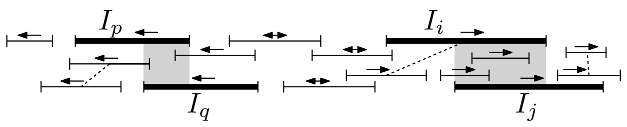


Figure 3: Illustration of presence intervals. Both $(I_i, I_j)$ and $(I_p, I_q)$ are separating pairs. The intervals of $L[i,j]$ ($R[p,q]$) are marked by a left (right) arrow. Intervals in conflict are connected by a dotted line.

strictly to the right of $[0,1]$. Note that since all original presence intervals are completely contained in $[0,1]$ every optimal solution contains both dummy separating pairs. Our algorithm computes a two-dimensional table $\mathcal{T}$, where for each separating pair $(I_i, I_j)$ there is an entry $\mathcal{T}[i,j]$ that stores the value of the optimal solution for $L[i,j]$. We compute $\mathcal{T}$ from left to right starting with the dummy separating pair $(I_{-1}, I_0)$ and initialize $\mathcal{T}[-1,0] = 0$. Then, we recursively define for every separating pair $(I_i, I_j) \in S$: $\mathcal{T}[i,j] = \max_{p,q}\{T[p,q] + w(I_p) + w(I_q) + \mathcal{A}_1(R[p,q] \cap L[i,j]) \mid p < q < j, (I_p, I_q) \in S$, the intervals $I_p, I_q, I_i,$ and $I_j$ are not in conflict$\}$. For a fixed separating pair $(I_i, I_j)$ our algorithm considers all possible directly preceding separating pairs, including the one in an optimal solution. For the intervals between both separating pairs $\mathcal{A}_1$ computes the optimal solution while a simple table lookup is sufficient to obtain the optimal solution for the remaining presence intervals.

By construction, the optimal solution to 2-RestrictedMaxTotal is stored in $\mathcal{T}[m+1, m+2]$. There are at most $O(m^2)$ separating pairs. Hence, for a single entry in $\mathcal{T}$ we need to execute the linear time algorithm $\mathcal{A}_1$ at most $O(m^2)$ times. The number of entries in $\mathcal{T}$ is also bounded by the number of separating pairs. Hence our algorithm, which we denote by $\mathcal{A}_2$, has time complexity $O(m^5)$.

We prove the correctness of the algorithm by contradiction. Assume that there exists an instance for which our algorithm does not compute the optimal solution correctly. That means, that there is a smallest separating pair $(I_i, I_j)$ for which the entry in $\mathcal{T}[i,j]$ does not contain the optimal solution for $L[i,j]$. Note that $(I_i, I_j)$ cannot be the dummy separating pair $(I_{-1}, I_0)$ since $\mathcal{T}[-1,0]$ is trivially correct. Hence, the optimal solution for $L[i,j]$ contains a separating pair directly preceding $(I_i, I_j)$, which we denote by $(I_p, I_q)$. Since there is no separating pair between $(I_i, I_j)$ and $(I_p, I_q)$ is in the optimal solution we can obtain the optimal solution for the presence intervals between both separating pairs by computing $\mathcal{A}_1(R[p,q] \cap L[i,j])$. Since, by assumption, $T[p,q]$ is correct, $\mathcal{A}_1$ is optimal, and our algorithm explicitly considers all preceding separating pairs, the entry $\mathcal{T}[i,j]$ must contain the optimal solution for $L[i,j]$. Thus, the correctness of our algorithm follows.

**Theorem 2** *Our algorithm solves* 2-RestrictedMaxTotal *in $O(m^5)$ time and requires $O(m^2)$ space.*

This approach can be generalized to solve $k$-RestrictedMaxTotal for any $k > 2$. We naturally extend the definition of separating pairs to separating $k$-tuples. Now, we can recursively define an algorithm $\mathcal{A}_k$ that solves $k$-RestrictedMaxTotal. We use essentially the same dynamic program as before, only that we replace algorithm $\mathcal{A}_1$ in the recurrence relation by $\mathcal{A}_{k-1}$. Note that $\mathcal{A}_k$ is a polynomial-time algorithm since we need to execute a polynomial-time algorithm a polynomial number of times. Due to space constraints we omit further details.

**Theorem 3** *Our algorithm solves $k$*-RestrictedMaxTotal *in AM1 in polynomial time and space.*

We can use the same basic idea of the dynamic program to solve $k$-RestrictedMaxTotal in AM2, but need to add for each presence interval, all possible subintervals to $\Psi$ that might be contained in an optimal solution. Moreover special care needs to be taken to ensure the witness condition of AM2 for all truncated intervals. It remains open whether $k$-RestrictedMaxTotal in AM3 can be solved in polynomial time.

## References

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory & Appl.*, 11(3-4):209–218, 1998.

[2] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Trans. Visualization and Computer Graphics*, 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. Theory & Appl.*, 43(3):312–328, 2010.

[4] M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discr. Appl. Math.*, 59(3):225–235, 1995.

[5] P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *Proc. ACM-SIAM Symp. Discr. Algorithms (SODA'09)*, pages 892–901, 2009.

[6] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[7] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In *Proc. 12th Int. Symp. Algorithms & Data Structures (WADS'11)*, volume 6844 of *LNCS*, pages 451–462. Springer-Verlag, 2011.

[8] J. Y. Hsiao, C. Y. Tang, and R. S. Chang. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Inform. Process. Lett.*, 43(5):229 – 235, 1992.

# Dynamic point labeling is strongly PSPACE-hard

Kevin Buchin* Dirk H.P. Gerrits*

## Abstract

An important but strongly NP-hard problem in automated cartography is how to best place textual labels for point features on a static map. We examine the complexity of various generalizations of this problem for dynamic and/or interactive maps. Specifically, we show that it is strongly PSPACE-hard to obtain a smooth dynamic labeling (function from time to static labelings) when the points move, when points are added and removed, or when the user pans, rotates, and/or zooms their view of the points.

## 1 Introduction

Map labeling involves associating textual *labels* with certain *features* on a map such as cities (points), roads (polylines), and lakes (polygons). This task takes considerable time to do manually, and for some applications *cannot* be done manually beforehand. In air traffic control, for example, a set of moving points (airplanes) has to be labeled at all times. In interactive maps users may pan, rotate, and/or zoom their view of the map, which may require relabeling.

**Static point labeling.** A good labeling for a point set has legible labels, and an unambiguous association between the labels and the points. This has been formalized by regarding the labels as axis-aligned rectangles slightly larger than the text they contain, which must be placed without overlap so that each contains the point it labels on its boundary. Not all placements are equally desirable, and as such various *label models* have been proposed, which specify the subset of allowed positions for the labels. In the *fixed-position models*, every point has a finite number of label candidates. In particular, in the 1-, 2-, and 4-position models a subset of 1, 2, or 4 corners is designated (the same subset for all labels) and each label must have one of these corners coincide with the point it labels. The *slider models* generalize this. In the 1-slider models one side of each label is designated, but the label may contain its point anywhere on this side. In the 2-slider models there is a choice between two opposite sides of the label, and in the 4-slider model the label can have the point anywhere on its boundary.

Ideally, one would like to label all points with pairwise non-intersecting labels. This is not always possible, however, and the decision problem is strongly NP-complete for the 4-position [2] and 4-slider [5] models, even if all labels are unit squares. We may deal with this difficulty in several ways. Firstly, we may reduce the font size, that is, shrink labels. The *size-maximization problem* asks to label all points with pairwise non-intersecting labels of maximal size. Secondly, we may remove labels. The *(weighted) number-maximization problem* asks to label a maximum-cardinality (maximum-weight) subset of the points with pairwise non-intersecting labels of given dimensions. Thirdly, we may allow labels to overlap, but try to keep such occurrences to a minimum. The *free-label-maximization problem* asks for all points to be labeled with labels of given dimensions, maximizing the number of non-intersecting labels. All of these problems are strongly NP-hard for the 4-position and 4-slider models because of the above result.

**Dynamic point labeling.** A natural generalization of static point labeling is dynamic point labeling. Here the point set $P$ changes over time, by points being added and removed, and/or by points moving continuously. We then seek a *dynamic labeling* $\mathcal{L}$, which for all $t$ assigns a static labeling $\mathcal{L}(t)$ to the static points $P(t)$ present at time $t$. For the 4-slider model we require that $\mathcal{L}$ is continuous in the sense that, if a point $p$ has a label over a non-empty time interval, then the label must move continuously over that interval. For the fixed-position and 2-slider models we must allow labels to make "jumps". We only allow $p$'s label to jump from position $A$ to position $B$, however, if there is no candidate position $C$ in between $A$ and $B$ in clockwise (or counter-clockwise) order around $p$. Thus, we allow horizontal and vertical (but no diagonal) jumps for the 4-position model. For a 2-slider model we only allow jumps from an endpoint of one slider to the "same" endpoint of the other slider.

For static point labeling, both practical heuristic algorithms and theoretical algorithms with guaranteed approximation ratios abound. Dynamic point labeling, however, has seen very few theoretical results. Been et al. [1] studied unweighted number-maximization for points under continuous zooming, giving constant-factor approximations for unit-square labels in the 1-position model. Gemsa et al. [3] similarly studied continuous rotation, and give a PTAS for unit-square labels in the 1-position model. Treatment of other label models and more general point trajectories are sorely missing.

*Dept. of Computer Science, TU Eindhoven, the Netherlands, k.a.buchin@tue.nl, dirk@dirkgerrits.com

**Our results.** We believe the relative lack of theoretical results for dynamic point labeling is not due to a lack of attempts. Intuitively, dynamic point labeling should be much harder than its static counterpart. We prove and quantify this intuition. Specifically, we prove for unit-square labels in the 4-position, 2-slider, and 4-slider models that deciding whether there exists a dynamic labeling without intersections is strongly PSPACE-complete. This is the case when points are added or removed from the point set, when (some of) the points move, and when the points set is panned, rotated, or zoomed within a finite viewport. Any dynamic generalization of the mentioned static optimization problems is therefore strongly PSPACE-hard in these settings. Additionally, we prove that label-size maximization on dynamic point sets admits no PTAS unless P=PSPACE. To save space in this extended abstract, we present our proofs in picture form, with minimal accompanying text.

## 2 Non-deterministic constraint logic

To prove PSPACE-hardness of dynamic point labeling, we will reduce from *non-deterministic constraint logic* (NCL) [4], which is a sort of abstract, single-player game. The game board is a *constraint graph*: an undirected graph with weights on both the vertices and the edges. A *configuration* of the constraint graph specifies an orientation for each of its edges. A configuration is *legal* if and only if each vertex $v$'s *inflow* (the summed weight of its incoming edges) is at least $v$'s own weight. To make a *move* is to reverse a single edge in a legal configuration such that the resulting configuration is again legal. For this game each of the following questions is PSPACE-complete [4].

- *Configuration-to-configuration NCL*: Given two legal configurations $A$ and $B$, is there a sequence of moves transforming $A$ into $B$?
- *Configuration-to-edge NCL*: Given a legal configuration $A$ and an orientation for a single edge $e_B$, is there a sequence of moves transforming $A$ into a legal configuration $B$ in which $e_B$ has the specified orientation?
- *Edge-to-edge NCL*: Given orientations for edges $e_A$ and $e_B$, do there exist legal configurations $A$ and $B$, and a sequence of moves transforming the one into the other, such that $e_A$ has the specified orientation in $A$ and $e_B$ has the specified orientation in $B$?

These decision problems remain PSPACE-complete even for planar, 3-regular constraint graphs consisting only of AND vertices and protected OR vertices. An *AND vertex* has a weight of 2, and its three incident edges have weights 1, 1, 2. To orient the weight-2 edge away from the vertex requires both weight-1 edges to be oriented towards the vertex. An *OR vertex* and its three incident edges all have weight 2. Thus at least one edge needs to be oriented towards the vertex at all times. An OR vertex is called *protected* if it has two edges that, because of constraints imposed on them by the rest of the constraint graph, cannot both be directed inward.

**Gadgets.** Figure 1 shows the *gadgets* we will employ in our reduction. We call the points with dark gray labels *blockers*, as we can consider these labels fixed obstacles (labeling them differently than shown will only restrict the placement of other labels further). In the 4-position model, the depicted blockers restrict the light gray labels marked $A$, $B$, and $C$ to two possible positions each. We call the label position closest to the center of the vertex gadget *inward*, and the other *outward*. In the figure, labels $A$ and $B$ are placed inward, and label $C$ is placed outward. In the slider models, labels can take on any position in between these two extremes, but we may assume that only a very small range of positions is actually used. Consider, for example, label $A$. Without moving $A'$, we can only move $A$ up by at most $\varepsilon$, or left by at most $2\varepsilon$. We refer to this range of positions as *inward*, and define it similarly for $B$ and $C$. If (and only if) $A'$ is moved left by at least $1-\varepsilon$, then $A$ can move further upward. We may then move $A$ all the way to its uppermost position, and there is no reason not to do so. Thus we may define *outward* as in the 4-position model. With this terminology in place, we shall prove that our gadgets faithfully simulate a constraint graph, with labels placed *inward* corresponding to edges directed *out* of a vertex, and labels placed *outward* corresponding to edges directed *into* a vertex.

**Theorem 1** *Let $G$ be a planar constraint graph with $n$ vertices, and let $\varepsilon$ and $\delta$ be two real numbers with $0 < \varepsilon \leqslant \delta < 1 - 3\varepsilon$. One can then construct a point set $P = P(G, \delta, \varepsilon)$ with the following properties:*

- *the size of $P$, and the coordinates of all points in $P$, are polynomially bounded in $n$,*
- *for any $s \in [1, 1 + \delta - \varepsilon]$, there is a one-to-one correspondence between legal configurations of $G$ and overlap-free static labelings of $P$ with $s \times s$ square labels in the 4-position model, and*
- *there is a sequence of moves transforming one legal configuration of $G$ into another if and only if there is a dynamic labeling transforming the corresponding static labelings of $P$ into each other.*

*When $\delta < 1/3 - 4\varepsilon/3$, the same results hold for the 2- and 4-slider models.*

**Proof.** Because of the structure of Hearn and Demaine's hardness proof of NCL [4], we may assume that the given constraint graph $G$ is embedded on a grid with its vertices on grid vertices and its edges being interior-disjoint paths along grid lines. We turn this grid by 45° relative to the coordinate axes, and replace the vertices and edges by appropriate gadgets.

Figure 1(a) shows an OR gadget that works if we assume that $A$ and $B$ are never both placed outward.

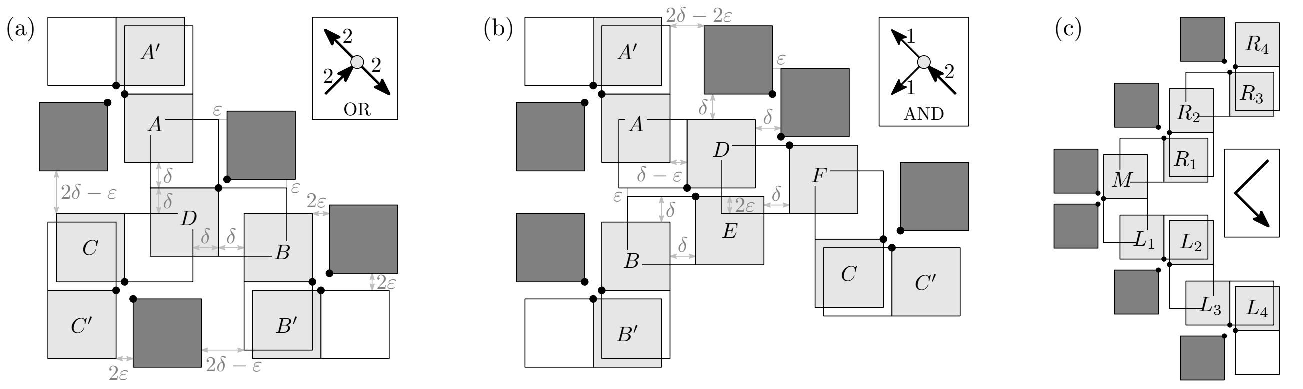


Figure 1: Gadgets simulating NCL's (a) protected OR vertices, (b) AND vertices, and (c) edges. The insets in the top-right corners show the NCL constructions being simulated.

This means it simulates a protected OR vertex where edges $A$ and $B$ are never both directed into the vertex. Figure 1(b) shows an AND gadget, where $A$ and $B$ represent the weight-1 edges. Figure 1(c) shows an edge gadget to connect vertex gadgets over longer distances. It is possible for both ends of an edge gadget to be placed inward for their respective vertex gadgets, however, this is of no concern as this corresponds to an edge of the constraint graph being directed into neither of its two incident vertices. If all inflow constraints are satisfied with such edges present, then orienting them arbitrarily does not disturb this.

If we scale the grid appropriately, then the gadgets will fit together so as to form the desired point set $P$. We omit the proof of $P$'s claimed properties; it can be reconstructed by ruminating on Figure 1. □

## 3 Hardness of dynamic point labeling

In this section we will define a number of dynamic point-labeling problems in an offline setting. That is, we assume we are given the arrival and departure times of all points, as well as their trajectories and the "trajectory" of the viewport. We will seek a dynamic labeling $\mathcal{L}$ for the time interval $[a, b]$, and distinguish four cases based on whether or not the static labelings $\mathcal{L}(a)$ and $\mathcal{L}(b)$ are pre-specified. We will show PSPACE-hardness by reduction from configuration-to-edge NCL for the case where $\mathcal{L}(a)$ is given, and omit the symmetric case where $\mathcal{L}(b)$ is given instead. We also omit the cases where neither or both static labelings are given, which can be proven by similar reductions from edge-to-edge and configuration-to-configuration NCL, respectively. (The sole exception is zooming, where pre-specifying neither static labeling makes the problem "merely" NP-complete.)

**Theorem 2** *The following decision problem is strongly PSPACE-complete for the 4-position, 2-slider, and 4-slider label models.*

***Given:*** *A dynamic point set $P$ (with given trajectories, arrival times, and departure times), numbers $a$ and $b$ with $a < b$, and a static labeling $\mathcal{L}(a)$ for $P(a)$.*

***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for $P$ respecting $\mathcal{L}(a)$ that labels all points with non-overlapping unit-square labels over the time interval $[a, b]$.*

*Additionally, unless P = PSPACE, the maximum label size for which there is such an $\mathcal{L}$ cannot be $(4/3 - \varepsilon')$-approximated in polynomial time for any $\varepsilon' > 0$. For the 4-position model, this holds even for a $(2 - \varepsilon')$-approximation.*

*All of the above remains true when*
- *all points are stationary, and during $[a, b]$ one point is removed and one point is added,*
- *no points are added or removed, and all points move at the same, constant speed along straight-line trajectories, or*
- *no points are added or removed, and all points but one are stationary.*

**Proof.** We only prove the claim of PSPACE-hardness, which we do by reducing from configuration-to-edge NCL. Thus we are given a constraint graph $G$ with a legal starting configuration $A$ and the goal orientation of a single edge $e_B$, and want to decide whether there is a sequence of moves on $G$ starting from $A$ that will result in $e_B$ having its specified orientation. By Theorem 1, we may construct a point set $P = P(G, \delta, \varepsilon)$ and a valid static labeling $\mathcal{L}(a)$ corresponding to configuration $A$. Now pick one blocker $q \in P$ in the edge gadget for $e_B$, and suppose $p \in P$ is the point for which $q$ blocks some label candidate(s). We now make $q$ move at a constant speed of $v = 2\varepsilon/(b - a)$ towards the nearest non-blocked candidate of $p$. Figure 2(a) shows the end result at time $b$: the edge gadget has become constrained to a single orientation around $p$. Thus there is a sequence of moves on $A$ resulting in $e_B$ having the specified orientation if and only if there is a dynamic labeling $\mathcal{L}$ for this dynamic point set starting at $\mathcal{L}(a)$. The same result may be obtained by moving $q$ at speed $v/2$ and moving all other points at speed $v/2$ in the opposite direction. Alternatively, we may remove the point $q$ from $P$ and re-insert it at its modified location. □

In interactive mapping applications, users are presented with a rectangular *viewport* $V$ showing a portion of a larger map. By dynamically panning, rotating, and/or zooming the map, the user controls which portion of the map is displayed at any given time. The task of labeling the points inside $V$ can be seen as a special case of labeling dynamic point sets. Continuous panning, rotation, and zooming of the map may cause points to enter or leave $V$ at its boundary, and causes all points within $V$ to move on continuous trajectories. We will require only the points inside $V$ to be labeled, and these labels must be fully contained in $V$. Points outside of $V$ need not be labeled, but we may wish to do so in order to ensure a smooth dynamic labeling. Whether we do so or not makes no difference in the complexity of these problems.

**Theorem 3** *The following decision problem is strongly PSPACE-complete for the 4-position, 2-slider, and 4-slider label models.*

***Given:*** *A closed rectangle $V$, a points set $P$ being panned, rotated, or zoomed in a single direction at constant speed, numbers $a$ and $b$ with $a < b$, and a static labeling $\mathcal{L}(a)$ for $P(a)$.*
***Decide:*** *Whether there exists a dynamic labeling $\mathcal{L}$ for $P$ respecting $\mathcal{L}(a)$ that labels $P(t) \cap V$ with non-overlapping unit-square labels inside $V$ for all $t \in [a, b]$.*

**Proof.** We reduce from configuration-to-edge NCL in a way similar to Theorem 2, but we now make sure that $G$ is laid out on the grid in such a way that $e_B$ is on the far right. Next, we add an additional points $q'$ to the right of $e_B$ by $1 - 3\varepsilon$, as in Figure 2(b). We then construct a viewport rectangle $V$ with its right side being $\varepsilon$ to the left of $q'$. If we then start panning so that the points move to the left, $q'$ will enter $V$ after a distance $\varepsilon$. We must then label $q'$ in such a way that $e_B$ becomes constrained to a single orientation. The same result can be achieved by rotating the points clockwise, assuming the center of viewport is above $q'$. Unlike with panning, rotating also changes the distances between the points inside of the gadgets. But if the initial distance between $q'$ and $V$ is small enough, $q'$ will enter $V$ before the gadgets are disturbed. Zooming out similarly makes $q'$ enter $V$. □

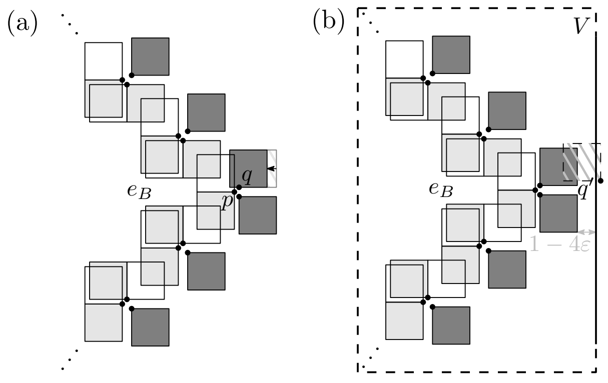


Figure 2: Reduction from configuration-to-edge NCL to dynamic labeling of (a) moving points, and (b) static points under panning of the viewport $V$.

## 4 Conclusion

We have examined the complexity of dynamic point labeling. For a set of points where points may be added or removed over time, and where the points present move along continuous trajectories, we seek a smooth function from time to static point labelings. For the 4-position, 2-slider, and 4-slider models we have shown that deciding whether there exists such a labeling in which no labels ever overlap is strongly PSPACE-complete. In addition, finding the maximum label size at which such a labeling does exist admits no PTAS unless P=PSPACE. For the 4-position model a 2-approximation is the best that can be hoped for, and for the slider models a 4/3-approximation. The PSPACE-completeness results also apply for special cases of dynamic point labeling in which the point set is panned, rotated, or zoomed inside a fixed viewport.

It remains to examine the complexity of other label models, specifically the 1- and 2-position models, as well as the 1-slider model. Presumably the decision problem is easier for these models, as in the case of static point labeling. It may also be of interest to re-examine the special cases of zooming and rotation with an infinite viewport. Our reduction currently uses points at the boundary of the viewport, but we believe this can be avoided. Most importantly, perhaps, is the continued pursuit of approximation algorithms for dynamic point labeling.

## References

[1] K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom. Theory Appl.*, 43(3):312–328, 2010.

[2] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[3] A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In F. Dehne, J. Iacono, and J.-R. Sack, editors, *Proc. 11th Internat. Sympos. Algorithms and Data Structures*, pages 451–462, 2009.

[4] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoret. Comput. Sci.*, 343(1–2):72–96, 2005.

[5] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13:21–47, 1999.