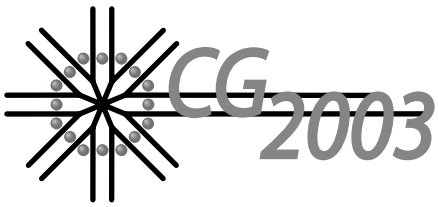


19th European Workshop on Computational Geometry

Program & Abstracts

March 24 - 26, 2003
Institute of Computer Science I
University of Bonn
Germany

Table of Contents

Preliminary Program

Monday, March 24th

08:00 – 09:00	Registration and welcome reception
09:00 – 09:10	Welcome
09:30 – 09:50	**The one-round Voronoi game replayed** Fekete, S.P., Meijer, H.
09:30 – 09:50	**The Voronoi diagram for n disjoint spherical sites of $O(1)$ different radii has complexity $O(n^2)$** Ó Dúnlaing, C.
09:50 – 10:10	**The anchored Voronoi diagram: Static and dynamic versions and applications** Barcia, J.A., Díaz-Báñez, J.M., Gómez, F., Ventura, I.
10:10 – 10:30	**Approximating planar subdivisions and generalized Voronoi diagrams from random sections** Coll, N., Hurtado, F., Sellarès, J.A.
10:30 – 11:00	Coffee break
11:00 – 11:20	**Sweeping an arrangement of quadrics in 3D** Mourrain, B., Técourt, J.-P., Teillaud, M.
11:20 – 11:40	**Parametric voxel geometry control for digital morphogenesis** Fischer, Th., Fischer, To.
11:40 – 12:00	**Controlled perturbation for arrangements of circles** Halperin, D., Leiserowitz, E.
12:00 - 13:00	Lunch
13:00 - 14:00	Invited Lecture: **Geometric approximation using core sets** Pankaj Kumar Agarwal, Duke University
14:00 – 14:20	Coffee break
14:20 – 14:40	**Vertex cover and connected guard set** Żyliński, P.
14:40 – 15:00	**An optimal competitive on-line algorithm for the minimal clique** Jaromczyk, J.W., Pezarski, A., Ślusarek, M.

15:00 – 15:20 **Convex sets in graphs**
Cáceres, J., Márquez, A., Oellermann, O.R., Puertas, M.L.

15:20 – 15:40 **Graphs of triangulations and perfect matchings**
Houle, M., Hurtado, F., Noy, M., Rivera-Campo, E.

15:40 - 16:00 Coffee break

16:00 – 16:20 **Computing the detour of polygons**
Grüne, A., Klein, R., Langetepe, E.

16:20 – 16:40 **More results about spanners in the l_1-metric**
Cáceres J., Grima, C.I., Márquez, A., Moreno-González, A.

16:40 – 17:00 **Approximately matching polygonal curves under translation, rotation and scaling with respect to the Fréchet-Distance**
Clausen, M., Mosig, A.

Tuesday, March 25th

09:00 – 10:00 Invited lecture:
Approximation algorithms for geometric shortest path problems
Jörg-Rüdiger Sack, Carleton University

10:00 – 10:20 Coffee break

10:20 – 10:40 **The polytope of non-crossing graphs on a planar point set**
Orden, D., Santos, F.

10:40 – 11:00 **Affine representations of abstract convex geometries**
Kashiwabara, K., Nakamura, M., Okamoto, Y.

11:00 – 11:20 **Maximum subsets in Euclidean position in Euclidean 2-orbifolds and the sphere**
Abellanas, M., Cortés, C., Hernández, G., Márquez, A., Valenzuela, J.

11:20 – 11:40 **Optimal pants decompositions and shortest freely homotopic loops on an orientable surface**
Colin de Verdière, É., Lazarus, F.

11:40 – 12:00 **Geometric games on triangulations**
Aichholzer, O., Bremner, D., Demaine, E.D., Hurtado, F., Kranakis, E., Krasser, H., Ramaswami, S., Sethia, S., Urrutia, J.

12:00 – 13:00 Lunch

13:00 – 13:20 **Cutting triangular cycles of lines in space**
Aronov, B., Koltun, V., Sharir, M.

13:20 – 13:40 **Red-blue separability problems in 3D**
Hurtado, F., Seara, C., Sethia, S.

13:40 – 14:00 **The maximum number of edges in a three-dimensional grid drawing**
Bose, P., Czyzowicz, J., Morin, P., Wood, D.R.

14:00 – 14:20 Coffee break

14:20 – 14:40 **Constrained higher order Delaunay triangulations**
Gudmundsson, J., Haverkort, H., van Kreveld, M.

14:40 – 15:00 **An approach to exhaustive generation of objects without testing on isomorphisms. Application of the method to the cell growth problem**
Alboul, L., Netchaev, A.

15:00 – 15:20 **Kinetic convex hull maintenance using nested convex hulls**
Razzazi, M.R., Sajedi, A.

15:40 - 16:00 **Optimal tolerancing in mechanical design using polyhedral computation tools**
Fukuda, K., Petit, J-P.

16:00 – 16:20 **Approximating the visible region of a point on a terrain**
Boaz, B.-M, Paz, C., Katz, M.J.

16:20 – 16:40 **Pheromone-guided dispersion for swarms of robots**
Hsiang, T.-R, Sztainberg, M.

16:40 – 17:00 Business meeting

19:30 Conference dinner

Wednesday, March 26th

09:15 – 10:15 Invited lecture:
Similarity between image and terrain: Geometric approaches to computer vision
Tetsuo Asano, JAIST

10:15 – 10:40 Coffee break

10:40 – 11:00 **Orthogonal segment stabbing**
Katz, M.J., Mitchell, J.S.B., Nir, Y.

11:00 – 11:20 **Alternating paths along orthogonal segments**
Tóth, C.D.

11:20 – 11:40 **Shortest paths in polygonal domains with polygon-meet constraints**
Khosravi, R., Ghodsi, M.

11:40 – 12:00 **Tiling polyominoes with squares that touch the boundary**
Spillner, A.

12:00 – 13:00 Lunch

13:00 – 13:20 **Best fitting rectangles**
Abellanas, M., Hurtado, F., Icking, C., Ma, M., Palop, B., Ramos, P.A.

13:20 – 13:40 **Algorithms for placing and connecting facilities and their comparative analysis**
Kedem, K., Rabaev, I., Sokolovsky, N.

13:40 – 14:00 **Chips on wafers, or packing rectangles into grids**
Andersson, M., Gudmundsson, J., Levcopoulos, C.

14:00 – 14:20 **Unit height k-position map labeling**
Rostamabadi, F., Ghodsi, M.

14:40 – 15:00 **Good NEWS: Partitioning a simple polygon by compass directions**
van Kreveld, M., Reinbacher, I.

15:00 – 15:20 **Significant-presence range queries in categorical data**
de Berg, M., Haverkort, H.J.

15:20 – 15:40 **Efficient contour tree construction and computation of Betti numbers in scalar fields**
Lenz, T., Rote, G.

15:40 – 15:45 Good-bye

Thursday, March 27th

10:00 – 14:00 Excursion

Invited Lectures

Geometric Approximation using Core Sets

Pankaj Kumar Agarwal
Duke University, USA

ABSTRACT

This talk presents a general approximation technique for various geometric optimization problems such as clustering, shape-fitting, and extent measures. For a given objective function μ and a parameter $\varepsilon > 0$, it computes in time $O(n + 1/\varepsilon^{O(1)})$ a subset $Q \subseteq P$ of size $1/\varepsilon^{O(1)}$, with the property that $(1-\varepsilon)\mu(P) \leq \mu(Q) \leq \mu(P)$. Specific applications of our technique include ε-approximation algorithms for (i) computing diameter, width, and smallest bounding box, ball, and cylinder of P, (ii) maintaining these measures in a streaming model, (iii) maintaining these measures for a set of moving points, (iv) fitting spheres and cylinders through a point set P, and (iv) clustering stationary and moving points.

Similarity between Image and Terrain: Geometric Approaches to Computer Vision

Tetsuo Asano
School of Information Science, JAIST, Japan

Abstract

Image and terrain have strong similarity in the sense that both of them are commonly represented in matrices. Each matrix element is height information for terrain and color/brightness for image. Terrain is usually converted to a contour representation that is a collection of contour lines of equal height. Our projects start with this similarity. That is, we apply conour representation to images. Once an image is converted to contour representation, it is now a geometric object to which a rich source of geometric algorithms can be applied. Another advantage of such geometric representation is that it gives us global information or structural information of images. It should be compared to the traditional approach using a quad-tree data structure on image lattice for characterizing a structure of an image.

In this talk I will first survey how computational geometry has contributed to computer vision and computer graphics mainly in connection to the contour representation. Several topics are included: an efficient algorithm for obtaining contour representation from a given image matrix with some experimental results, how to represent structural information among contour lines and how to compute it efficiently without following all contour lines — contour tree, direct application of contour representation to geometric deformation of an image such as scaling operation for enlargement or contraction, rotation, and removing a part of an image. Such deformations may look easy tasks, but simple algorithms may produce ugly images. In fact, contraction of an image seems to be the easiest task among them, but it is related to some combinatorial optimization problem.

A natural way of defining a contour line for a brightness level i is to follow the boundary of a connected region of pixels with brightness levels greater than or equal to i, which results in a closed loop consisiting of alternating horizontal and vertical line segments. How to approximate such a rectilinear path by a smoother curve such as Basier or Spline curve is also an interesting topic. It is not so obvious due to an important property of contour representation that any contour line does not properly intersect itself or any other contour line. If we use a polygonal line instead of those curves then we can prove some convergence theory that the length of the polyline converges to the true length of an object characterized by the contour line with increase of resolution.

Contour lines play an important role for region segmentation, a basic task in pattern recognition. Region segmentation is to separate an object from its background. If brightness levels in an objact are well separated to those in the background, it is easy to find an appropriate contour line enclosing the object. Otherwise, we could find appropriate contour line by evaluating separation of every contour line. A serious problem of this approach is its computing time. Fortunately, we could use a rough region segmentation restricted to x-monotone shapes using dynamic programming as a guide of our objective contour line.

Approximation algorithms for geometric shortest path problems*

Jörg-Rüdiger Sack†

ABSTRACT

Shortest path problems are among the fundamental problems studied in computational geometry, network optimization and graph algorithms. These problems arise naturally in application areas such as robotics and geographical information systems. Aside from the importance of shortest paths problems in their own right, often they appear in the solutions to other problems.

Shortest path problems can be categorized by various factors which include the dimensionality of the space, the type and the number of objects or obstacles, and the distance measure used (e.g., Euclidean, number of links, or weighted distances). In two and three dimensions a variety of shortest path problems have been studied over the last three decades. Particular problem instances studied include computing Euclidean shortest paths between two points inside a simple polygon and amidst polygonal and polyhedral obstacles, reporting shortest paths on the surface of a convex or a non-convex polyhedron, including approximation algorithms. Research articles and surveys have been written presenting the state-of-the-art in this area.

Of particular interest for this talk is the *weighted region problem* which is a natural generalization of the Euclidean shortest path problem. In 2-dimensions it can be stated as: A planar triangulated subdivision is given consisting of n faces, where each face has a positive weight which represents

*Research supported in part by NSERC and SUN Microsystems of Canada.

†School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. sack@scs.carleton.ca Some of the work discussed in this talk is jointly carried out with A. Maheshari and L. Aleksandrov.

the cost of traveling through that face. The weight could be determined e.g., by the slope or other characteristics of the face (water, forest, rock, ...). The cost of travel through the face is the product of Euclidean length and face-weight. The cost of a (weighted) path is sum of the costs of travel through all faces intersected by the path.

Existing algorithms for many of the interesting shortest path problems are either very complex in design/implementation and/or have very large time and space complexities. Hence they are unappealing to practitioners and pose a challenge to theoreticians. This coupled with the fact that geographic/spatial models are approximations of reality anyway and high-quality paths are favored over optimal paths that are "hard" to compute, approximation algorithms are suitable and necessary.

In this talk, we discuss the classical geometric problem of determining shortest paths through weighted (and unweighted) domains (in 2 and 3 dimensions). We present several approximation algorithms and discuss them from a practical and/or theoretical view-point.

Contributions

The One-Round Voronoi Game Replayed

Sándor P. Fekete * Henk Meijer †

Abstract

We consider the one-round Voronoi game, where the first player ("White", called "Wilma") places a set of n points in a rectangular area Q of aspect ratio $\rho \leq 1$, followed by the second player ("Black", called "Barney"), who places the same number of points. Each player wins the fraction of Q closest to one of his points, and the goal is to win more than half of the total area. This problem has been studied by Cheong et al. who showed that for large enough n and $\rho = 1$, Barney has a strategy that guarantees a fraction of $1/2 + \alpha$, for some small fixed α.

We resolve a number of open problems raised by that paper. In particular, we give a precise characterization of the outcome of the game for optimal play: We show that Barney has a winning strategy for $n \geq 3$ and $\rho > \sqrt{2}/n$, and for $n = 2$ and $\rho > \sqrt{3}/2$. Wilma wins in all remaining cases, i.e., for $n \geq 3$ and $\rho \leq \sqrt{2}/n$, for $n = 2$ and $\rho \leq \sqrt{3}/2$, and for $n = 1$. We also discuss complexity aspects of the game on more general boards, by proving that for a polygon with holes, it is NP-hard to maximize the area Barney can win against a given set of points by Wilma.

Keywords:

Voronoi diagram, Voronoi game, Competitive facility location, NP-hardness.

1 Introduction

When determining success or failure of an enterprise, *location* is one of the most important issues. Probably the most natural way to determine the value of a possible position for a facility is the distance to potential customer sites. Various geometric scenarios have been considered; see the extensive list of references in the paper by Fekete, Mitchell, and Weinbrecht [6] for an overview.

One particularly important issue in location theory is the study of strategies for competing players. See the surveys by Tobin, Friesz, and Miller [8], by Eiselt and Laporte [4], and by Eiselt, Laporte, and Thisse [5].

A simple geometric model for the value of a position is used in the *Voronoi game*, which was proposed by Ahn et al. [1] for the one-dimensional scenario and extended by Cheong et al. [2] to the two- and higher-dimensional case. In this game, a site s "owns" the part of the playing arena that is closer to s than to any other site. Both considered a two-player version with a finite arena Q. The players, White ("Wilma") and Black ("Barney"), place points in Q; Wilma plays first. No point that has been occupied can be changed or reused by either player. Let W be the set of points that were played by the end of the game by Wilma, while B is the set of points played by Barney. At the end of the game, a Voronoi diagram of $W \cup B$ is constructed; each player wins the total area of all cells belonging to points in his or her set. The player with the larger total area wins.

Ahn et al. [1] showed that for a one-dimensional arena, i.e., a line segment $[0, 2n]$, Barney can win the *n-round game*, in which each player places a single point in each turn; however, Wilma can keep Barney's winning margin arbitrarily small. This differs from the *one-round game*, in which both players get a single turn with n points each: Here, Wilma can force a win by playing the odd integer points $\{1, 3, \ldots, 2n-1\}$; again, the losing player can make the margin as small as he wishes. The used strategy focuses on "key points". The question raised in the end of that paper is whether a similar notion can be extended to the two-dimensional scenario. We will see in Section 3 that in a certain sense, this is indeed the case.

Cheong et al. [2] showed that the two- or higher-dimensional scenario differs significantly: For sufficiently large $n \geq n_0$ and $\rho = 1$, the second player has a winning strategy that guarantees at least a fixed fraction of $1/2 + \alpha$ of the total area. Their proof used a clever combination of probabilistic arguments to show that Barney will do well by playing

*Abt. für Mathematische Optimierung, TU Braunschweig, D-38106 Braunschweig, Germany, s.fekete@tu-bs.de.

†Dept. of Computer Science, Queen's University, Kingston, Ont K7L 3N6, Canada, henk@cs.queensu.ca. Partially supported by NSERC. This work was done while visiting TU Braunschweig.

a random point. The paper gives rise to some interesting open questions:

- How large does n_0 have to be to guarantee a winning strategy for Barney? Wilma wins for $n = 1$, but it is not clear whether there is a single n_0 for which the game changes from Wilma to Barney, or whether there are multiple changing points.
- Barney wins for sufficiently "fat" arenas, while Wilma wins for the degenerate case of a line. How exactly does the outcome of the game depend on the aspect ratio of the playing board?
- What happens if the number of points played by Wilma and Barney are not identical?
- What configurations of white points limit the possible gain of black points? As candidates, square or hexagonal grids were named.
- What happens for the multiple-round version of the game?
- What happens for asymmetric playing boards?

For rectangular boards and arbitrary values of n, we will show when Barney can win the game. If the board Q has aspect ratio ρ with $\rho \leq 1$, we prove the following:

- Barney has a winning strategy for $n \geq 3$ and $\rho > \sqrt{2}/n$, and for $n = 2$ and $\rho > \sqrt{3}/2$. Wilma wins in all remaining cases, i.e., for $n \geq 3$ and $\rho \leq \sqrt{2}/n$, for $n = 2$ and $\rho \leq \sqrt{3}/2$, and for $n = 1$.
- If Wilma does not play her points on an orthogonal grid, then Barney wins the game.

In addition, we hint at the difficulties of more complex playing boards by showing the following:

- If Q is a polygon with holes, and Wilma has made her move, it is NP-hard to find a position of black points that maximizes the area that Barney wins.

This result is also related to recent work by Dehne, Klein, and Seidel [3] of a different type: They studied the problem of placing a single black point within the convex hull of a set of white points, such that the resulting black Voronoi cell in the unbounded Euclidean plane is maximized. They showed that there is a unique local maximum.

The rest of this paper is organized as follows. After some technical preliminaries in Section 2, Section 3 shows that Barney always wins if Wilma does not place her points on a regular orthogonal grid. This is used in Section 4 to establish our results on the critical aspect ratios. Section 5 presents some results on the computational complexity of playing optimally in a more complex board. Some concluding thoughts are presented in Section 6.

For this 4-page abstract, all proofs have been omitted; a full version of the paper [7] is available electronically.

2 Preliminaries

In the following, Q is the playing board. Q is a rectangle of *aspect ratio* ρ, which is the ratio of the length of the smaller side divided by the length of the longer side. Unless noted otherwise (in some parts of Section 5), both players play n points; W denotes the n points played by Wilma, while B is the set of n points played by Barney. All distances are measured according to the Euclidean norm. For a set of points P, we denote by $V(P)$ the (Euclidean) Voronoi diagram of P. We call a Voronoi diagram $V(P)$ a regular grid if

- all Voronoi cells are rectangular, congruent and have the same orientation;
- each point $p \in P$ lies in the center of its Voronoi cell.

If e is a Voronoi edge, $C(e)$ denotes a Voronoi cell adjacent to e. If $p \in P$, then $C(p)$ denotes the Voronoi cell of p in $V(P)$. $\partial C(p)$ is the boundary of $C(p)$ and $|C(p)|$ denotes the area of $C(p)$. $|e|$ denotes the length of an edge e. Let x_p and y_p denote the x- and y-coordinates of a point p.

3 A Reduction to Grids

As a first important step, we reduce the possible configurations that Wilma may play without losing the game. The following holds for boards of any shape:

Lemma 1 *If $V(W)$ contains a cell that is not point symmetric, then Barney wins.*

The following theorem is based on this observation and will be used as a key tool for simplifying our discussion in Section 4.

Theorem 2 *If the board is a rectangle and if $V(W)$ is not a regular grid, then Barney wins.*

4 Critical Aspect Ratios

In this section we prove the main result of this paper: if $n \geq 3$ and $\rho > \sqrt{2}/n$, or $n = 2$ and $\rho > \sqrt{3}/2$, then Barney wins. In all other cases, Wilma wins. The proof proceeds by a series of lemmas. We start by noting the following easy observation.

Lemma 3 *Barney wins, if and only if he can place a point p that steals an area strictly larger than $|Q|/2n$ from W.*

Next we take care of the case $n = 2$; this lemma will also be useful for larger n, as it allows further reduction of the possible arrangements Wilma can choose without losing.

Lemma 4 *If $n = 2$ and $\rho > \sqrt{3}/2$, then Barney wins. If the aspect ratio is smaller, Barney loses.*

The gain for Barney is small if ρ is close to $\sqrt{3}/2$. Computer experiments have been used to compute the gain for Barney for values of $\rho > \sqrt{3}/2$. Not surprisingly, the largest gain was found for $\rho = 1$. If the board has size 1 by 1, Barney can gain an area of approximately 0.2548 with his first point, by placing it at (0.66825,0.616). as illustrated in Figure 1(a).

Lemma 5 *Suppose that the board is rectangular and that $n = 4$. If Wilma places her point on a regular 2×2 grid, Barney can gain 50.78% of the board.*

The value in the above lemma is not tight. For example, if Wilma places her point in a 2 by 2 grid on a square board, we can compute the area that Barney can gain with his first point. If Barney places it at (0.5,0.296), he gains approximately 0.136. For an illustration, see Figure 1(b). By placing his remaining three points at $(0.25 - 4\varepsilon/3, 0.25)$, $(0.25 - 4\varepsilon/3, 0.75)$, and $(0.75 + 4\varepsilon/3, 0.75)$ Barney can gain a total area of size of around $0.511 - \varepsilon$ for arbitrary small positive ε. For non-square boards, we have found larger wins for Black. This suggests that Barney can always gain more than 51% of the board if Wilma places her four points in a 2 by 2 grid.

The above discussion has an important implication:

Corollary 6 *If $n \geq 3$, then Wilma can only win by placing her points in a $1 \times n$ grid.*

This sets the stage for the final lemma:

Lemma 7 *Let $n \geq 3$. Barney can win if $\rho > \sqrt{2}/n$; otherwise, he loses.*

Computational experiments have confirmed that Barney wins the largest area with his first point if he places it at $(0, (4r - 2\sqrt{r^2 + 6})/3)$.

Theorem 8 *If $n \geq 3$ and $\rho > \sqrt{2}/n$, or $n = 2$ and $\rho > \sqrt{3}/2$, then Barney wins. In all other cases, Wilma wins.*

5 A Complexity Result

The previous section resolves most of the questions for the one-round Voronoi game on a rectangular board. Clearly, there are various other questions related to more complex boards; this is one of the questions raised in [2]. Lemma 1 still applies if Wilma's concern is only to avoid a loss. Moreover, it is clear that all of Wilma's Voronoi cells must have the same area. For many boards, both of these conditions may be impossible to fulfill. It is therefore natural to modify the game by shifting the critical margin that decides a win or a loss. We show in the following that it is NP-hard to decide whether Barney can beat a given margin for a polygon with holes, and all of Wilma's stones have already been placed. (In a non-convex polygon, possibly with holes, we measure distances according to the geodesic Euclidean metric, i.e., along a shortest path within the polygon.)

Theorem 9 *For a polygon with holes, it is NP-hard to maximize the area Barney can claim, even if all of Wilma's points have been placed.*

6 Conclusion

We have resolved a number of open problems dealing with the one-round Voronoi game. There are still several issues that remain open. What can be said about achieving a fixed margin of win in all of the cases where Barney can win? We believe that our above techniques can be used to resolve this issue. As we can already quantify this margin if Wilma plays a grid, what is still needed is a refined version of Lemma 1 and Theorem 2 that guarantees a fixed margin as a function of the amount that Wilma deviates from a grid. Eventually, the guaranteed margin should be a function of the aspect ratio. Along similar lines, we believe that it is possible to resolve the question stated by [2] on the scenario where the number of points played is not equal.

Probably the most tantalizing problems deal with the multiple-round game. Given that finding an optimal set of points for a single player is NP-hard,

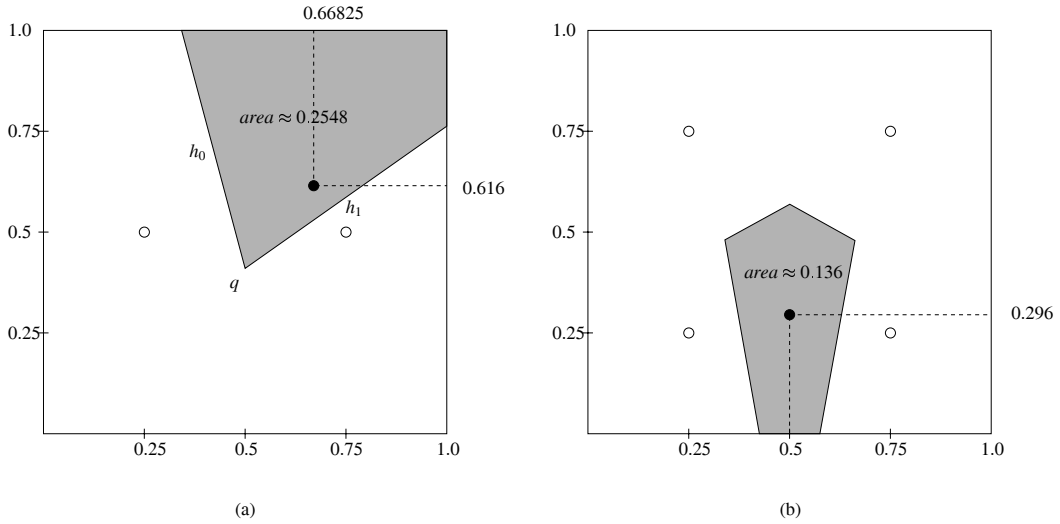


Figure 1: Barney has gained more than a quarter (a) more than an eighth (b) of the playing surface.

it is natural to conjecture that the two-player, multiple round game is PSPACE-hard. Clearly, there is some similarity to the game of Go on an $n \times n$ board, which is known to be PSPACE-hard [9] and even EXPTIME-complete [10] for certain rules.

However, some of this difficulty results from the possibility of capturing stones. It is conceivable that at least for relative simple (i.e., rectangular) boards, there are less involved winning strategies. Our results from Section 4 show that for the cases where Wilma has a winning strategy, Barney cannot prevent this by any probabilistic or greedy approach: Unless he blocks one of Wilma's key points by placing a stone there himself (which has probability zero for random strategies, and will not happen for simple greedy strategies), she can simply play those points like in the one-round game and claim a win. Thus, analyzing these key points may indeed be the key to understanding the game.

References

[1] H.-K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, and R. van Oostrum. Competitive facility location along a highway. In *Proc. 9th COCOON*, Springer LNCS # 2108, pages 237–246, 2001.

[2] O. Cheong, S. Har-Peled, N. Linial, and J. Matousek. The one-round Voronoi game. In *Proc 18th SoCG*, pages 97–101, 2002.

[3] Frank Dehne, Rolf Klein, and Raimund Seidel. Maximizing a Voronoi region: The convex case. In *Proc. 13th ISAAC*, Springer LNCS #2518, pages 624–634, 2001.

[4] H.A. Eiselt and G. Laporte. Competitive spatial models. *European Journal of Operational Research*, 39:231–242, 1989.

[5] H.A. Eiselt, G. Laporte, and J.-F. Thisse. Competitive location models: A framework and bibliography. *Transportation Science*, 27:44–54, 1993.

[6] S. P. Fekete, J. S. B.Mitchell, and K. Weinbrecht. On the continuous Weber and k-median problems. In *Proc. 16th SoCG*, pages 70–79, 2000.

[7] S.P. Fekete and H. Meijer. The one-round Voronoi game replayed. *http://mo.math.nat.tu-bs.de/~fekete/publications.html*, 2002.

[8] T.L. Friesz, R.L. Tobin, and T. Miller. Existence theory for spatially competitive network facility location models. *Annals of Operations Research,* 18:267276, 1989.

[9] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM,*27:393401, 1980.

[10] J.M. Robson. The complexity of Go. *In Information Processing: Proceedings of IFIP Congerss,*pages 4134417, 1983.

The Voronoi diagram for n disjoint spherical sites of $O(1)$ different radii has complexity $O(n^2)$ (extended abstract)

Colm Ó Dúnlaing*
Mathematics, Trinity College, Dublin 2, Ireland

Abstract

It is proved that if S is a set of n disjoint spherical sites in $\mathbf{R}^3$, of at most k different radii, then the cell owned by a smallest site has fewer than $3^{k-1}n$ different faces.

It follows that the Voronoi diagram of S has complexity $O(n^2)$, assuming a bound on the number of different radii among the sites in S.

It is also shown that without the bound on the number of different radii, the cell owned by a point site can have complexity $\Omega(n^2)$.

1 Voronoi diagrams

This paper considers the Voronoi diagrams of spherical sites in $\mathbf{R}^3$. For a general survey of Voronoi diagrams see, e.g., [1]. The current state of knowledge about the complexity of Voronoi diagrams, in 3 dimensions, is scanty. It is known to be $O(n^2)$ for n point sites, and this bound is tight. When the sites are straight lines, the complexity is known to be $o(n^{2+\epsilon})$ for all $\epsilon > 0$, granted that either the distance function is polyhedral, based on a fixed convex polyhedron [2], or the distance is Euclidean but the lines are in $O(1)$ different directions [3].

The so-called *sites* will be a set S of n disjoint closed balls in $\mathbf{R}^3$. The *Voronoi diagram* of S is the set of points in $\mathbf{R}^3$ which have more than one site closest to them. The Voronoi diagram is a 2-dimensional complex with faces, edges, and vertices. The faces are connected subsets of what we call *bisectors*.

1.1 Definition. Let B and B' be disjoint spherical sites (point sites are allowed), Then the (B, B')-*bisector* is the Voronoi diagram of $\{B, B'\}$, that is, the set of points equidistant from B and B'.

1.2 Lemma. *If B and B' are spherical sites with radii r, r' respectively, where $r \geq r' \geq 0$, then the bisector of B and B' is a plane if $r = r'$ and a (single sheet of a 2-sheeted) hyperboloid of revolution, whose axis is the line joining their centres, if $r > r'$.*

In either case, the bisector partitions $\mathbf{R}^3$ into two regions, and that containing B' is convex. See Figure 1. □

1.3 Corollary. *Suppose that S is a set of disjoint spherical sites whose minimum radius is r_1. Let S' be the set of spherical sites obtained by replacing every site B in s by a site with same centre and radius $r - r_1$, where r is the radius of B.*

Then $\mathrm{Vor}(S) = \mathrm{Vor}(S')$, and the smallest sites in S' are point sites. □

1.4 Definition. Let B be one of the sites in a set S of disjoint spherical sites. The *Voronoi cell* of B consist of all points which are as close, or closer to, B than to any other site in S.

*e-mail: odunlain@maths.tcd.ie. Mathematics department website: http://www.maths.tcd.ie.

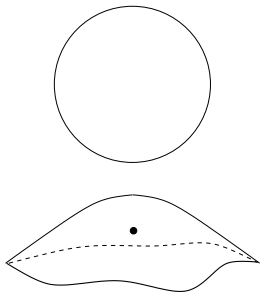


Figure 1: The bisector is a hyperboloid of revolution.

Clearly, the Voronoi diagram is the union of boundaries of cells of all sites in S.
The complexity of the Voronoi diagram for point sites is known:

1.5 Proposition. *If S is a set of n point sites, then* $\mathrm{Vor}(S)$ *has n cells and $O(n^2)$ faces, edges, and vertices. This bound is tight.* □

1.6 Lemma. *Let $\mathcal{C}$ be a collection of sets S of disjoint spherical sites in $\mathbf{R}^3$. Let $M(n)$ denote the maximum complexity of* $\mathrm{Vor}(S)$ *for all $S \in \mathcal{C}$ such that $|S| = n$. Then $M(n)$ is $O(n^2)$ if and only if for every $S \in \mathcal{C}$,* $\mathrm{Vor}(S)$ *contains a cell of complexity $O(|S|)$.* □

2 Re-inflating deflated sites

It is our aim to show that when S is a set of disjoint spherical sites with at most k distinct radii, and B is a site of minimum radius in S, then its cell in $\mathrm{Vor}(S)$ has at most $3^{k-1}n$ faces. With k fixed it can be regarded as having $O(n)$ different faces, and hence its complexity is $O(n)$. This is enough (Lemma 1.6) to ensure that $\mathrm{Vor}(S)$ has complexity $O(n^2)$, when the number of different radii occurring among the sites in S is bounded.

(2.1) Inflating sites. We imagine the sites being 'inflated' to their correct size: an increasing parameter r is given, and $S(r)$ is the set of sites with their radius bounded by r. As r increases, the sites inflate until all have reached their correct radius. We study how the Voronoi diagram evolves.

2.2 Definition. Let S be a set of n spherical sites with centres c_i and radii r_i. For any $r \geq 0$, the *r-bounded version* $S(r)$ of S is the set of n sites whose centres are c_i but whose radii are $\min(r_i, r)$.

We can assume (Corollary 1.3) that S contains a point site. For the remainder of this section, p will denote a point site in S.

2.3 Definition. Suppose that B is a site in $S(r)$. If the corresponding site in S has radius $> r$ then we say B is *expanding*, otherwise it is *stable*.

A face, edge, or vertex of $\mathrm{Vor}(S(r))$ is called *stable, transient static, or moving* according as all sites closest to it are stable, all are expanding, or some but not all are expanding, respectively.

2.4 Definition. $C(r)$ will denote the cell owned by p in $\mathrm{Vor}(S(r))$.

2.5 Lemma. *$C(r)$ is convex. (Immediate from Lemma 1.2.)* □

We consider the evolution of $C(r)$ as r increases (up to the maximum radius occurring in S).

$C(0)$ is a convex polyhedron with at most $n-1$ faces. As r increases, some of these faces become curved, and new faces appear and disappear.

2.6 Lemma. *The only way a new face can be introduced to $C(r)$ is when a bisector passes through a stable or transient static vertex of $C(r)$.*

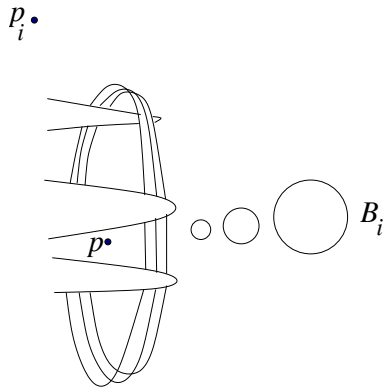


Figure 2: Cell of p has $\Omega(n^2)$ incident edges.

Sketch proof. We must consider all possible ways in which the number of faces of $C(r)$ can change. We classify them as follows:

(A) Several cases which turn out to be impossible:

- (Ai) A face gets separated when two opposite edges touch.
- (Aii) A face gets introduced when two bisectors touch.
- (Aiii) A face gets introduced when a bisector touches an edge, and the face begins to separate the edge.

(B) A bisector passes through a moving vertex.

(C) A bisector passes through a transient static vertex.

(D) A bisector passes through a stable vertex.

In case (B) a face disappears, and cases (C) and (D) are as predicted. □

2.7 Corollary. *$C(r)$ has at most 3^{k-1} faces.*

Sketch proof. Let

$$0 = r_1 < r_2 < \ldots < r_k$$

be the different radii occurring among the sites in S. Whenever $C(r)$ acquires a new face, a vertex was lost under case (C) or (D). Suppose $r_s \leq r < r_{s+1}$. If case (D) applied, then that vertex existed in $C(r_s)$, and we can assume by induction that there are at most $2 * 3^s$ vertices in $C(r_s)$. If the vertex did not exist in $C(r_s)$, then the vertex must have been introduced at some time r', $r_s < r' < r$, in an event of type (B). But then a face was lost from $C(r')$, which can be offset against the gain of a new face by $C(r)$ through a type (C) event. □

3 The bound on number of radii is essential

Let p be a point site located at $(0, 0, 0)$. Let H be the unit sphere centred at $(0, 0, 1)$. H touches p. Place n balls B_j centred on the x-axis at $(1/2^j, 0, 0)$ and tangent to H: they are disjoint. Place n point sites p_i around the circle $x = 0, y^2 + z^2 = 4$. The points p, p_i and the balls B_j form a set S of $2n + 1$ sites.

The circle $E: \ x = 0, y^2 + z^2 = 1$ is a degenerate edge where the cell of p in $\mathrm{Vor}(S)$ meets those of the p_i and the B_j.

Slightly expand the sites B_j, and displace the point sites p_i slightly towards p. The effect is to replace E by $n - 1$ circles close to E, and the point sites p_i split these $n - 1$ circles into n edges each. The cell of p has more than $n(n - 1)$ incident edges. The idea is illustrated in Figure 2.

4 References

1. Franz Aurenhammer (1990). Voronoi diagrams— A survey of a fundamental geometric data structure. *ACM computing surveys* **23.3**, 345–405.

2. L. Paul Chew, Klara Kedem, Micha Sharir, Boris Tagansky, and Emo Welzl (1998). Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *Journal of Algorithms 29*, 238–255.

3. Vladlen Koltun and Micha Sharir (2002). Three dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. *Proc. 18th European Workshop on Computational Geometry*, 1–3.

The anchored Voronoi diagram: static and dynamic versions and applications

J. A. Barcia[*] *J. M. Díaz-Báñez*[†] *F. Gómez*[‡] *I. Ventura*[§]

Abstract

Given a set S of n points in the plane and a fixed point o, we introduce the Voronoi diagram of S anchored at o. It will be defined as an abstract Voronoi diagram that uses as bisectors the following curves. For each pair of points p, q in S, the bisecting curve between p and q is the locus of points x in the plane such that the line segment $\overline{ox}$ is equidistant to both p and q. We show that those bisectors have nice properties and, therefore, this new structure can be computed in $O(n \log n)$ time and $O(n)$ space. Also, under a slightly different model of computation, we prove that the dynamic version of this diagram can be built in $O(n^2 \lambda_{6s+2}(n))$ time complexity, where s is a constant depending on the function that describes the motion of the points. Both static and dynamic diagrams can be used for solving maximin location problems, where the goal is the placement of a line segment connecting two fixed curves.

1 Introduction

Given a set of n sites in a continuous space, the subdivision of the space into regions, one per site, according to some influence criterion is a central topic in Computational Geometry and it has been applied to many fields of science. The standard name for this geometric structure is due to Voronoi, who proposed the first formalization. Originally, this structure was used for characterizing regions of proximity for the sites. Since then, mMany extensions and generalizations have been proposed (see the surveys [1, 5, 9]). Also, other general approachs have been introduced [4, 8] where the concepts of site or distance functions are not explicitly used. In this paper, we introduce an abstract Voronoi diagram in the sense of [8], *the anchored Voronoi diagram*. In section 2, we formally define this structure, give some properties and show how to compute it; we also give an application to a facility location problem. In section 3, we deal with the dynamic version of the anchored diagram; in particular, we discuss the topological matters that lead to its construction and, finally, we show how to apply this structure to solving some maximin problems. Those problems consist of finding the bridge that connects to curves so that the minimum distance from the bridge to a given point set is maximized. Concluding remarks of the paper are put forward in Section 4 .

2 The anchored Voronoi diagram

2.1 Definition and properties

Given a set S of n points in the plane, the Euclidean distance between two points p and q will be denoted by $d(p, q)$. We define an *anchored* segment as a line segment when the initial point is fixed. Without loss of generality, we will consider the anchor to be the origin. Finally, the distance between a point p and an anchored segment connecting o with a point $x \in I\!R$ will be defined as $d(p, \overline{ox}) := \min\{d(p, q) : q \in \overline{ox}\}$.

[*]Departamento de Matemática Aplicada II , Universidad de Sevilla (jbarcia@us.es)
[†]Departamento de Matemática Aplicada II , Universidad de Sevilla (dbanez@us.es)
[‡]Departamento de Matemática Aplicada I , Universidad Politécnica de Madrid (fmartin@eui.upm.es)
[§]Departamento de Matemáticas, Universidad de Huelva (iventura@us.es)

For any two different points p, q in S, a *bisecting curve* $L(p,q)$ is defined as the locus of points x in the plane such that the line segment $\overline{ox}$ is equidistant to both p and q, that is, $L(p,q) = \{x \in I\!R^2 : d(p,\overline{ox}) = d(q,\overline{ox})\}$. An exhaustive study of the properties and the shape of $L(p,q)$ have been carried out in [2]. $L(p,q)$ is homeomorphic to a line and dissects the plane into two open domains $D(p,q)$ and $D(q,p)$ having $L(p,q)$ as boundary. We define the *Anchored Voronoi Region* $AVR(p,S)$ to be the intersection of the domains $D(p,q)$, where $q \in S \setminus \{p\}$. Then the *Anchored Voronoi Diagram* $AVD(S)$ of the bisecting curves $L(p,q)$ is defined as the union of all boundaries of at least two Voronoi region have in common. In Figure 1, all types of bisecting curves are shown. Note that in the case (a.3) $L(p,q)$ includes a region. In order to simplify the discussion and for this degenerate situation,we will take $L(p,q) = \{x \in I\!R^2 \mid d(p,x) = d(q,x)\}$ as the bisector of the line segment $\overline{pq}$.

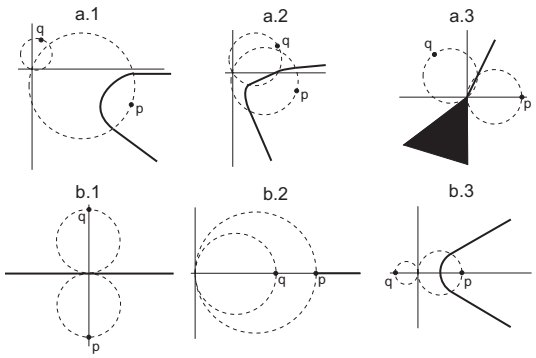


Figure 1: The locus $L(p,q)$.

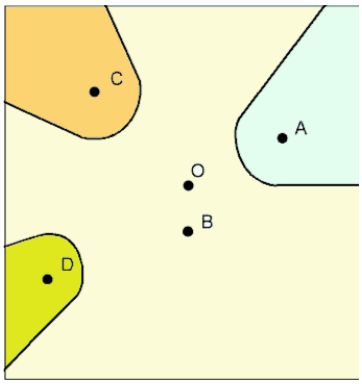


Figure 2: $AVD(S)$ for four points.

An edge can be composed into pieces which are either half-lines, or line segments, or arcs of a curve of degree four or arcs of a circle (refer to Figure 2). A vertex of $AVD(S)$ (defined by at most three points) can either be a point or a half-line or an arc of circle. Note that vertices are defined as the intersections of the Voronoi edges.

2.2 Computation

In order to compute the $AVD(S)$, the divide & conquer approach given in [8] can be used. The system $\mathcal{L} = \{L(p,q) : p, q \in S, p \neq q\}$ is called *admissible* iff for each subset S' of S of size at least 3 the following conditions are fulfilled: (1) the Voronoi regions are path-connected; (2) each point of the plane lies in a Voronoi region or on the Voronoi diagram; (3) the intersection of two bisecting curves only consists of finitely many components. By using non-trivial geometrical properties, we have proved the following results.

Lemma 2.1 *The set of locus $\mathcal{L}$ is an admissible system.*

Theorem 2.1 *The Anchored Voronoi Diagram of a set of point S in the plane can be constructed in $O(n \log n)$ time and $O(n)$ space.*

2.3 Application

We next show how to use the $AVD(S)$ as a data structure for computing a solution for a location maximin problem. The *obnoxious anchored bridge* problem is stated as follows:

OABP: *Let S be a set of n points in $I\!R^2 \setminus \{o\}$ and let $\mathcal{C}$ be a curve (typically, in most applications, an algebraic curve of constant degree). Compute a line segment connecting o with a point x on $\mathcal{C}$ for which $\min_{p \in S} d(p,\overline{ox})$ is maximized.*

Typically, in most applications, curve $\mathcal{C}$ will be an algebraic curve of constant degree, a trigonometric function or similar.The following results solve this problem.

Lemma 2.2 *There exists a point x^* which is the intersection between the curve $\mathcal{C}$ and the structure $AVD(S)$ such that the segment $\overline{ox^*}$ is a solution for the problem OABP.*

Theorem 2.2 *Once the $AVD(S)$ is given, the problem OABP can be solved in linear time and space.*

At this point, we should note that there are certain operations here that exceed the power of the usual real RAM model. The model of computation should be augmented with the pertinent primitives as neccesary.

3 The dynamic anchored Voronoi diagram

We are given a finite set of $n \geq 3$ points $S = \{p_1, \ldots, p_n\}$ each moving along a polynomial trajectory of maximum degree s, for some constant s. Let $p_i(t)$ denote the position of point p_i at time t. We further assume that the points move without collisions.

Definition 3.1 *Given an anchored segment S of length l and $\varepsilon \geq 0$, the locus of points that are at distance ε from S is called an anchored hippodrome centered at S of radius ε.*

In our context, we consider the points in general position when no four points are *co-hippodromal*, in other words, there not exists a hippodrome with four points on the boundary. We study how the structure $AVD(S(t))$ changes with time. Similarly to the ordinary dynamic Voronoi diagram [6], $AVD(S)$ changes continuously but its combinatorial structure only changes at critical values of t. We will call the dual graph of $AVD(S(t))$ *the anchored graph* $AG(S(t))$.

3.1 Topological changes

In order to obtain a bound of the number of changes in $AG(S(t))$ we describe how an edge can be removed or added to the graph as the points move. In the following, we characterize these elementary changes.

It follows from the definition of the anchored dual graph that there is an edge between two points if and only if there exists an empty hippodrome that passes through those points (The converse is also true.) Let $p_i(t)$, $p_j(t)$ be two points in $S(t)$ and let be given a hippodrome that have them on its boundary. This hippodrome determines a unique line segment, one of whose endpoints is the origin and the other is a point $x_{ij}(t)$. Of course, $x_{ij}(t)$ lies on the Voronoi edge contained in the bisector curve $L(p_i(t), p_j(t))$. Let $d_{ij}(t)$ denote the distance from point $p_i(t)$ to line segment $\overline{ox_{ij}(t)}$. When a point $p_k(t)$ enters into an empty hippodrome given by points $p_i(t)$, $p_j(t)$, then a combinatorial change takes place. Such change will correspond to an intersection between function $d_{ij}(t)$ and another function $d_{ik}(t)$ or $d_{jk}(t)$. What really matters here is when the first point that enters into the empty hippodrome, which results in only considering the lower enveloppe of the functions $\{d_{ij}(t), i \neq j\}$. By examining those intersections, we can give an upper bound on the number of topological changes.

We can see that any pair of functions $\{d_{ij}(t), i \neq j\}$ intersects at most $6s$ times. Hence, the number of breakpoints of the lower envelope of the distance functions is $O(\lambda_{6s+2}(n))$. By repeating this argument for all pair of points in $S(t)$, we obtain the desired upper bound. We thus conclude with the following theorem.

Theorem 3.1 *The number of topological changes of the combinatorial structure of $AVD(S(t))$, when each point in S moves along a trajectory defined by polynomial of maximum degree s, is $O(n^2\lambda_{6s+2}(n))$.*

3.2 Computing the diagram

The topological structure of an anchored Voronoi diagram under continuous motions of the points in S can be maintained dynamically. By using a similar approach to those in [3, 7], we are able to update the changes in $O(\log n)$ time.

Theorem 3.2 *The dynamic $AVD(S)$ can be constructed in $O(n^2\lambda_{6s+2}(n)\log n)$ time.*

3.3 Application

With the dynamic anchored Voronoi diagram we can solve a general problem for locating an obnoxious bridge. Consider the problem of Section 2.3 but we suppose the anchor point o can be moved through a polynomial trajectory. The problem now is:

OBP: *Let S be a set of n points in $\mathbb{R}^2$, $\mathcal{P}$ be a polynomial curve and $\mathcal{C}$ a curve. Compute a line segment l connecting $\mathcal{P}$ with $\mathcal{C}$ for which $\min_{p\in S} d(p,l)$ is maximized.*

Observe that we can solve this problem by fixing an endpoint of the segment l on a point of $\mathcal{P}$ and moving the points in S along such a trajectory. Recall that the $OABP$ problem can be solved by finding the Voronoi vertices for which the segment is the center of the largest empty hippodrome. Then, as t varies, we maintain for each Voronoi vertex the starting time t_0 at which appears and also the time t_1 at which disappears. Between t_0 and t_1, we compute when the width of the corresponding hippodrome is maximized. Finally, by keeping track of those maximum values, the general problem OBP can be solve within the time obtained for the computation of the dynamic anchored Voronoi diagram.

4 Conclusion

We have introduced in this paper the anchored Voronoi diagram as an abstract Voronoi diagram. The bisecting curves are induced by the distance to a line segment anchored at the origin. Also, for the dynamic version we have found an upper bound based in a Davenport-Schinzel argument. We have finished by showing an application, namely, solving a maxmin bridge problem.

References

[1] F.Aurenhammer, Voronoi diagrams: A survey of a fundamental geometric data structure, *ACM Comput. Surv.* , 23, 1991, 345-405.

[2] J.A. Barcia, J.M. Díaz-Báñez, A. Lozano and I. Ventura, Computing an obnoxious anchored segment, to appear in *Operations Research Letters*, 2003.

[3] L.P. Chew, K. Kedem, A convex polygon among polygonal obstacles: placement and high-clearance motion. *Computational Geometry: Theory and Applications*, 3 , 1993, 59-89.

[4] H. Edelsbrunner and R. Seidel, Voronoi diagrams and arrangements, *Discrete Computational Geometry*, 1, 1986, 25-44.

[5] S. Fortune, Voronoi diagrams and Delaunay trangulations, In Computing in Euclidean Geometry, D.-Z. Du and F.K. Hwang, eds, Lectures Notes Series on Comput. 1, World Scientific, Singapore, 1992, 193-233.

[6] L. Guibas, J.S.B. Mitchell and T. Roos, Voronoi Diagrams over Dynamic Scenes. *Lecture Notes in Computer Science*, 570, 1991, 113-125.

[7] K. Imai, H. Imai, T. Tokuyama, Maximin location of convex objects in a polygon and related dynamic Voronoi diagrams. *Journal of the Operations Research*, 42, 1999.

[8] R. Klein, Concrete and Abstract Voronoi Diagrams over Dynamic Scenes. *Lecture Notes in Computer Science*, 400, 1989.

[9] A. Okabe, B. Boots and K. Sugihara, Spatial tessellations: concepts and applications of Voronoi diagrams, Wiley, Chichester, UK, 1992.

Approximating planar subdivisions and generalized Voronoi diagrams from random sections

Narcís Coll * Ferran Hurtado † J.Antoni Sellarès ‡

Abstract

We present an algorithm for constructing from a set of sampled sections a piecewise-linear approximation of an unknown planar subdivision, including generalized Voronoi diagrams as outstanding example. The input of the algorithm is a set of lines uniformly distributed over the theoretical subdivision. For each input line, the ordered set of sections in which the line and the planar subdivision intersect is given or computed. The algorithm outputs a triangulation from which the approximation of the unknown subdivision, both in the topological and the metrical sense, can easily be extracted. The correctness of the algorithm and the evaluation of its time complexity follow from results of Integral Geometry and Geometric Probability.

1 Introduction

We present an algorithm that allows to construct a piecewise-linear approximation of an unknown planar subdivision from a set of sampled line-sections. The input of our algorithm is a sufficiently "dense" set of lines uniformly distributed over a bounding box together with the ordered set of sections in which each line intersects the theoretical planar subdivision. We want to remark that, in the context of integral geometry and geometric probability, uniformly distributed means that the probability that a line intersects a piece of the boundary of and edge, independent of its location and orientation, is proportional to its length [17]. The algorithm outputs a triangulation from which a planar subdivision approximating the unknown planar subdivision can easily be extracted. The input lines are sequentially processed and the method is progressive, in the sense that for each line we obtain a new approximation from the previous one.

Some of the ideas used in our algorithm extend previous work on reconstructing planar shapes from random sections as a problem related to curve reconstruction [6]. Different cases of the curve reconstruction problem have been treated: uniformly or non-uniformly sampled points, closed or open curves, smooth or non-smooth curves and many algorithms have been proposed for all these cases [2, 3, 4, 7, 8, 9, 10, 11, 13, 14, 15]. The main difference between these algorithms and ours is that all them compute the reconstruction off-line, while our algorithm works on-line. As mentioned above, the fact that the random lines are sequentially processed as they arrive allows also progressive refinement that may be tuned by the user.

The method can be applied to the construction of planar subdivisions, and in particular to generalized Voronoi diagrams, that correspond to two-dimensional scenarios for which the whole

*Institut d'Informàtica i Aplicacions, Universitat de Girona, Girona, España (coll@ima.udg.es). Partially supported by grants TIC2000-1009, TIC2001-2226-C02-02 and DURSI 2001SGR-00296

†Departament de Matemàtica Aplicada II, U.P.C, Barcelona, España (hurtado@ma2.upc.es). Partially supported by MEC-DGES-SEUID PB98-0933, MCYT-FEDER BFM2002-0557 and DURSI 2001SGR00224

‡Institut d'Informàtica i Aplicacions, Universitat de Girona, Girona, España (sellares@ima.udg.es). Partially supported by grants TIC2000-1009, TIC2001-2226-C02-02 and DURSI 2001SGR-00296

structure is difficult to obtain, but such that its intersection with any given line is easy to compute or to obtain with some device.

Voronoi diagrams are a fundamental structure in computational geometry, both for the rich theory they embody as for the impressive variety and number of applications they have. Many variants have been considered: by taking sites of different shape or nature, associating weights to the sites, changing the underlying metrics, or using individualized distance functions for the sites. Classic and generalized Voronoi diagrams are described in the surveys [1, 16].

The algorithms designed for computing exact generalized planar Voronoi diagrams often have numerical robustness problems and are time-consuming due to the numerous high precision calculations that are required. However in some applications (like motion planning or geographic map simplification) the computation of an approximated Voronoi diagram within a predetermined precision is sufficient, and several algorithms have been proposed for the approximation of Voronoi diagrams [5, 12, 18, 19]. We present an algorithm for approximating generalized planar Voronoi diagrams for different site shapes (points, line-segments, curve-arc segments, ...) and different distance functions (Euclidean metrics, convex distance functions, ...). The algorithm is robust and fast (in terms of running time). Moreover it is very general: not all the sites must be homogeneous in shape or have associated the same distance function, and there are no restrictions on the connectivity of the Voronoi regions –for a single site they may have several components–, the degree of the Voronoi vertices, or the dimensionality of the bisectors.

2 Sketch of the method

We assume that the planar subdivision $\mathcal{P}$ to be approximated is contained in a tight axis-parallel bounding-box K. The algorithm takes as input a set L of m lines uniformly distributed over K. For each line l of L we have the ordered set $S(l)$ of intersections between l and the regions of $\mathcal{P}$. The algorithm outputs a triangulation $T(\mathcal{P})$ of K represented by a DCEL structure. Each triangle of the $T(\mathcal{P})$ is assigned to one of the regions of $\mathcal{P}$ or to an auxiliary region that we call background. From this triangulation it can easily be obtained, in cost linear with respect to the number of triangles, the piecewise-linear approximation $A(\mathcal{P})$ of $\mathcal{P}$.

The main part of the algorithm processes the sections of the lines of L sequentially. When a new section is considered, the triangles that contain the endpoints of the section are subdivided, and a region determined by a subset of the triangles crossed by the section is retriangulated in order to connect two vertices by an edge. The DCEL structure is actualized properly. To describe the general idea followed to design the algorithm, we explain the particular case of the five sections of Figure 1. First we introduce section a as an edge. Since section b intersects section a, a triangulated quadrilateral that connects the endpoints of section b with the endpoints of section a is created. As section c do not intersect the quadrilateral, then it is inserted as an edge. Since section d intersects section c, a second triangulated quadrilateral is also created. Section e intersects the two quadrilaterals, then the endpoints of the section are connected with the quadrilaterals, and the edges of the quadrilaterals crossed by the section are also connected. In order to guarantee the desired cost of the algorithm, we maintain the length of the new edges produced by the subdivision of the triangles inversely proportional to the number of processed lines.

We can prove, using results of Integral Geometry and Geometric Probability, that by taking the number of lines m of L large enough the piecewise-linear subdivision $A(\mathcal{P})$ obtained with our algorithm tends to the planar subdivision $\mathcal{P}$ in both the topological and the metrical sense, and we can show also that the mean computational cost of the algorithm is $O(m \log m)$.

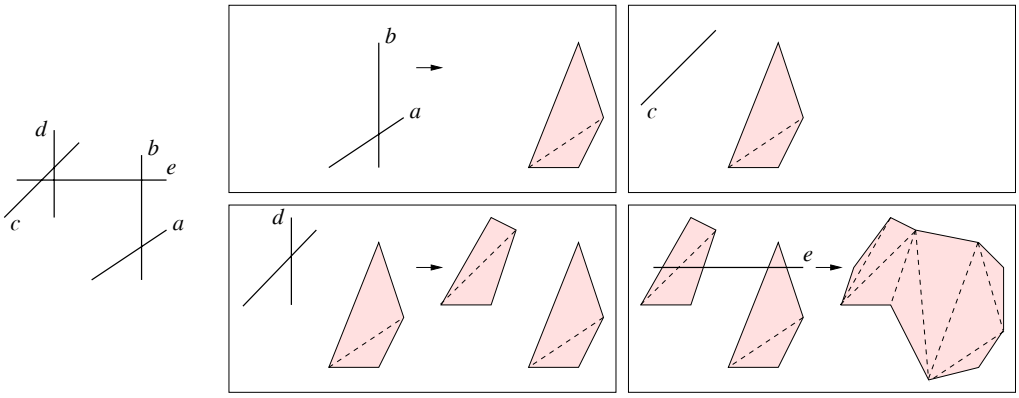


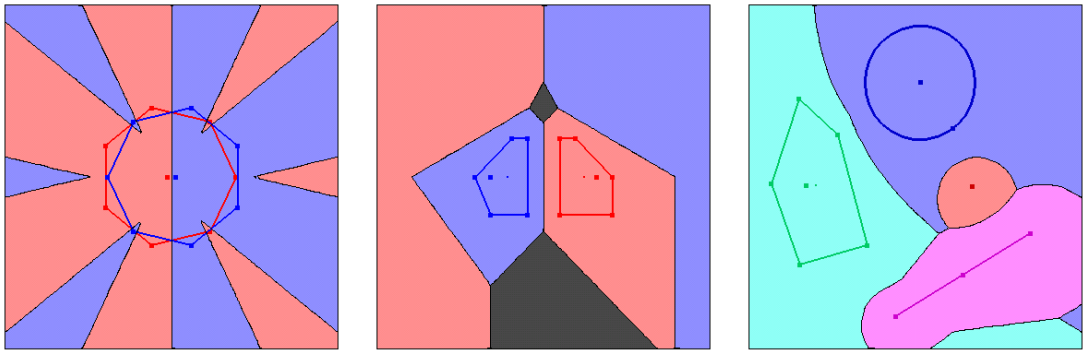
Figure 1: *Triangulation process*

3 Voronoi diagram approximation

Our algorithm can be used in order to approximate generalized planar Voronoi diagrams for different site shapes (points, line-segments, curve-arc segments, ...) and different distance functions (Euclidean metrics, convex distance functions, ...). The resulting approximation algorithm is very general: not all the sites must be homogeneous in shape or have associated the same distance function, and there are no restrictions on the connectivity of the Voronoi regions, which for a single site may have several components, the degree of the Voronoi vertices or the dimensionality of the bisectors.

The total cost of approximating the generalized Voronoi diagram of n sites inside a bounding-box K, using m lines uniformly distributed on K, is $O(mn \log n + m \log m)$.

We have implemented both the general algorithm and its particularization to Voronoi diagrams. An example of the results is shown in the next set of figures which has been obtained using convex distance functions. Figure 2.a shows the diagram of two sites with non connected Voronoi regions. Figure 2.b shows the diagram of two sites with a two dimensional bisector. The darkest regions that appear in the image represent the two dimensional parts of the bisector. Figure 2.c shows the diagram obtained using different site shapes and different distances: a point and a segment with the Euclidean metric, a point with a quadrilateral as associated convex shape, and a point with a non-unit circle as associated convex shape.



(a) *Non connected regions.* (b) *Two dimensional bisector.* (c) *Different distances.*

Figure 2: *Convex distance*

References

[1] F. Aurenhammer and R. Klein. *Voronoi diagrams. In Handbook of Computational Geometry. J.-R. Sack, J. Urrutia, editors.*

[2] E. Althaus and K. Mehlhorn, Polynomial time TSP-based curve reconstruction. *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pp 686-695, (2000).

[3] N. Amenta, M. Bern and D. Eppstein, The crust and the β-skeleton: combinatorial curve reconstruction. *Graphic Models and Image Processing* 60, pp 125-135, (1998).

[4] D. Attali, r-regular shape reconstruction from unorganized points. *Proc. 13th ACM Symposium on Computational Geometry*, pp 248-253, (1997).

[5] I. Boada, N. Coll and J.A. Sellarès, Hierarchical Planar Voronoi Diagram Approximations. *Proc. 14th Canadian Conference on Computational Geometry 2002*, pp 40-45,(2002).

[6] N. Coll and J.A. Sellarès, Planar Shape Reconstruction from Random Sections. *Proc. 17th European Workshop on Computational Geometry CG 2001* , pp 121-124, (2001).

[7] T.K. Dey and P. Kumar, A simple provable algorithm for curve reconstruction. *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pp 893-894, (1999).

[8] T.K. Dey, K. Mehlhorn and E.A. Ramos, Curve Reconstruction: Connecting Dots with Good Reason. *Computational Geometry Theory Appl.* 15, pp 229-244, (2000).

[9] T.K. Dey and R. Wenger, Reconstruction Curves with Sharp Corners. *Proc. 16th ACM Symposium on Computational Geometry*, pp , (2000).

[10] H. Edelsbrunner, Shape reconstruction with the Delaunay complex. *LATIN'98: Theoretical Informatics, Lecture Notes in Computer Science*, v. 1380, pp 119-132, (1998).

[11] H. Edelsbrunner, D.G. Kirkpatrick and R. Seidel, On the shape of a set of points in the plane. *IEEE Trans. Information Theory* 29, pp 71-78, (1983).

[12] K. Hoff, T. Culver, J. Keyser, M. Lin, D. Manocha, Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. *Proc. SIGGRAPH'99, ACM Press*, pp 277-286,(1999).

[13] J. Giesen, Curve reconstruction, the TSP, and Menger's theorem on length. *Proc. 15th ACM Symposium on Computational Geometry*, pp 207-216, (1999).

[14] C. Gold and J. Snoeyink, Crust and anti-crust: a one-step boundary and skeleton extraction algorithm. *Proc. 15th ACM Symposium on Computational Geometry*, pp 189-196, (1999).

[15] M. Melkemi, A-shapes of a finite point set. *Proc. 13th ACM Symposium on Computational Geometry*, pp 367-369, (1997).

[16] A. Okabe, B. Boots, K. Sugihara and S. N. Chiu, *Spatial Tessellations: Concepts and Application of Voronoi Diagrams*, John Wiley & Sons, (2000).

[17] L.A. Santalò, *Geometric Probability*, Society for Industrial and Applied Mathematics, (1976).

[18] M. Teichmann and S. Teller, Polygonal approximation of Voronoi diagrams of a set of triangles in three dimensions. *Technical Report 766*, Laboratory of Computer science, MIT,(1997).

[19] J. Vleugels and M. Overmars, Approximating Generalized Voronoi Diagrams in Any Dimension. *International Journal on Computational Geometry and Applications*, 8:201-221,(1995).

Sweeping an Arrangement of Quadrics in 3D *

Bernard Mourrain Jean-Pierre Técourt Monique Teillaud

Abstract

1 Introduction

Arrangements are the underlying structure of many applications, especially in robot motion planning. They have been extensively studied in the literature (see [Hal97] for a survey).

The arrangement of a set of objects $\mathcal{S}$ in $\mathbb{R}^d$ is the decomposition of $\mathbb{R}^d$ into cells of dimensions $0, 1, \ldots, d$ induced by $\mathcal{S}$. The topology of an arrangement is often quite complex, and the description of a given cell can be of non-constant size. Therefore, vertical decompositions are often used, allowing to partition the space into simpler constant sized cells (for a complete bibliography, we refer to [SH02]). A sweep-based algorithm was followed in [dBGH96, SH02] to produce a vertical decomposition of an arrangements of triangles in $\mathbb{R}^3$.

The manipulation of algebraic surfaces plays an important role in solid modeling. Geismann *et al.* presented two methods to compute a given cell in an arrangement of quadrics [GHS01]. The first method uses projection techniques based on resultants, while the second method uses solid modeling techniques.

We propose here a sweeping algorithm to compute effectively the arrangement of a set of quadrics in $\mathbb{R}^3$.

2 Overview

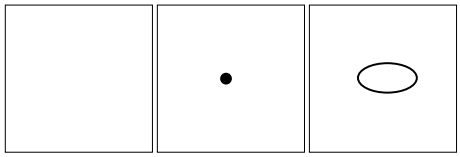
Let $\mathcal{S} = \{Q_i, i = 1, \ldots, n\}$ be a set of n quadrics. We denote by Q_i both a quadric and its equation. Let ∇Q_i be the gradient vector of Q_i. We assume that no quadric is a product of planes.

We choose a generic direction (say z) and we sweep $\mathcal{S}$ by a plane in this direction. Every z-section of the arrangement is an arrangement of conics in the plane. We initialize the sweep at some chosen value of z. Let us consider the different types of events where the topology of the z-section is changing during the sweep.

a) $Q_{i_1} = 0, \partial_x(Q_{i_1}) = 0, \partial_y(Q_{i_1}) = 0$: horizontal tangent plane.

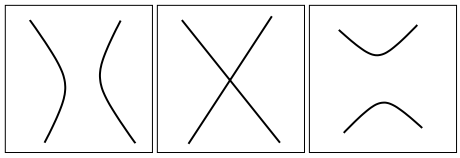
More precisely, depending on the signature of Q_{i_1}, three events can appear:

i. (3,1), (1,3):

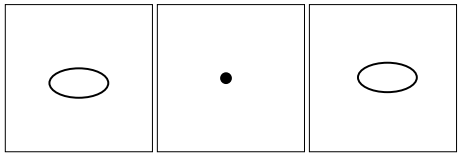


*GALAAD, INRIA, BP 93, 06902 Sophia Antipolis cedex, FRANCE
{Bernard.Mourrain, Jean-Pierre.Tecourt, Monique.Teillaud}@sophia.inria.fr
http://www-sop.inria.fr/galaad/
This work is partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces)
http://www-sop.inria.fr/prisme/ECG/

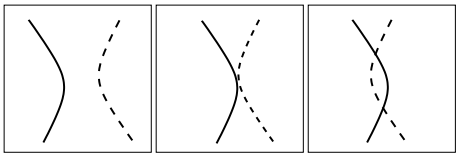
ii. (2,2):



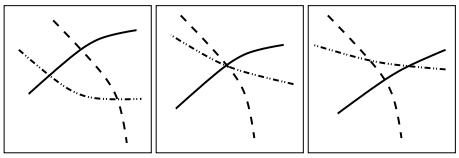
iii. (2,1), (1,2):



b) $Q_{i_1} = 0, Q_{i_2} = 0, (\nabla Q_{i_1} \bigwedge \nabla Q_{i_2})_z = 0$: horizontal tangent for the intersection curve between Q_{i_1} and Q_{i_2}.



c) $Q_{i_1} = 0, Q_{i_2} = 0, Q_{i_3} = 0$: intersection point of three quadrics.

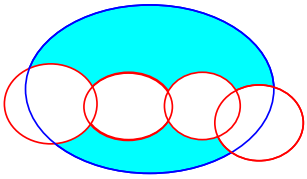


We do not consider degenerate cases such as intersections of more than three quadrics at the same point.

3 From cells to "trapezoids"

The first idea consists in characterizing each 2-dimensional cell of the arrangement of conics in a section by sign conditions. For one or two conics, the sign conditions are determined by the equations of the quadrics and the equations of lines depending on the quadrics. See [MTT02] for more details.

In the case when cells are defined by more than 3 quadrics, the following picture shows that two different cells (the two gray cells) can be characterized by exactly the same sign conditions.



To solve this issue, we choose to compute a "trapezoidal" decomposition of the arrangement in the z-section, as explained in the following paragraph.

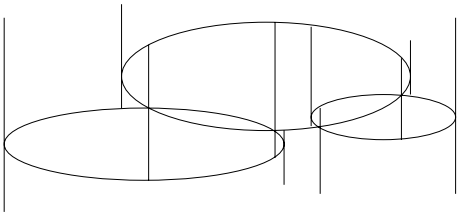
Trapezoids. We draw segments parallel to the y-axis. This is done in a very similar way as done usually for the trapezoidal map in the case of a planar arrangement of line segments. A vertical segment will be drawn through:

- intersection points between two conics

- points where the tangent to the conic is parallel to the y-axis.

We obtain "trapezoids" of constant size description: the boundary of each trapezoid consists into two vertical walls, a ceiling and a floor. Both the ceiling and the floor are conic arcs.

Deciding whether a point lies in a trapezoid, reduces to compare the x-coordinates of the point and the walls and then for a fixed x, to compute the sign of the conics defining the ceiling and the floor or the sign of rational expressions formed on their coefficients.



The drawback is that maintaining the vertical decomposition introduces additional events that have no meaning in the 3D arrangement, but the big advantage of this decomposition is that all events described in Section 2, except events of type (a.i), can be easily detected during the algorithm: each time a new trapezoid is created, we compute the z for which it disappears. All the events of Section 2 are some of these events.

When a trapezoid disappears, the 2D arrangement needs to be updated: the trapezoid is replaced by other trapezoids, and its neighbors are modified, too. Enumerating the different types of trapezoids is quite easy, as well as the way they need to be updated, depending on the type of event that cause them to disappear. Details are omitted in this abstract.

Only events of type (a.i) will be precomputed and sorted. When such an event is encountered, a point location has to be performed. The decomposition into trapezoids allows to locate such a point easily in practice, either in a naive way by testing all the trapezoids, or by walking along a line.

3D decomposition. Another advantage of the trapezoidal decomposition is that it induces a decomposition of the arrangement of quadrics in $\mathbb{R}^3$ into simple regions, that are the regions swept by the trapezoids. The decomposition we get with our method is not quite the same as the so-called *vertical decomposition* [SA95].

We skip the discussion on the combinatorial complexity in this abstract. The data structures used are roughly similar to the ones described in [SH02]. We chose to focus on algebraic aspects.

4 Algebraic aspects

The events of type (a.i) in Section 2 are precomputed by solving algebraic equations of degree 2, and they are sorted.

Location in the trapezoidal map. As written above, deciding whether a point lies in a trapezoid reduces to compute signs of rational expressions in the coefficients of the quadrics, and to compare the x-coordinates of the point and the vertical walls. So, the point location for an event of type (a.i) performs such evaluations of signs at points whose coordinates belong to an algebraic extension of degree at most 2, and comparisons of degree 2 and 4 algebraic numbers.

Detecting and comparing new events. A trapezoid is defined by two vertical walls, a floor and a ceiling. To predict how a trapezoid will disappear, we need to compute when its floor and its ceiling collide, or when its vertical walls coincide.

The worst case, in terms of algebraic degree, is achieved by the events when the x-coordinate of the intersection between two conics coincides with the x-coordinate of the intersection between two other conics. This leads to the computation of points of intersection of 4 quadrics in a space of dimension 4, whose coordinates lie in an algebraic extension of degree at most 16. The coordinates of the intersection points are rational functions of these algebraic numbers.

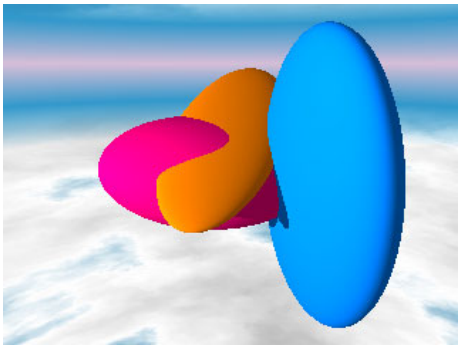
In order to sort the events according to the z-direction, we have to determine the sign of the difference of two algebraic numbers. In the worst case, we are interested in algebraic numbers of degree 16 belonging to independent algebraic extensions of the initial field. So, the difference is in an algebraic extension of degree 256.

Preliminary experimental results. These computations can be done in practice with the SYNAPS library[1]. Let us consider the arrangement of the following 3 quadrics:

$$272x^2 + 96xy + 192xz + 32y^2 + 64yx2 + 64z^2 - 571.2x - 142.4y - 252.8z + 323.64 = 0$$
$$128x^2 + 1152y^2 - 1024yz + 256z^2 - 144x - 886.4y + 358.4z + 220.12 = 0$$
$$64x^2 + 256y^2 + 128z^2 - 64x - 288y - 160z + 143 = 0$$

We have considered the events corresponding to a change in the topology of the cross section (See Section 2). The events corresponding to changes in the trapezoidal map are not computed in these preliminary tests. An approximation of the events (a), (b), (c) is computed with the following running times on a PC workstation (i686, 2.2 GHz, 256 M):
(a) 3×2 real solutions (0.01s).
(b) $3 \times 8 = 24$ complex solutions and 6 are real (0.06s).
(c) 8 complex solutions and 2 real (0.02s).
Then, the events are sorted according to the z-coordinate as follows:

```
(a) [0.825000,0.700000,0.287500]
(a) [0.562500,0.544649,0.359835]
(a) [0.500000,0.562500,0.448223]
(b) [0.498552,0.561349,0.448234]
(b) [0.687835,0.570199,0.508852]
(b) [0.677133,0.617014,0.519616]
(c) [0.676862,0.612181,0.521687]
(c) [0.638126,0.657542,0.685372]
(b) [0.534420,0.666721,0.719519]
(b) [0.662072,0.686211,0.723158]
(b) [0.627783,0.558545,0.776837]
(a) [0.500000,0.562500,0.801777]
(a) [0.562500,0.780351,0.890165]
(a) [0.675000,0.300000,0.912500]
```



References

[dBGH96] M. de Berg, Leonidas J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. *Discrete Comput. Geom.*, 15:35–61, 1996.

[GHS01] N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 264–273, 2001.

[Hal97] D. Halperin. Arrangements. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[MTT02] Bernard Mourrain, Jean-Pierre Tecourt, and Monique Teillaud. Algebraic methods for dealing with 3d implicit quadrics. Technical Report ECG-TR-182105-02, INRIA Sophia-Antipolis, 2002.

[SA95] Micha Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.

[SH02] H. Shaul and D. Halperin. Improved construction of vertical decompositions of three-dimensional arrangements. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 283–292, 2002.

[1] `http://www-sop.inria.fr/galaad/logiciels/synaps/`

Parametric Voxel Geometry Control for Digital Morphogenesis

Thomas Fischer
Spatial Information Architecture Laboratory
Royal Melbourne Institute of Technology
Melbourne, Australia
sdtom@polyu.edu.hk

Torben Fischer
Mathematische Fakultät
Georg-August-Universität Göttingen
Göttingen, Germany
torben.fischer@stud.uni-goettingen.de

1 Abstract

We discuss the extension of an experimental 3D voxel-automata system for generative (evolutionary developmental) design with means for parametric geometry control on the scale of single voxel units. The objective of this extension is to allow the automata system to support the generation of *free form*. Being the result of a longer-term interdisciplinary software development project, the voxel automata system allows simulations of three-dimensional, computation-universal cellular structures and behaviours based on decentralised, massively parallel, programmable units. It represents a kit of parts in which virtual form, programme and data structure are fused. From this top-down perspective we demonstrate a number of geometric operations we have identified and show some early results. We also give an outlook into more challenging future developments.

2 Zellkalkül

The software is named *Zellkalkül* in reference to Konrad Zuse's *Plankalkül* and his early reflections on spatial calculation. It is designed to facilitate explorations of tempo-spatial mechanisms of morphogenesis in cellular (voxel-based) architectural and design contexts such as theoretical architectural design research [3] as well as in generative design teaching [2]. *Zellkalkül* differs from classic cellular automata systems in terms of its programming logic as well as geometrically. Its programming logic supports non-uniform high level coding. That is, different cells can be equipped with individual code scripts. The scripting language used is an extended version of ECMAScript. Besides its general control structures and data types, which are commonly known from other ECMAScript dialects such as JavaScript, our system also provides purpose-centered functions and objects to support intercellular communication and developmental actions. These are partially inspired by biological epigenetic processes (splitting, differentiating, moving, dying) and partially intended to support generic actions (evaluating fixed identities, exchanging and modifying code scripts etc.). The cellular voxel units are based on rhombo-dodecahedral geometry in close-packing arrangement.

We prefer this topology, derived from face-centered cubic close-packing of spheres, over the square or cubic arrangement of common 2D or 3D cellular automata systems since it allows equal distances and relationships between all neighbouring cells (as discussed and exemplified in [4]). Moreover, it bears a close resemblance to many natural cell tissues as identified and illustrated early by [6] (see left of figure 1). The arrangement is equivalent to the *isotropic vector matrix*, which, used for example to form so-called *octet trusses*, is of special relevance to architecture and structural engineering (see [5], p.138 ff.). At the time of this writing, *Zellkalkül* supports 3D rendering of this data structure in form of "solid" spheres and as rhombic dodecahedra. Vector

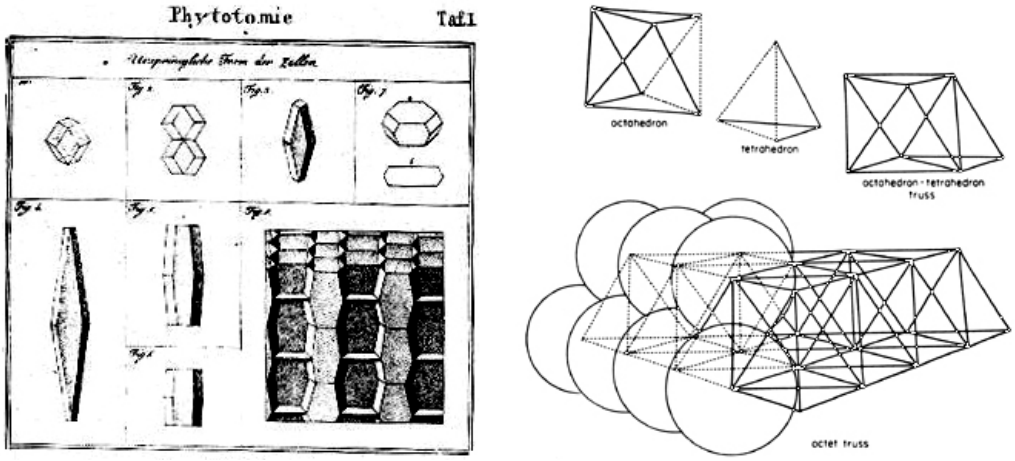


Figure 1: Rhombo-dodecahedral cell geometry by [6] (left) and octet truss by [5] (right)

representation similar to the illustration on the right of figure 1 is planned.

3 Towards Supporting *Free Form*

We believe that *Zellkalkül*'s geometry is more flexible than traditional square or cubic cellular automata systems. However, from an architectural design viewpoint, this current spatial/cellular automata structure still shows a particular shortcoming with respect to its formal expressive capabilities. Geometrically, generated tissues are largely restrained to the system's homogeneously close-packed lattice structure and generate "jagged" forms only. Support of forms with smooth surfaces, straight edges and so forth (or architecturally speaking: *free form*) is being developed at this point and it is the purpose of this paper to explore possible solution strategies. Our interest is to complement customary surface and solid modelling techniques with a different operational mode that embraces developmental morphogenesis in form finding. Approaches that could in principle qualify to allow the generation of free forms in such a way include:

1. Using large numbers of automata at a high resolution to approximate smooth 3D shapes. One disadvantage of this approach is that smooth surfaces are not achieved, only approximated at the cost of exponentially increasing memory consumption during shape generation. Formal expression is moreover achieved primarily by means of additive composition and not by parametric control of geometric relations. The usefulness of both parametric design principles in combination has been discussed by [7].

2. "Skinning" of cell assemblies using curve fitting algorithms.

3. "Skinning" using cell centre points or cell attributes to control skin geometry, e.g. mapping cell location or other cell-related data onto free curve control-points.

4. Mapping of data generated in *Zellkalkül*'s automata structure onto secondary output geometries. This and the above two strategies however contradict the software's intention to provide a single unified geometric and data structure.

5. Parametric position control of cell polygon vertices. We will discuss this approach in most of the remaining part of this paper.

6. Parametric cell sizes control. This appears to a highly interesting future extension. We have however not yet been able to identify suitable operations and constraints for this approach (see section 5.2).

7. Changing the distances between cell centres, i.e substituting the isotropic vector matrix by an "anisotropic vector matrix". This approach is a combination of the above two approaches in which tissues are represented in form of vector trusses instead of as "solid" cells.

Our first approach towards parametric manipulation of the fourteen cell vertices involves the following initial steps. First of all, the rhombic cell faces are triangulated in order to allow vertices to move individually without affecting other vertex positions of the same cell while continuing to allow tissues to remain close-packed without void or overlapping spaces. This triangulation is also useful in facilitating the creation of triangle-based output file formats such as STL, which is useful in producing stereo-lithographic rapid prototype models of generated form. For this purpose, rhombic faces are simply split into two isosceles triangles. As a next step it is necessary to identify the movement ranges of all vertices — the spaces or domains within which each vertex is allowed to move while avoiding vertex eversions (which would describe unwanted *inverse spaces*). This eversion problem would for instance occur if vertices {1} and {9} on the right of figure 4 were to change their positions in such a way that {9} will be located above {1}; the cellular space between both vertices would evert and result in an undefined space. A last step in this preliminary study requires a suitable control mechanism, which will allow users to move vertices in intuitive ways using scripting functions. Before discussing these control mechanisms, we will first proceed to describe the identified vertex ranges.

To allow vertices to move freely without mutual eversion, the ranges must occupy the entire tissue space while not overlapping and hence themselves allow close packing. The left of figure 4 shows a rhombic dodecahedron with faces numbered using Miller indices (a face identification system adopted from the field of mineralogy), vertex numbers as used in *Zellkalkül* and two different vertex types labeled 'a' and 'b'. Vertices between six adjacent cells (labeled 'a') have octahedral ranges while vertices between four adjacent cells (labeled 'b') have tetrahedral ranges. The side length (or in Fuller's terminology: the geodesic vector length) of both geometries is 1 (identical to cell diameter). Figure 2 shows both types of these *platonic shapes* (left, octahedra are only shown half) and (right) their ability to close-pack into cuboctahedral assemblies. The twelve cells neighbouring the central cell are shown as wire frames in both images. By constraining vertices to remain within their ranges, this topology guarantees that self-intersecting cell surfaces are avoided.

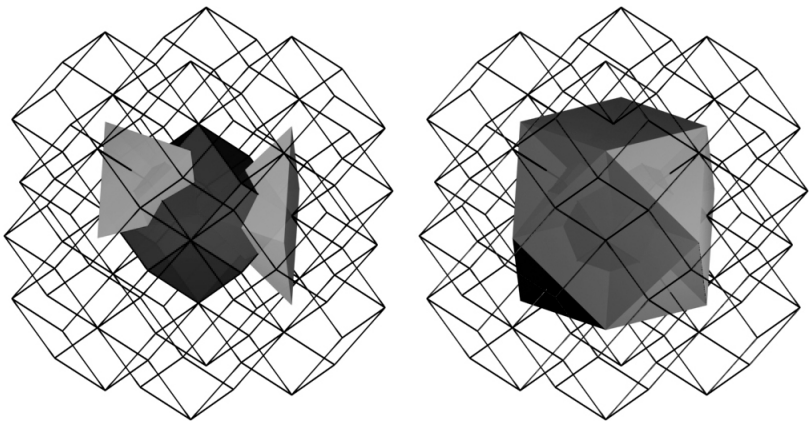


Figure 2: Cuboctahedral vertex range clusters defined by tetrahedral and octahedral (half shown) vector matrices

Though the coboctahedral geometry shown in light grey on the right of figure 2 as such does not allow space-filling close packing, this arrangement still occupies complete tissue spaces due to partial overlapping. The reason for this overlapping is that ranges are associated with multiple cells that have vertices in common.

4 User Interface

The user interface for vertex position control should be as simple, intuitive and yet as flexible as possible. For this reason, from the user's perspective, it does not distinguish between the two different vertex types 'a' and 'b' even though they are internally processed in different ways. We have decided to use a *pressure* model that allows users to control the pressure cells bear in the direction of a given vertex. Figure 3 shows the force vectors with which adjacent cells can manipulate both types of vertices. An advantage of using a pressure model for vertex position control is that positions, as in Nature, result relatively from intercellular "negotiation" rather than by unilaterally controlled, absolute positioning. A problem emerges when trying to move a vertex between four cells (shown on the left of figure 3) by means of four pressure vectors in caltrop arrangement (see inside tetrahedral range) since this does not allow the vertex to reach any point within this range. We have solved this problem by modelling forces as negative pressure (or: "tension") rather than as "pressure".

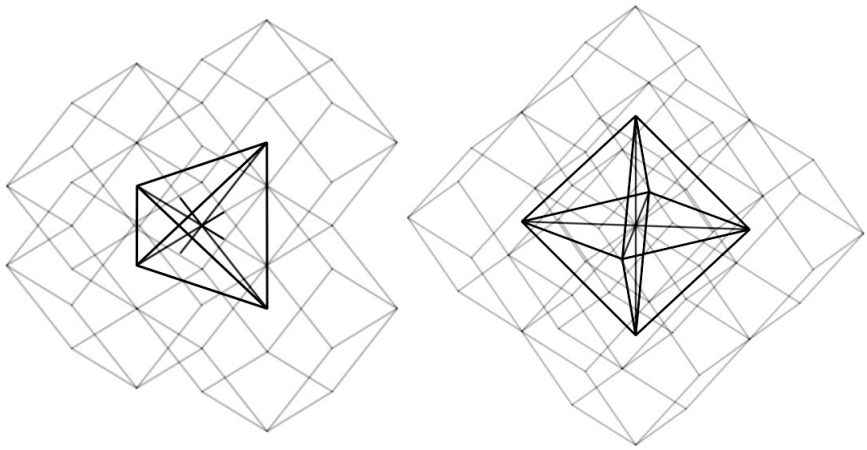


Figure 3: Force vectors within vertex ranges

The user interface is represented by the script interpreter associated with every cell. For this purpose we have extended the scripting language specification with functions that allow the identification of a vertex and a (negatively interpreted) pressure in the format `setTension(v, p)` to set a pressure and in the format `getTension(v)` to acquire a pressure. The pressure parameter is expressed as a byte value which allows relative pressure control at a resolution of 255 steps per cell involved in a vertex positioning operation. The neutral default pressure of each vertex, as well as the "atmospheric" pressure (relevant at tissue edges where vertices have no neighbours), is 127. With this default value assigned to all vertices internally and to vertices of adjacent cells, a cell assumes a normal rhombic dodecahedral shape. Other values result in "morphed" variations as shown in image 5.

5 Outlook

This section discusses further system extensions that, within this ongoing project, are of interest to us but for which we have not yet been able to identify appropriate algorithms and constraints. Firstly, the fixed ranges described in the above section appear to be rather limiting compared to the possibility of *dynamic ranges*. Secondly, *flexible cell sizes* would be of great value for generating geometrically non-uniform tissues and structures.

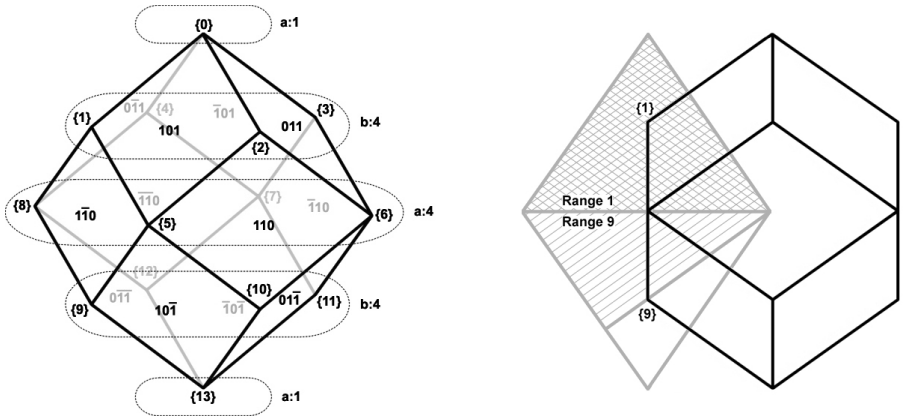


Figure 4: Miller face indices, vertex numbers and vertex layers (left), vertex ranges 1 and 9 in front view (right)

5.1 Dynamic Vertex Ranges

The ranges (vertex domains) discussed above are rigidly defined as *platonic* tetrahedra and octahedra. With these two types of ranges, vertices are not yet given the maximum possible ranges of movement as shown in the illustration on the right of figure 4. Given that vertex {9} was to remain its shown default position, the tetrahedral range of vertex {1} (double hatched) could potentially be extended by as much as half of the range of vertex {9} (single hatched). The three-dimensional interrelatedness of all vertices of a cell and its neighbours suggests a solution strategy that is based on a process of recursive approximation. This approximation should make highly economic use of physical machine resources. In tissues with large numbers of cells (tissues with up to 20,000 cells have been generated and larger ones are likely to be generated in the future) such operations are likely to jeopardize the system's present interactive responsiveness. It was noted by [1] that in the translation from idea to a product, it is in this responsiveness, where one of the key values of parametric design lies.

5.2 Flexible Cell Sizes

Especially from a structural design point of view, a function to generate geometrically non-uniform structures (cell tissues, octet trusses etc.) is enticing. This will require (again virtual pressure-based) geometric control algorithms for variable automata diameters and consequently for variable centre point locations. For such a function, however, a non-uniform 3D close packing system will not be sufficient (an early investigation into this possibility was presented by [8]). In order to maintain *Zellkalkül*'s intercellular communication infrastructure, such a system needs to be constrained in a way that always preserves the number of twelve cell neighbours. An alternative solution could be based on intercellular communication facilities that are designed for variable numbers of neighbours. Since this would however result not only in structurally chaotic forms but also in a highly untidy organisation of the user scripting interface, a solution based on geometric constraints would be preferable.

6 Acknowledgements

We gratefully acknowledge the support and advice from our colleagues and teachers at the School of Design at the Hong Kong Polytechnic University, at the Spatial Information Architecture Laboratory at the Royal Melbourne Institute of Technology and at the Faculty of Mathematics at the University of Göttingen, in particular Prof. John Frazer, Prof. Mark Burry and Timothy Jachna.

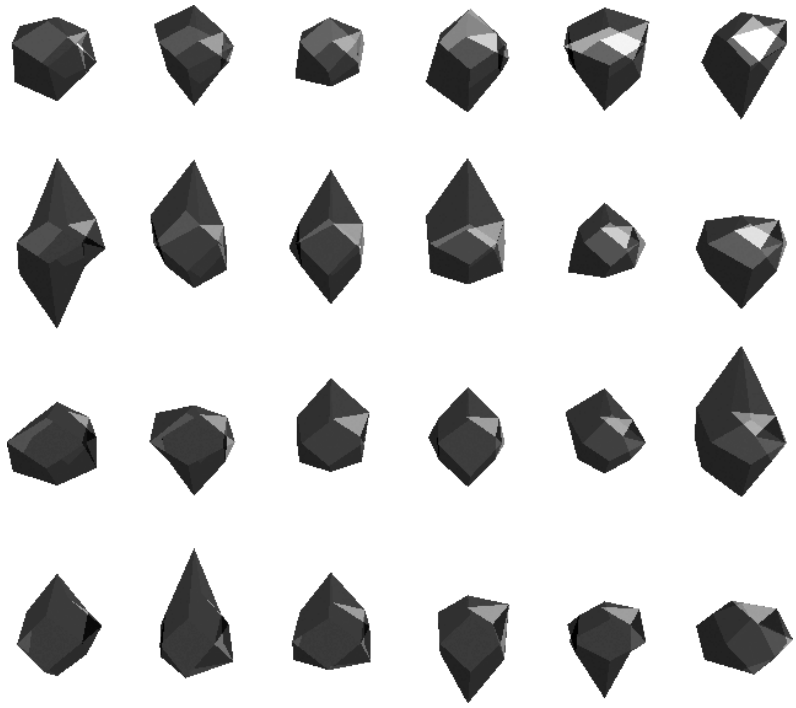


Figure 5: Array of parametrically morphed cells

References

[1] Mark C. Burry and Zolna Murray. Architectural design based on parametric variation and associative geometry. In *Challenges of the Future. Proceedings of the 15th eCAADe Conference*, pages 1–11, Österreich Kunst und Kulturverlag, Vienna, Austria, 1997.

[2] Thomas Fischer. Computation-universal voxel automata as material for generative design education. In Celestino Soddu et al., editor, *The Proceedings of the 5th Conference and Exhibition on Generative Art 2002*, pages 10.1–1.11, Generative Design Lab, DiAP, Politechnico di Milano University, Milan, Italy, 2002.

[3] Thomas Fischer, Torben Fischer, and Cristiano Ceccato. Distributed agents for morphologic and behavioral expression in cellular design systems. In George Proctor, editor, *Thresholds. Proceedings of the 2002 Conference of the Association for Computer Aided Design in Architecture*, pages 113–123, Department of Architecture, College of Environmental Design, California State Polytechnic University, Pomona, Los Angeles, 2002.

[4] John H. Frazer. *An Evolutionary Architecture*. Architectural Association, London, 1995.

[5] R. Buckminster Fuller. *Synergetics. Explorations in the Geometry of Thinking*. Macmillan Publishing, New York, 1975.

[6] D. G. Kieser. *Phytotomie, oder Grundzüge der Anatomie der Pflanzen*. Cröcker, Jena, 1815.

[7] Branco Kolarevic. Digital morphogenesis and computational architectures. In Jose Ripper Kos, editor, *The Proceedings of SIGraDi2000 - Construindo (n)o espacio digital (constructing the digital Space)*, pages 98–103, Facultad de Arquitectura - Universidad Nacional de Mar del Plata, Rio de Janeiro, Brazil, 2000.

[8] Gary Kong. Cellular automatas and "stacking balls". Technical dissertation forming part of the Diploma of the Architectural Association, London, 1994.

Controlled Perturbation for Arrangements of Circles*

Extended Abstract

Dan Halperin Eran Leiserowitz

School of Computer Science
Tel Aviv University
{danha,leiserow}@tau.ac.il

Abstract

Given a collection $\mathcal{C}$ of circles in the plane, we wish to construct the arrangement $\mathcal{A}(\mathcal{C})$ (namely the subdivision of the plane into vertices, edges and faces induced by $\mathcal{C}$) using floating point arithmetic. We present an efficient scheme, controlled perturbation, that perturbs the circles in $\mathcal{C}$ slightly to form a collection $\mathcal{C}'$, so that all the predicates that arise in the construction of $\mathcal{A}(\mathcal{C}')$ are computed accurately and $\mathcal{A}(\mathcal{C}')$ is degeneracy free.

We introduced controlled perturbation several years ago, and already applied it to certain types of arrangements. The major contribution of the current work is the derivation of a good (small) resolution bound, that is, a bound on the minimum separation of features of the arrangement that is required to guarantee that the predicates involved in the construction can be safely computed with the given (limited) precision arithmetic. A smaller resolution bound leads to smaller perturbation of the original input.

We implemented the perturbation scheme and the construction of the arrangement and we report on experimental results.

1 Introduction

Computational geometry algorithms often assume general position of the input and the "real RAM" computation model. In the case of an arrangement of circles, general position of the input means that there is no outer or inner tangency between two circles, and that no three circles intersect at a common point (see Figure 1 for a degenerate arrangement). If one wishes to use floating-point arithmetic (to achieve fast running time), then even if the input *is* in general position, round-off errors may cause the algorithm to fail.

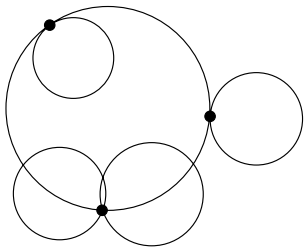


Figure 1: Arrangement of circles with several degeneracies.

Thus, while building the arrangement in an incremental fashion (that is, adding one circle at a time), we will check if there is a potential degeneracy induced by the newly added circle, and if so, we will move that circle, so no degeneracies will occur. The main idea is to carefully relocate the circle — move the circle enough to avoid the degeneracies, but not too much. Depending on the precision of the machine floating-point representation, and some properties of the arrangement to be handled, we determine a bound δ on the magnitude of the perturbation, namely, we guarantee that any input circle will not be moved by a distance greater than δ.

Such a perturbation scheme, as was described above, could be useful for the following reasons: (i) floating-point arithmetic is usually supported by hardware, making computations very fast, (ii) degeneracies are eliminated, thus an algorithm is made easier to analyze and implement, (iii) implementations using exact arithmetic with floating-point filtering, can be sped up, since the perturbation will cause the predicates to be evaluated using the floating-point filters, thus avoiding the use of exact computation.

In many situations, the original input data is inac-

*Work reported in this paper has been supported in part by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

curate to begin with (due to, for example, measuring errors or approximate modeling), so the damage incurred by perturbing slightly is negligible.

The predicates that arise in the construction of arrangements of circles include expressions that contain division and square-root operations. Those operation are usually more difficult to handle robustly than addition, subtraction and multiplication.

The perturbation scheme that we follow, controlled perturbation, was first presented in [6] as a method to speed up molecular surface computation. The use of exact computation turned out to be too slow for real time manipulation, so a finite precision method was needed. Controlled perturbation was devised to handle the robustness issues caused by the use of finite precision arithmetic, and to remove all the degeneracies. It was extended in [9], where it was applied to arrangements of polyhedral surfaces. Those arrangements require complex calculations in order to achieve a good perturbation bound.

In [9] (as in [6]), the *resolution bound* (defined in the next section) is assumed to be given. *The resolution bound is a key element in the scheme.* In this work we describe a method for obtaining good resolution bounds, which we anticipate will lead to a better understanding of the method and will open the way to applying the method in other settings.

Related work

Robustness and precision issues have been intensively studied in Computational Geometry in recent years [10].

A prevailing approach to overcoming robustness issues in computational geometry is to use exact computation [7, 13]. Such a strategy gives accurate results, and sometimes even allows the input to be degenerate. When applied naively, exact computation can considerably slow down the performance of a program. One of the possible solutions is to use *filtering* [2, 4, 11]. Typically, the filtering is done at the level of the number type. That is, a predicate is evaluated using exact computation *only* if it cannot be correctly evaluated using finite precision arithmetic. In [12], high-level filtering is done on arrangements of conic arcs; a different approach for computing arrangements of conic arcs is given in [1].

An alternative approach aims to compute robustly with limited precision arithmetic, often by approximating or perturbing the geometric objects [3, 5, 8]. A variety of methods for handling imprecise geometric computations are surveyed in [10]. Controlled perturbation is a method of this type.

2 Overview of the Perturbation Scheme

For an input circle C_i, our algorithm will output a copy C_i' with the same radius but with its center possibly perturbed. We denote by $\mathcal{C}_j$ the collection of circles $\{C_1, \ldots, C_j\}$, and by $\mathcal{C}_j'$ the collection of circles $\{C_1', \ldots, C_j'\}$.

The input to our algorithm is the collection $\mathcal{C} = \mathcal{C}_n$ of n circles. Each circle C_i is given by the coordinates of its center X_i, Y_i and its radius R_i (we assume that all the input parameters are representable as floating-point numbers with the given precision). The input consists of two additional parameters: (i) the machine precision p, namely the length of the mantissa in the floating-point representation, and (ii) an upper bound on the absolute value of each input number X_i, Y_i and R_i. The perturbation scheme transforms the set $\mathcal{C}$ into the set $\mathcal{C}' = \mathcal{C}_n'$.

We will build the arrangement in an incremental fashion (that is, adding one circle at a time), and if there is a potential degeneracy while adding the current circle, we will perturb it, so no degeneracies will occur. We next describe the two key parameters that govern the perturbation scheme, the resolution bound and the perturbation bound.

Resolution bound

A degeneracy occurs when a predicate evaluates to zero. The goal of the perturbation is to cause all the values of all the predicate expressions (that arise during the construction of the arrangement of the circles) to become significantly non-zero, namely to be sufficiently far away from zero so that our limited precision arithmetic could enable us to safely determine whether they are positive or negative.

The degeneracies that arise in arrangement of circles have a natural geometric characterization as *incidences*. For example, in *outer tangency*, two circles intersect in a single point. In our scheme we transform the requirement that the predicates will evaluate to sufficiently-far-from-zero values into a geometric distance requirement.

This is a crucial aspect of the scheme: the transformation of the non-degeneracy requirement into a separation distance. We will call the bound on the minimum required separation distance, the *resolution bound* and denote it by ε. Deriving a good resolution bound is a central innovation in this work. Previously (e.g., [6]) we assumed that these bounds were given, and in our experiments we used crude (high) bounds. The bound on ε depends on the size of the input numbers (center coordinates and radii) and the machine precision. It is independent of the number

n of input circles.

Perturbation bound

Suppose indeed that ε is the resolution bound for all the possible degeneracies in the case of an arrangement of circles for a given machine precision. When we consider the current circle C_i to be added, it could induce many degeneracies with the circles in $\mathcal{C}'_{i-1}$. Just moving it by ε away from one degeneracy may cause it to come closer to other degeneracies. This is why we use a second bound δ, the *perturbation bound*. The bound δ depends on ε, on the maximum radius of a circle in $\mathcal{C}$, and on a *density* parameter k of the input which bounds the number of circles that are in the neighborhood of any given circle and may effect it during the process, $k \leq n$ (a formal definition of k is given in the full version of the paper; in the worst case $k = n$).

We say that a point q is a valid placement for the center of the currently handled circle C_i, if when moved to q this circle will not induce any degeneracy with any of the circles in $\mathcal{C}'_{i-1}$. The bound δ is computed such that inside the disc D of radius δ centered at the original center of C_i, at least half the points (constituting half of the area of the disc) will be valid placements for the circle. This means that if we choose a point uniformly at random inside D to relocate the center of the current circle, it will be a valid placement with probability at least $\frac{1}{2}$.

After the perturbation, the arrangement $\mathcal{A}(\mathcal{C}')$ is degeneracy free. Moreover, $\mathcal{A}(\mathcal{C}')$ can be robustly constructed with the given machine precision.[1]

An alternative view of our perturbation scheme is as follows. We look to move the centers of the input circles slightly from their original placement such that when constructing the arrangement $\mathcal{A}(\mathcal{C}')$ while using a fixed precision (floating-point) filter, the filter will always succeed and we will never need to resort to higher precision or exact computation.

The details of how to compute the resolution bound and the perturbation bound are given in the full version of this paper.

We quote the result summarizing the resources required by the algorithm.

Theorem 1 *Given a collection $\mathcal{C}$ of n circles, the perturbation algorithm which allows the construction of the arrangement $\mathcal{A}(\mathcal{C}')$ runs in total expected $O(n^2 \log n)$ time.*

[1] The perturbation algorithm should not be confused with the actual construction of the arrangement. It is only a preprocessing stage. However, it is convenient to combine the perturbation with an incremental construction of the arrangement.

3 Defining the Predicates and Determining a Worst Case ε

As was already stated, the main contribution of this work is in computing the resolution bound. To do so, we examine the possible degeneracies, and find the ε required to remove them once we are given the precision of the underlying arithmetic. In other words, we determine for each degeneracy a distance ε such that if a pair of features related to this degeneracy are at least ε apart, then we can safely evaluate the corresponding predicate with the given precision. For each degeneracy we present the appropriate predicate and also compute the worst case ε. Using this ε we then compute the value of δ, the maximum distance of a perturbed circle C_i from its original position, as described in the previous section.

Denote a predicate which takes m arguments and determines the sign of an expression by $Pr_s = sign(E(x_1, \ldots, x_m))$. Denote by Pr_p the predicate which takes m arguments and returns *true* iff $E(x_1, \ldots, x_m) > 0$. We define a degeneracy when $E = 0$.

Since we are using floating-point arithmetic, we cannot compute E exactly. Instead, we are only computing an approximation $\tilde{E}$ of E. We also compute a bound $B > 0$ on the maximum difference between $\tilde{E}$ and the exact value E, namely, $|E - \tilde{E}| \leq B$ or $\tilde{E} - B \leq E \leq \tilde{E} + B$. Thus, if $\tilde{E} > B$ then $E > 0$, and if $\tilde{E} < -B$ then $E < 0$. The bound B is computed according to the method given in [2].

When we add C_i to the collection $\mathcal{C}'_{i-1}$, if *for all* the predicates involving C_i (regarding all the circles that were already inserted), $|\tilde{E}| > B$, then C_i is in a valid place, and there is no need to perturb it. If there *exists* a predicate P, for which $|\tilde{E}| \leq B$, we define such a configuration as a *potential degeneracy*, and we need to perturb C_i. *For each predicate, we need to understand the geometric meaning, of $|\tilde{E}| > B$, so it will be reflected in ε and then in δ.* The details are given in the full version of the paper.

4 Experimental Results

In this section we report on experimental results with our implementation of the perturbation scheme that was described above. We implemented the perturbation scheme as a set of C++ classes. We also implemented the DCEL (Doubly Connected Edge List) construction with a simple point-location mechanism.

We have tested our program on four input sets (see Figure 2): `grid`, `flower`, `rand_sparse`, and `rand_dense`. For `rand_sparse` and `rand_dense`, all the input parameters are given as integers (to "promote" degeneracies). The properties of each input set

are given in Table 1. The results of the perturbation and running times for those inputs are give in Table 2 (with the IEEE double number type). The tests have been performed on an Intel Pentium III 1 GHz machine with 2 GB RAM, operating on a Redhat 7.1 using gcc 3.03.

Name	n	R	M
`grid`	320	10	140
`flower`	40	100	100
`rand_sparse`	40	20	100
`rand_dense`	100	49	100

Table 1: n denotes the number of circles, R denotes the maximum radius and M is the maximum input size (center coordinates).

name	avg.	max.	p_time	t_time
`grid`	0.1319	0.7275	0.386	0.404
`flower`	0.9783	3.5470	0.274	0.28
`rand_sparse`	0.0158	0.0172	0.004	0.006
`rand_dense`	0.0382	0.3860	0.22	0.23

Table 2: Avg. denotes the average perturbation size, max. denotes the maximum perturbation size, p_time denotes the time of the perturbation (in seconds) and t_time denotes the total (perturbation and DCEL construction) time (in seconds). The given results are from averaging the results of 5 tests for each input.

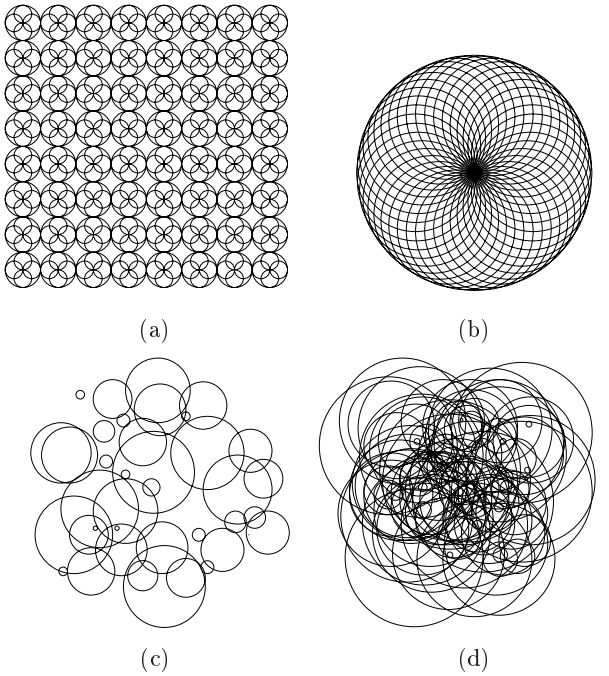


Figure 2: (a) A grid of 320 circles, which involves many inner and outer tangencies. (b) A "flower" composed of 40 circles, all intersecting in a common point. (c) A collection of 40 random circles. (d) A collection of 100 random circles.

References

[1] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and Boolean operations on conic polygons. In *Proc. ESA 2002*, pages 174–186. Springer-Verlag, 2002.

[2] C. Burnikel, S. Funke, and M. Seel. Exact geometric computation using cascading. *International Journal of Computational Geometry and Applications (IJCGA)*, 11(3):245–266, 2001.

[3] S. Fortune and V. Milenkovic. Numerical stability of algorithms for line arrangements. In *Proc. 7th ACM Sympos. Comput. Geom.*, pages 334–341, June 1991.

[4] S. Fortune and C. J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15(3):223–248, July 1996.

[5] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th ACM Sympos. Comput. Geom.*, pages 208–217, 1989.

[6] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.

[7] K. Melhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[8] V. J. Milenkovic. Verifiable implementations of geometric algorithms using finite, precision arithmetic. *Artif. Intell.*, 37:377–401, 1988.

[9] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Symposium on Computational Geometry*, pages 163–172, 1999.

[10] S. Schirra. Robustness and precision issues in geometric computation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[11] J. Shewchuk. Adaptive robust floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18:305–363, 1997.

[12] R. Wein. High level filtering for arrangements of conic arcs. In *Proc. ESA 2002*, pages 884–895. Springer-Verlag, 2002.

[13] C. K. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.

Vertex Cover and Connected Guard Set

Paweł Żyliński

Institute of Mathematics
University of Gdańsk
80-392 Gdańsk, Poland
e-mail: pz@math.univ.gda.pl

The *art gallery problem* asks how many guards are sufficient to see every point of the interior of an n-vertex simple polygon. The guard is a stationary point who can *see* any point that can be connected to it with a line segment within the polygon. A collection of guards $S = \{g_1, ..., g_k\}$ is said to *cover* the polygon P if every point $x \in P$ can be seen by some guard $g \in S$.

For a guard set S we define the *visibility graph* $VG(S)$ as follows: the vertex set is S and two vertices v_1, v_2 are adjacent if the line segment with endpoints v_1 and v_2 is a subset of P, i.e. $\overline{v_1 v_2} \subseteq P$. Next, the guard set S is said to be *cooperative* (*connected*) if the graph $VG(S)$ is connected.

The concept of *cooperative guards* was proposed by Liaw, Huang and Lee [5]. They established that *The Minimum Cooperative Guards Problem* for simple polygons is NP-hard, but for spiral and 2-spiral polygons this problem can be solved in linear time [5]. For k-spiral polygons the minimum number of cooperative guards is at most N_k, the total number of reflex vertices in the k-spiral polygon [3]. The cooperative guards problem for general simple polygons has been completely settled by Hernández-Peñalver, proving that $\lfloor \frac{n}{2} \rfloor - 1$ cooperative guards are always sufficient and occasionally necessary to guard a polygon of n vertices [4].

The *diagonal graph* G_D of any triangulation of an n-vertex polygon P is a graph obtained only from $n-3$ internal diagonals of the triangulation: the edges correspond to the diagonals and the vertices correspond to all endpoints of diagonals. Herein we discuss the relation between a vertex cover of a diagonal graph and a connected vertex guard set in a polygon (guards are restricted to be located only at the vertices of the polygon): we show that any set S is a vertex cover of G_D if and only if S forms a connected vertex guard set in P.

1 Diagonal graphs

Lemma 1.1 *Let P, T and G_D be a simple n-vertex polygon ($n \geq 4$), its arbitrary triangulation, and the diagonal graph of triangulation T, respectively. Then G_D is connected.* □

Let us recall that a graph is *outerplanar* if it can be embedded in the plane so that all of its vertices lie on the exterior face.

Lemma 1.2 *Let m be the number of edges of a connected outerplanar graph G. Then there exists a vertex cover of cardinality at most $\lfloor \frac{m+1}{2} \rfloor$.* □

Let P be a polygon of n vertices. Any of its diagonal graphs has $n-3$ edges, and, of course, it is outerplanar. By Lemma 1.1 and Lemma 1.2 we get the following:

Corollary 1.3 *Let G_D be the diagonal graph of a triangulation of an n-vertex polygon. Then there exists a vertex cover of cardinality at most $\lfloor \frac{n-2}{2} \rfloor$.* □

2 Vertex cover vs. connected guard set

A *triangulation graph* G_T of an n-vertex simple polygon P is a graph obtained by triangulation P with internal diagonals between vertices: the vertices of G correspond to the n vertices of P, and the edges correspond to the n edges of P and $n-3$ diagonals.

A *vertex guard* in G_T is a single vertex of G_T. A set of guards $S = \{g_1, \ldots, g_k\}$ is said to *dominate* G_T if every triangular face of G_T has at least one of its vertices assigned as a guard ($\in S$). Finally, the collection of guards $S = \{g_1, \ldots, g_k\}$ is said to be *connected* if for any two guards $g_i, g_j \in S$ there exists a path $p = (g_i, p_1, \ldots, p_l, g_j)$ in triangulation graph G_T that all $p_t \in S$, for $t = 1, \ldots, l$. Guards in graph G_T are called *combinatorial connected guards* to distinguish them from the *geometric connected guards* introduced earlier. The reason for introducing triangulation graphs is that a proof of sufficiency of a certain number of combinatorial connected guards establishes the sufficiency of the same number of geometric connected guards in a polygon.

Lemma 2.1 [4] *Let P be a simple polygon, and G_T be one of its triangulation graphs. If G_T can be dominated by k combinatorial connected guards, then P can be covered by k geometric connected vertex guards.* □

The main use of diagonal graphs is the following result.

Theorem 2.2 *Let T, G_T, G_D be any triangulation of a simple polygon, a triangulation graph of T and the diagonal graph of T, respectively. If $C = \{g_1, \ldots, g_k\}$ is a vertex cover of graph G_D, then C is a connected guard set in G_T.* □

We note in passing that Theorem 2.2 holds also for *iff*:

Theorem 2.3 *Let T, G_T, G_D be any triangulation of a simple polygon, a triangulation graph of T and the diagonal graph of T, respectively. A connected guard set S in G_T is a vertex cover of diagonal graph G_D.* □

Corollary 1.3 and Theorem 2.2 lead immediately to the following:

Corollary 2.4 *$\lfloor \frac{n-2}{2} \rfloor$ connected guards are sometimes necessary and always sufficient to cover any polygon of n vertices.* □

3 Final remarks

The idea of the proof of the sufficiency of $\lfloor \frac{n-2}{2} \rfloor$-bound leads immediately to a linear approximation algorithm AD for finding any connected guard set for a polygon P (guards will be located at vertices):

(1) triangulate P; (in $O(n)$ steps [2])

(2) find any minimum vertex cover of the diagonal graph G_D. (in $O(n)$ steps [8])

Nevertheless, this algorithm can be arbitrarily bad.

Let $S_{AD}(P)$ and $S_{OPT}(P)$ denote the number of connected guards obtained by algorithm AD, and the minimal number of connected guards that cover P, respectively. It is natural to ask about:

$$\lim_{n\to\infty} \max_{G_D} \frac{S_{AD}(G_D)}{S_{OPT}(P)},$$

that is how the obtained result can differ from the optimal solution.

Consider a polygon P of $4k+2$ vertices, its triangulation T, and its corresponding diagonal graph G_D shown in Fig. 1. It is clear, that any minimal vertex cover of G_D is of cardinality $k+1$,

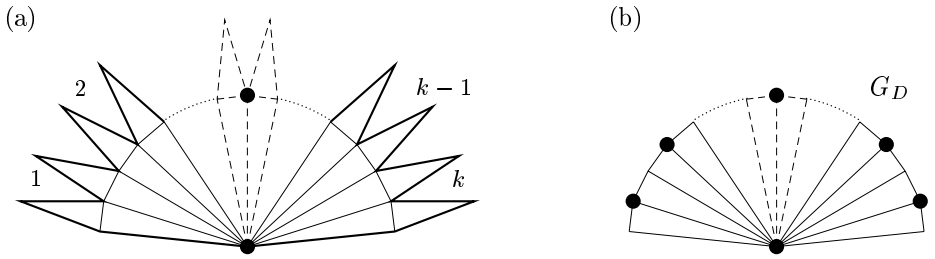


Fig. 1. A star polygon of $4k+2$ vertices that requires only 2 connected vertex guards, (a) its triangulation T, (b) the minimum vertex cover of G_D is of cardinality $k+1$.

and as P is a star polygon with one of its vertices in the kernel, it can be guarded only by two connected vertex guards. Thus:

$$\lim_{n\to\infty} \max_{G_D} \frac{S_{AD}(G_D)}{S_{OPT}(P)} = \infty.$$

We recall that The Minimum Connected Guard Problem for simple polygons was shown to be NP-hard [5].

References

[1] V. Chvátal. A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B* **18**, 39-41 (1975).

[2] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.* **6(5)**, 485-524 (1991).

[3] J.S. Deogun, S.T. Sarasamma. On the minimum cooperative guard problem. *J. Combin. Math. Combin. Comput.* **22**, 161-182 (1996).

[4] G. Hernández-Peñalver. Controlling guards. *Proc. of Sixth Canadian Conference on Computational Geometry*, 387-392 (1994).

[5] B-C. Liaw, N.F. Huang, R.C.T. Lee. The minimum cooperative guards problem on k-spiral polygons. *Proc. of the Fifth Canadian Conference on Computational Geometry*, 97-101 (1993).

[6] J. O'Rourke. Galleries need fewer mobile guards: a variation to Chvatátal's Theorem. *Geometriae Dedicata* **14**, 273-283 (1983).

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

[8] I.G. Tollis. On finding a minimum vertex cover of a series-parallel graph. *Appl. Math. Lett.* **2**, *no.* 3, 305-309 (1989).

[9] J. Urrutia. *Art Gallery and Illumination Problems.* Handbook on Computational Geometry, Elsevier Science, Amsterdam (2000).

An optimal competitive on-line algorithm for the minimal clique cover problem in interval and circular-arc graphs

Jerzy W. Jaromczyk
Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA
jurek@cs.uky.edu

Andrzej Pezarski
Institute of Computer Science
Jagiellonian University
Krakow, Poland
pezarski@ii.uj.edu.pl

Maciej Ślusarek
Institute of Computer Science
Jagiellonian University
Krakow, Poland
slusarek@ii.uj.edu.pl

ABSTRACT

We define and study an on-line version of the *shooter problem O-LSP*. In the standard off-line version the problem is: given a finite set S of line segments in R^2 and a point p find the smallest number of half-lines that start at p and intersect all the segments in S. In our *O-LSP* on-line version the segments $S = \{p_1, ..., p_n\}$ are given one by one and the selection of the suitable intersecting half-line for p_i must be done, and cannot be changed, immediately after seeing p_i. In this process we can use a half-line that already intersects some of the segments S_0 in $\{p_1, ..., p_{i-1}\}$ by possibly rotating this half-line in such a way that it keeps intersecting S_0. Via the well-know relation between the shooter problem and circular-arc graphs we reduce it to the on-line Minimal Clique Cover problem (*MCC*) for the classes of interval and circular-arc graphs. Specifically, we are interested in the competitiveness of on-line algorithms for *MCC*. An on-line algorithm A is c_F-competitive for the family F of graphs if for all $G \in F$, $A(G) \leq c_F OPT(G) + b_F$, c_F and b_F constants, where $A(G)$ is the solution found by algorithm A for graph G and $OPT(G)$ is the optimal (off-line) solution. We analyze on-line algorithms and two simple (and seemingly similar) greedy strategies (called LGR and EGR) for *MCC* (and *O-LSP*) and show both upper and lower bounds on their competitiveness ratio c_F. We demonstrate that:

- $c_F \geq 2$, for any on-line algorithm;
- c_F is unbounded for LGR; hence LGR is not competitive.
- $c_F = 2$ for EGR; hence EGR is optimal.

Keywords

Shooting Problem, On-line algorithms, Interval Graphs, Circular-Arc Graphs

1. DEFINITIONS AND BACKGROUND

An interval graph G is the intersection graph of a family of closed intervals in the real line. Similarly, a circular-arc graph is the intersection graph of a family of closed arcs in a circle. Each vertex v of G corresponds to an interval (an arc, respectively) and the collection of intervals (arcs, respectively) is the representation of G. Interval graphs are often defined in terms of posets as each of them is the co-comparability graph of a set of closed intervals; see [F85]. Note that interval graphs form a proper subclass of circular-arc graphs. The class of interval and circular-arc graphs have been intensively studied, in particular because of their practical applications (e.g., in memory allocation and in organizing records in databases [BL76]).

The clique cover for graph G is defined as the family of subgraphs of G such that each subgraph is a clique and their union is G. Note that subgraphs in the clique cover do not need to be vertex-disjoint. The clique cover of the smallest cardinality $\theta(G)$ is called a minimal clique cover and the algorithmic problem of finding it will be denoted in this paper by *MCC*.

Specifically, we study on-line algorithms for *MCC* and our motivations stem from the *shooting problem* studied in computational geometry; see [ChN99,JK02]. The problem can be stated as follows: given a finite set S of line segments in R^2 and a point p find the smallest cardinality set of half-lines that start at p and intersect (stab) all the segments in S. After projecting the segments onto a disc centered at p, the problem corresponds to stabbing arcs; see Figure 1. The on-line version of *MCC* means that the segments are given one-by-one is some order and the shooter must decide immediately after seeing this segment which of the current shooting directions or a new one will be used; after the decision has been made the shots cannot be reassigned. Clearly, segments intersected by the same shot form a clique in the corresponding circular-arc graph and the smallest number of such cliques determines the smallest number of shots. An important theorem, due to Hsu (Theorem 3.2, [HT91]), relates the size of the *MCC* in circular-arc graphs G with the maximal independent set size $\alpha(G)$.

THEOREM 1.1. *[HT91] If G is not a clique then $\theta(G) = \alpha(G)$ or $\theta(G) = \alpha(G) + 1$.*

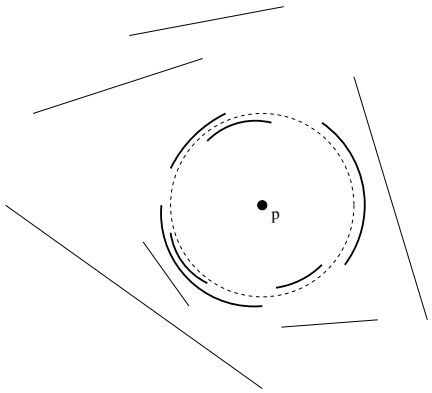


Figure 1: An instance of the shooting problem.

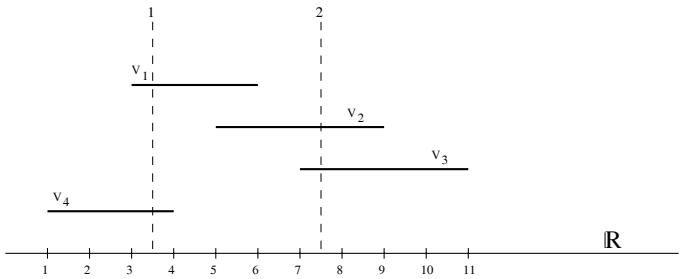


Figure 2: Shots and their ranges: $A(v_1) = A(v_4) = 1$, $r_4[1] = [3,4]$, $A(v_2) = A(v_3) = 2$, $r_4[2] = r_3[2] = [7,9]$.

MCC problem for graphs G is clearly related to the graph-coloring problem where the objective is to find the minimum integer k, called the chromatic number $\chi(G)$, and a function $f : V(G) \to \{1, 2, \ldots k\}$ such that no edge $e = (u, v)$ has $f(u) = f(v)$. For interval graphs G, we have $\theta(G) = \chi(\overline{G})$, where $(\overline{G})$ is the graph complement of G. There is a number of strong results related to on-line coloring algorithms for interval and circular-arc graphs; see e.g. [K98,KQ95,S89,S95]. Since the complements of interval graphs are not, in general, interval graphs, the above relation does not help to solve our *MCC* problem via coloring. Similarly, results for graph-coloring do not help with the circular-arc graphs.

2. ON-LINE *MCC* PROBLEM

An on-line presentation $G^{\ll}$ of a graph G is a linear order $\ll$ of vertices V of $G(V, E)$. $G_i^{\ll}$ is the on-line graph induced by the first i elements $\{v_1, \ldots, v_i\}$ of V in $\ll$ order. The on-line minimum clique cover (on-line *MCC* in short) is specified as follows:

An algorithm A is an on-line algorithm for the minimum clique cover of an on-line graph G, if given a presentation $G^{\ll}$ with the order $V = \{v_1, \ldots, v_n\}$ it computes a sequence of positive integers $A(v_i), i = 1, \ldots, n$, where $A(v_i)$ is the name of a clique that covers v_i, in such a way, that for each i, $A(v_i)$ depends exclusively on $G_i^{\ll}$.

In other words, the vertices are input one by one, and the number of the clique that covers v_i is established irrevocably after reading $v_1, \ldots, v_i$, together with their adjacency structure.

The quality of on-line algorithms is measured by the *competitive ratio*. An on-line algorithm A is c_F-competitive for the family F of graphs if for all $G \in F$, $A(G) \leq c_F OPT(G) + b_F$, c_F and b_F constants, where $A(G)$ is the solution found by algorithm A for graph G and $OPT(G)$ is the optimal (off-line) solution. In case of the Minimum Clique Cover, we have $OPT(G) = \theta(G)$.

We are looking for competitive algorithms for the on-line *MCC* problem on interval and circular arc-graphs. For the sake of simplicity we describe all results for the class of interval graphs, mentioning the differences for circular-arc graphs whenever necessary.

The main assumption for our considerations is that the input graph is given already in its interval (respectively: circular-arc) representation, i.e., $v_i = [p_i, q_i]$ – a closed interval on the real line or a closed arc on a circle. This is exactly the input to *O-LSP*. The letter s, possibly with subscripts, will be used to denote shots (i.e., the ordinal numbers for the cliques). If A is an on-line clique cover algorithm then by $A(v)$ we denote the shot that covers v, the one that is assigned immediately after reading v. Observe that in general when the algorithm ends there can be also other shots that stab v.

For an interval graph a shot (i.e., a clique) can be depicted as a vertical line and a set of segments stabbed with this line. Usually the choice of such a line for a fixed clique in an interval graph is not unique. To make our reasoning precise we introduce the notion of the shot's range; see Figure 2.

DEFINITION 2.1. *Assume that s is a shot number that is already in use after processing $G_m^{\ll}$. The range $r_m[s]$ of s at phase m is defined as follows: $r_m[s] = \bigcap\{v_j \mid 1 \leq j \leq m, A(v_j) = s\}$.*

Observe that for each shot s in use we have $r_m[s] \neq \emptyset$. The ranges of shots potentially decrease in course of the computation, $r_{m+1}[s] \subseteq r_m[s], m = 1, \ldots, n-1$.

3. GREEDY ALGORITHMS

DEFINITION 3.1. *Algorithm A for the on-line* MCC *problem is greedy if for each vertex v_i in $G^{\ll}$, upon assigning $A(v_i)$, if there is already a shot range that intersects v_i then $A(v_i)$ is not set to a new shot number.*

In other words, a greedy strategy always tries to assign shots already in use, if at all possible. Note that this rule alone may lead to ambiguous decisions. In the following example: $v_1 = [1,3]$, $v_2 = [4,6]$, $v_3 = [2,5]$ a greedy algorithm yields $A(v_1) = 1, A(v_2) = 2$, and $A(v_3)$ can be either 1 or 2. Despite this ambiguity we can formulate an easy yet very important property that turns out to be useful for the subsequent considerations.

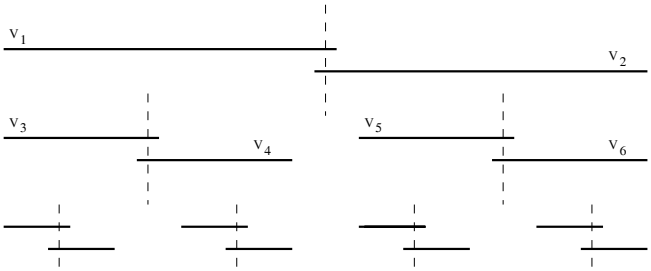


Figure 3: Tree of intervals

LEMMA 3.1. *If $s_1 \neq s_2$ are two shots already in use by a greedy algorithm A at the moment m then $r_m[s_1] \cap r_m[s_2] = \emptyset$.*

PROOF: The lemma follows easily from the monotonicity of shots' ranges. □

We are able to formulate and prove the first of our main results.

THEOREM 3.1. *For any $\epsilon > 0$ there does not exist a $(2-\epsilon)$-competitive algorithm (of any kind) that solves the on-line* MCC *problem.*

PROOF: We are going to build a strategy for an adversary who plays against a *MCC* on-line algorithm A. In order to clear away the influence of the additive constant b_F that stands in the definition of the competitiveness we have to prove that for any $\epsilon > 0$ and $m > 0$ the adversary can impose $A(G)/\theta(G) > 2 - \epsilon$ for some interval graph G such that $\theta(G) \geq m$.

For any natural number k we construct an appropriate sequence of $n = 2^{k+1} - 2$ intervals. The construction for $k = 3$ is depicted in Figure 3. The adversary uses this sequence, perhaps several times, according to the following rules:

1. If all A's clique assignments adhere to the greedy principle (as shown in Figure 3) this part of the game ends after exhausting the whole sequence. We obtain $A(G) = n/2 = 2^k - 1$ and $\theta(G) = \alpha(G) = 2^{k-1}$. Hence $A(G)/\theta(G)$ can be made arbitrarily close to 2 by a suitable choice of k.
2. If at some moment algorithm A defines two new cliques for two consecutive intervals v_{2i-1}, v_{2i}, $i = 1, 2, \ldots$ (that is, A does not adhere to the greedy principle) complete the current level of the tree and finish this part of the game. Let $j - 2$ be the index of the last vertex on this level, i.e. j be the closest power of two greater than $2i$. Then the results are: $\theta(G) = j/4$, and, summing up the last level separately with the remaining ones, $A(G) \geq (j/4 + 1) + (j/4 - 1) = j/2$, which makes the ratio $A(G)/\theta(G)$ greater or equal 2.

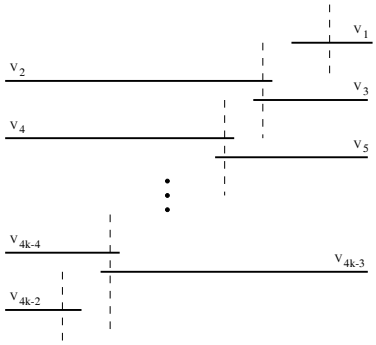


Figure 4: Intervals for LGR.

The whole game is constructed by repeating the above strategy in separate areas (intervals) of the real line sufficiently many times in order to obtain the required size of the graph $\theta(G) \geq m$. □

The ambiguity of the clique assignment by the greedy property can be resolved in various ways. We analyze two most intuitive strategies, *leftmost greedy LGR* and *earliest greedy EGR*.

The LGR strategy has a clear geometrical flavor, and it works according to the following greedy principle: From the available shots ranges that intersect the current interval choose the leftmost one. Since the ranges of any pair of shots are disjoint (see Lemma 3.1) this assignment is well defined.

The EGR works similarly: it selects the earliest available shot instead of the leftmost one. In this regard it resembles classical First-Fit packing algorithms.

Despite some similarity of the two variants of the greedy approach there exists a broad gap between their efficiency. The former turns out to be non-competitive while the latter is optimal.

THEOREM 3.2. *LGR is not competitive.*

PROOF: For any $k > 0$ we construct a sequence of $4k - 2$ intervals, as depicted in Figure 4. It is easy to see that for the graph G generated by these intervals we have $LGR(G) = 2k$ and $\theta(G) = 2$ – just a shot along left ends of odd-numbered intervals and another one through the right ends of the even numbered intervals suffice. Hence the ratio $LGR(G)/\theta(G) = k$ which is arbitrarily large. □

Below we present two lemmas that describe some interesting properties of EGR strategy. Let S be an independent set of maximum cardinality $|S| = \alpha(G)$. We may assume that no interval from V is properly contained in any of the intervals from S; such a set S exists and can be effectively constructed

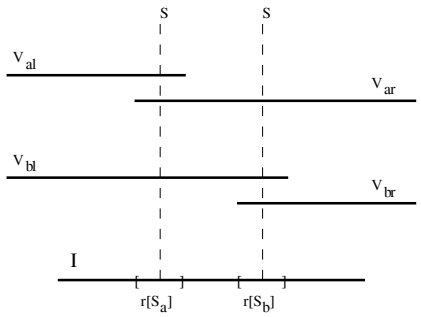


Figure 5: Ranges included in intervals.

for a given interval graph [HT91]. Denote the last property of S by Π. Recall that the size of S states a lower bound to the efficiency of *MCC* algorithms.

LEMMA 3.2. *At any stage j of computation, for any $v \in S$ there exists at most one shot s such that $r_j[s]$ is properly included in v.*

PROOF: Assume the contrary: at some moment j there are shots s_a, s_b whose ranges are properly included in some interval $v \in S$. From Lemma 3.1 it follows that without loss of generality the situation can be depicted as in Figure 5. We have $EGR(v_{a_l}) = A(v_{a_r}) = s_a$ and $EGR(v_{b_l}) = EGR(v_{b_l}) = s_b$. Observe that property Π implies that all the four intervals considered extend beyond the respective endpoints of v.

Let s_a be the shot that has been introduced earlier of the two. Then s_b could not be assigned to v_{b_l} since at the time v_{b_l} was considered by the algorithm, shot s_a had already been in use. A symmetric argument applies to the case that s_b is earlier than s_a and interval v_{a_r}, showing that the initial assumption is false. This completes the proof of the lemma. □

By replacing property Π with the maximality of the independent set S, a similar argument can be used to prove:

LEMMA 3.3. *Let $v_l = [p_l, q_l]$ and $v_r = [p_r, q_r]$ be two adjacent intervals from the set S such that $q_l < p_r$, and let $I = [q_l, p_r]$ be the gap between them. At any stage j of computation there exists at most one shot s such that $r_j[s]$ is properly included in I.*

From the above two lemmas we derive that the number of shots whose ranges are fully included either in some interval from set S or in some gap between intervals equals $2\alpha(G) + 1$, which is sufficient to prove that EGR is 4-competitive. However, we can show a stronger result.

THEOREM 3.3. $EGR(G) \leq 2\theta(G) + 1$.

PROOF: Assume that the output of algorithm EGR on an interval graph presentation $G^{\ll}$ is given. Order all of the shots from left to right and assign to them new ordinal numbers 1 through $m = EGR(G)$. Since the ranges of the shots are disjoint this assignment is well defined.

Fix $i \in \{1, \ldots, \lfloor m/2 \rfloor\}$ and let $r[2i-1] = [a_{2i-1}, b_{2i-1}]$, $r[2i] = [a_{2i}, b_{2i}]$ be the ranges of two adjacent shots, $a_{2i-1} < b_{2i-1} < a_{2i} < b_{2i}$. Without loss of generality assume that shot $2i-1$ was used for the first time before shot $2i$ was introduced. Then there exists an interval $v_{j_i} = [p_{j_i}, q_{j_i}]$ such that $EGR(v_{j_i}) = 2i$, $q_{j_i} = b_{2i}$ (i.e. v_{j_i} defines the right boundary of the range of v_{2i}), and $p_{j_i} > b_{2i-1}$ (otherwise v_{j_i} would be stabbed by the earlier shot $2i-1$).

Such an interval v_{j_i} exists for each pair of shots $2i-1$, $2i$, $i = 1, \ldots, \lfloor m/2 \rfloor$, and v_{j_i}, $i = 1, \ldots, \lfloor m/2 \rfloor$ are pairwise disjoint. They form an independent set of G of size $\lfloor m/2 \rfloor$. Therefore $\theta(G) \geq \lfloor m/2 \rfloor$, hence $EGR(G) \leq 2\theta(G) + 1$. □

Observe that the whole argument in the proof does not change if we replace intervals by arcs. Therefore we obtain the final result:

THEOREM 3.4. *There is an optimal on-line algorithm for* MCC *and* O-LSP *problems with the competitive ratio* 2.

Acknowledgement Support of the Kentucky Biomedical Research Infrastructure Grant is acknowledged.

Bibliography

[BL76] S. Booth, S. and S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, J. Comput. Syst. Sci. 13 (1976), 335-379.

[ChN99] Chaudhuri, J. and S. C. Nandy. Generalized shooter location problem. Lecture Notes in Computer Science 1627 (1999), 389-399.

[F85] Fishburn, P. C. Interval Orders and Interval Graphs: A Study of Partially Ordered Sets. New York: Wiley, 1985.

[HT91] Hsu, W.-L., and K.-H. Tsai, Linear time algorithms on circular-arc graphs, Information Processing Letters 40 (1991), 123-129.

[JK02] Jaromczyk, J. W. and M. Kowaluk, A kinetic view of the shooter problem, Proceedings of the 18th European Workshop on Computational Geometry, 2002.

[K98] Kierstead, H. A., Recursive and On-Line Graph Coloring, in Handbook of Recursive Mathematics, vol. 2, Recursive Algebra, Analysis and Combinatorics, Elsevier (1998), 1233-1269.

[KQ95] Kierstead, H. A. and J.Qin, Coloring interval graphs with First-Fit, Discrete Math. 144 (1995) 47-57.

[S89] Ślusarek, M., A coloring algorithm for interval graphs, in: Mathematical Foundations of Computer Science 1989, Lecture Notes in Computer Science 379 (1989) 471-480.

[S95] Ślusarek, M., Optimal on-line coloring of circular arc graphs, RAIRO Inform. Theor. Appl. 20 (1995) 423-429.

Convex sets in graphs

J. Cáceres [*1] A. Márquez [*2] O.R. Oellermann [†3] M.L. Puertas [*1]

The study of abstract convexity began in the early fifties with the search for an axiom system that defines a convex set and in some way generalises the classical concept of a Euclidean convex set. Numerous contributions to this topic have been made. An extensive survey of this subject can be found in [15].

Among the wide variety of structures that have been studied under abstract convexity are metric spaces, ordered sets or lattices and graphs, the last being the focus of this paper. Several abstract convexities associated with the vertex set of a graph are well-known (see [8]). Their study is of interest in Computational Geometry and has some direct applications to other areas such as, for example, Game Theory (see [4]).

For graph terminology we follow [11]; except that we use vertex instead of point and edge instead of line. All graphs considered here are finite, simple, unweighted and undirected. The *interval* between a pair u, v of vertices in a graph G is the collection of all vertices that lie on some shortest $u-v$ path in G and is denoted by $I_G[u,v]$ or $I[u,v]$ if G is understood. Intervals in graphs have been studied extensively (see [2, 13, 14]) and play an important role in the study of several classes of graphs such as the Ptolemaic graphs or block graphs. A subset S of vertices of a graph is said to be *convex* if it contains the interval between every pair of vertices in S. This definition allows us to study several problems from Euclidean convexity in a finite and discrete setting.

If S is a convex set in a graph, a vertex $p \in S$ is said to be an *extreme point* for S if $S - \{p\}$ is still convex. A vertex in a graph is *simplicial* if its neighbourhood induces a complete subgraph. So p is an extreme vertex for a convex set S if and only if p is simplicial in the subgraph induced by S.

The *convex hull* of a set S of vertices in a graph G is the smallest convex subset of G that contains S and is denoted by $\mathrm{CH}(S)$. It is true, in general, that the convex hull of the extreme points of a vertex set S is contained in S, but equality holds only in special cases. If a graph satisfies this property for every convex subset of the vertex set, it is said to have the *Minkowski-Krein-Milman property*. In [8] it is shown that a graph has this property if and only if it has no induced cycles of length bigger than 3 and has no induced 3-fan (see Figure 1).

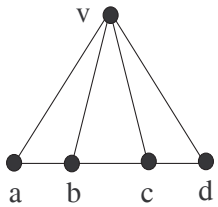


Figure 1: A 3-fan

If a graph G has the Minkowski-Krein-Milman property and S is a convex set of $V(G)$, then we

[*]Research partially support by FQM164, FQM305, BFM2001-2474 grants.
[†]Research supported by an NSERC grant CANADA.
[1]University of Almería (Spain).
[2]University of Winnipeg (Canada).
[3]University of Sevilla (Spain).

can rebuild the set S from its extreme vertices using the convex hull operation. Since this cannot be done with every graph, using only the extreme vertices of a given convex set S, it is natural to ask if it is possible to extend the set of extreme vertices of S to a set that allows us to rebuild S using the vertices in this extended set and the convex hull operation. We answer this question in the affirmative using the collection of 'contour vertices' of a set. To this end, let S be a set of vertices in a graph G and recall that the *eccentricity in* S of a vertex $u \in S$ is given by $ecc_S(u) = \max\{d(u,v) : v \in S\}$ and a vertex $v \in S$ for which $d(u,v) = ecc_S(u)$ is called an *eccentric vertex* for u in S. In case $S = V(G)$, we denote $ecc_S(u)$ by $ecc(u)$. A vertex $u \in S$ is said to be a *contour vertex* of S if $ecc_S(u) \geq ecc_S(v)$ for every neighbour v of u in S. The set of all contour vertices of S is called the *contour* of S and is denoted by $\mathrm{Ct}(S)$. If $S = V(G)$, the set is called the *contour* of G and is denoted by $\mathrm{Ct}(G)$.

The relationship between contour and extreme points is shown in the two following results.

Lemma 1. *Let G be a graph and $S \subseteq V(G)$. Then $\mathrm{Ct}(S)$ contains all extreme vertices of S.*

Proposition 2. *Let G be a distance-hereditary graph without induced 4-cycles. A vertex $x \in V(G)$ is a contour vertex for G if and only if each neighbour v of x which is on a shortest path between x and some eccentric vertex for x satisfies $N(x) \subseteq N(v)$.*

The following result shows that the convex hull of the contour set of a convex set of vertices in a graph is the entire set, without any restriction on the graph. So this result is similar to the Minkowski-Krein-Milman property and holds for all graphs.

Theorem 3. *Let G be a graph and S a convex subset of vertices. Then $S = \mathrm{CH}(\mathrm{Ct}(S))$.*

Now we characterize those graphs that are the contour of some other graph. The following results tells us which graphs are not the contour of any graph.

Proposition 4. *If H is a connected, non-complete graph with radius 1, then H is not the contour of any graph.*

On the other hand, suppose that H is a connected graph with radius greater than 1. We now describe a graph G such that its contour is H, using the construction given in [3]. Let G be the join of H and K_1. Then every vertex of H has eccentricity 2 and the vertex of $G - V(H)$ has eccentricity 1. Hence the vertices of H are precisely the contour vertices of G.

A slightly different construction allows us to obtain a graph with given disconnected contour set such that the eccentricities of the vertices in every component are given numbers at least 2. More precisely, let H be a disconnected graph with components, $H_1, H_2, \ldots, H_k$. Let $n_1, n_2, \ldots n_k$ be k natural numbers such that $n_1 = n_k = \max\{n_1, n_2, \ldots n_k\}$ and $M = \max\{n_1, n_2, \ldots n_k\} \leq 2\min\{n_1, n_2, \ldots, n_k\} = 2m$. Note that these are natural restrictions, because M will be the diameter of the graph G and m will be greater than or equal to the radius. Then there exists a connected graph G such that H is the contour of G and the eccentricity of every vertex in each component H_i of H is equal to n_i. To construct such a graph G we begin with the path $v_1 v_2 \ldots v_{M+1}$ of order $M+1$. Now replace v_1 by H_1 and v_{M+1} by H_k so that all vertices in H_1 are neighbours of v_2 and all vertices in H_k are neighbours of v_M.

Now, for each i, $2 \leq i \leq k-1$ there exists a vertex v_{n_i} on the path such that its eccentricity is $n_i - 1$. We now add H_i to the graph and join all the vertices of H_i to v_{n_i} (see Figure 2). Then $ecc(u_i) = n_i$ for all $u_i \in H_i$, and $\mathrm{Ct}(G) = H$.

In order to find the convex hull of a set S one begins by taking the union of the intervals between pairs of vertices of S, taken over all pairs of vertices in S. We denote this set by $I_G[S]$ or $I[S]$, i.e., $I[S] = \cup_{\{u,v\} \subseteq S} I[u,v]$ and call it the *geodetic closure* of S. One then repeats this procedure with the new set and continues until, for the first time, one reaches a set T for which the geodetic closure is the set itself , i.e., $T = I[T]$. This is then the convex hull of S. If this procedure only has to be performed once, we say that the set S is a *geodetic set* for its convex hull. In general a subset S of a convex set T is a *geodetic set* for T if $I[S] = T$. The notion of a geodetic set for the vertex set of a graph was first defined in [5].

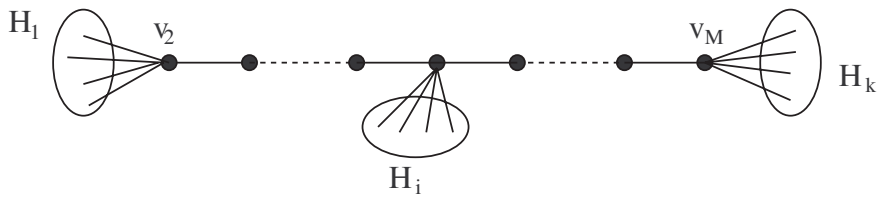


Figure 2: A disconnected contour set

We now focus on geodetic sets in 'distance hereditary graphs'. We first discuss here how these graphs are related to the graphs with the Minkowski-Krein-Milman property. Howorka [12] defined a connected graph G to be *distance hereditary* if for every connected induced subgraph H of G and every two vertices u, v in H, $d_H(u, v) = d_G(u, v)$. In the same paper several characterisations for this class of graphs are given. We state here only one of these which we will use in this paper.

Theorem 5 ([12]). *A connected graph G is distance hereditary if and only if every cycle in G of length at least 5 has a pair of crossing chords.*

Further useful characterizations for this class of graphs were established in [1, 7, 10]. Apart from having elegant characterisations, distance hereditary graphs possess other useful properties. It is a class of graphs for which several NP-hard problems have polynomial solutions. For example, it has been shown in [6, 7] that the Steiner problem for graphs, which is known to be NP-hard (see [9]), can be solved in polynomial time in distance hereditary graphs. Moreover, these graphs are Steiner distance hereditary as was shown in [7]; i.e., the Steiner distance of a set of vertices is the same, in any connected induced subgraph that contains it, as it is in the graph itself.

The class of distance hereditary graphs also properly contains the graphs that possess the Minkowski-Krein-Milman properly since a graph is chordal without an induced 3-fan if and only if it is a distance hereditary graph without an induced 4-cycle. It was shown in [8] that in a chordal graph every non-simplicial vertex lies on a chordless path between two simplicial vertices. If G is a chordless graph without an induced 3-fan, then G is distance hereditary and thus every induced path is necessarily a shortest path. Hence the simplicial vertices for a convex set S in a graph with the Minkowski-Krein-Milman property is a geodetic set for S. We show that the contour vertices of a distance hereditary graph form a geodetic set for the graph.

We need the following Lemma, that relates eccentric and contour points in distance hereditary graphs.

Lemma 6. (a) *If G is a distance hereditary graph and $x \in V(G)$, then there is an eccentric vertex for x that is a contour vertex.*
(b) *Let G be a distance hereditary graph without induced 4-cycles. If $x \in V(G)$ is such that $ecc(x) \geq 2$, then each eccentric vertex of x is a contour vertex of G.*

Theorem 7. *Let G be a distance hereditary graph. Then $\mathrm{Ct}(G)$ is a geodetic set for G.*

The graph of Figure 3 shows that Theorem 7 does not hold for graphs in general. Note that the contour set of this graph G is $\mathrm{Ct}(G) = \{v_2, v_5, w\}$ and $v_1 \notin I[\mathrm{Ct}(G)]$.

Indeed if we replace v_1 by a clique of arbitrarily large order and join every vertex in this clique with v_2 and v_8, we see that the ratio $|I[\mathrm{Ct}(G)]|/|V(G)|$ can be made arbitrarily small.

As we mentioned in the introduction, the process of taking geodetic closures starting from a set S of vertices can be repeated to obtain a sequence $S_0, S_1, \ldots$ of sets where $S_0 = S$, $S_1 = I[S]$, $S_2 = I[I[S]] \ldots$. Since $V(G)$ is finite, the process terminates with some smallest r for which $S_r = S_{r+1}$. The set S_r is then the convex hull of S and r is called the *geodetic iteration number*, $gin(S)$, of S. In the graph G of Figure 3, $gin(\mathrm{Ct}(G)) = 2$. It remains an open problem to determine if $gin(\mathrm{Ct}(G))$ can be larger than 2 and indeed if $gin(\mathrm{Ct}(G))$ can be arbitrarily large.

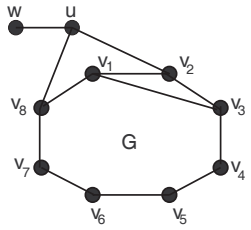


Figure 3: A graph whose contour set is not geodetic

References

[1] H.-J. Bandelt and H.M. Mulder, Distance-hereditary graphs. *J. Combin. Theory Ser. B* **41** (1986), pp. 182–208.

[2] H.-J. Bandelt and H.M. Mulder, Three interval conditions for graphs. *Twelfth British Combinatorial Conference (Norwich, 1989). Ars Combin.* **29B** (1990) pp. 213–223.

[3] H. Bielak and M. Syslo, Peripherical Vertices in Graphs. *Studia Scientiarum Mathematicarum Hungarica* **18** (1983) pp. 269–275.

[4] J.M. Bilbao and P.H. Edelman, The Shapley value on convex geometries. *Discr. Appl. Math.* **103** (2000) pp. 33–40.

[5] G. Chartrand, F. Harary and P. Zhang, Geodetic sets in graphs. *Discuss. Math. Graph Theory* **20** (2000) pp. 29–138.

[6] A.D'Atri and M.Moscarini, Distance-hereditary graphs, Steiner trees and connected domination. *SIAM J. Comput.* **17** (1988) pp. 521–538.

[7] D.P. Day, O.R. Oellermann and H.C. Swart, Steiner distance-hereditary graphs. *SIAM J. Discrete Math.* **7** (1994) pp. 437–442.

[8] M. Farber and R.E. Jamison, Convexity in graphs and hypergraphs. *SIAM J. Alg. Disc. Math.* **7** (1986) pp. 433–444.

[9] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-Completeness.* W.H. Freeman, New York, 1979.

[10] P.L. Hammer and F. Maffray, Completely seperable graphs, *Discr. Appl. Math.* **27** (1990) pp. 85–100.

[11] F. Harary, *Graph Theory.* Perseus Books, Cambridge, Massachusetts, 1969.

[12] E. Howorka, A characterization of distance hereditary graphs, *Quart. J. Math. Oxford* **28** (1977) pp. 417-420.

[13] H.M. Mulder, *The interval function of a graph.* Mathematical Centre Tracts, 132. Mathematisch Centrum, Amsterdam, 1980.

[14] L. Nebeský, Characterizing the interval function of a connected graph. *Mathematica Bohemica* **123** (1998), pp. 137–144.

[15] M.J.L. Van de Vel, *Theory of convex structures.* North-Holland, Amsterdam, 1993.

Graphs of triangulations and perfect matchings

M. Houle[1] F. Hurtado[2] M. Noy[2] E. Rivera-Campo[3]

Abstract

Given a set P of points in the plane, the graph of triangulations $\mathcal{T}(P)$ has a vertex for every triangulation of P, and two of them are adjacent if they differ by a single edge exchange. In this paper we prove that the subgraph $\mathcal{T}_{\mathcal{M}}(P)$ of $\mathcal{T}(P)$, consisting of all triangulations of P that admit a perfect matching, is connected. A main tool in our proof is a result of independent interest, namely that the graph $\mathcal{M}(P)$ that has as vertices the non-crossing perfect matchings of P and two of them are adjacent if their symmetric difference is a single non-crossing cycle, is also connected.

Keywords. Triangulation. Perfect matching. Non-crossing graph.

1 Introduction

Given a set P of points in the plane, the graph of triangulations $\mathcal{T}(P)$ has a vertex for every triangulation of P, and two of them are adjacent if they differ by a single edge exchange. Graphs of triangulations have been widely studied; see for example [5, 6]. In particular, it is well-known that $\mathcal{T}(P)$ is a connected graph.

In this paper we study the subgraph $\mathcal{T}_{\mathcal{M}}(P)$ of $\mathcal{T}(P)$, consisting of all triangulations of P that admit a perfect matching. Not every triangulation contains a perfect matching, so in general $\mathcal{T}_{\mathcal{M}}(P)$ is a proper subgraph of $\mathcal{T}(P)$. Our main result is that the graph $\mathcal{T}_{\mathcal{M}}(P)$ is connected for any set P in general position. In other words, we show that any two triangulations of P containing a perfect matching can be connected through a sequence of edge exchanges, always resulting in triangulations containing a perfect matching.

In order to prove our main result, we first prove another result of independent interest, which we now describe. Given a set P in the plane of even cardinality, a perfect matching in P is said to be non-crossing if no two of its edges intersect. The graph $\mathcal{M}(P)$ has as vertices the non-crossing perfect matchings of P, and two of them are adjacent if their symmetric difference is a single non-crossing cycle. The case were P is in convex position was studied in [4]. We show that the graph $\mathcal{M}(P)$ is connected for any set P in general position; this is the key ingredient for proving that $\mathcal{T}_{\mathcal{M}}(P)$ is a connected graph.

The rest of the paper is organized as follows. Section 2 contains the results on graphs of perfect matchings, and Section 3 on graphs of triangulations containing perfect matchings. Our graph theory terminology follows that of [2]. Throughout the paper we assume that all point sets are in general position, that is, no three points are collinear.

[1]IBM Research, Tokyo Research Laboratory, Japan, meh@trl.ibm.com.

[2]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain, hurtado@ma2.upc.es, noy@ma2.upc.es. Partially supported by Projects DGES-SEUID PB98-0933, MCYT-BFM2001-2340, MCYT-FEDER-BFM2002-0557 and Gen. Cat 2001SGR00224.

[3]Departamento de Matemáticas. Universidad Autónoma Metropolitana-I, México, erc@xanum.uam.mx. Part of the research was done while this author was on sabbatical leave visiting the Universitat Politècnica de Catalunya with grants by MECD-Spain and CONACYT-México.

2 Graphs of perfect matchings

Let P be a set of $2m$ points in general position in the plane. The symmetric difference of two non-crossing perfect matchings in P is a set of alternating cycles; some of these cycles may have crossings, see Figure 1. We say that two perfect matchings M_1 and M_2 differ in a *single alternating non-crossing cycle exchange* if their symmetric difference is a single non-crossing cycle; for brevity we say that M_2 is obtained from M_1 by performing a *flip*.

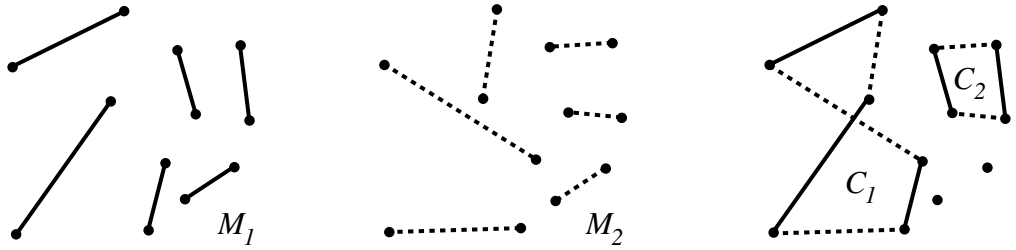


Figure 1: Two matchings M_1 and M_2; their symmetric difference (right) is the union of two alternating cycles C_1 and C_2, but only C_2 is non-crossing.

The graph of non-crossing perfect matchings $\mathcal{M}(P)$ of P is the graph with one vertex for each non-crossing perfect matching of P, in which two matchings are adjacent if and only if one can be obtained from the other by a flip. The requirement that the cycle involved in the exchange is non-crossing is not only a natural one, but it is critical when applying Theorem 2.1 in the next section.

Theorem 2.1. *For any set P of $2m$ points in general position in the plane, the graph $\mathcal{M}(P)$ is a connected graph.*

3 Graphs of triangulations

Let P be a set of points in the plane in general position. The graph of triangulations $\mathcal{T}(P)$ is the graph with one vertex for each triangulation of P, in which two triangulations T_1 and T_2 are adjacent if and only there are edges $e \in T_1 \setminus T_2$ and $f \in T_2 \setminus T_1$ such that $T_2 = T_1 \setminus \{e\} \cup \{f\}$. In other words, T_2 is obtained from T_1 by replacing the diagonal of a convex quadrilateral by the other diagonal.

For a non-crossing set E of line segments with endpoints in P, let $\mathcal{T}(P, E)$ be the subgraph of $\mathcal{T}(P)$ induced by the set of triangulations of P that contain all edges in E.

Lemma 3.1. *Let P be a set of points in general position in the plane, E be a non-crossing set of line segments with ends in P and $e \notin E$ be a line segment, also with ends in P, and such that $E \cup \{e\}$ is a non-crossing set. For each triangulation T of P that contains all edges in E there is a triangulation S of P containing $E \cup \{e\}$ which is connected to T in $\mathcal{T}(P, E)$.*

Theorem 3.2. *$\mathcal{T}(P, E)$ is a connected graph for any set P of points in general position in the plane and any non-crossing set E of line segments with ends in P.*

For a set P of $2m$ points in general position in the plane, let $\mathcal{T}_{\mathcal{M}}(P)$ be the subgraph of $\mathcal{T}(P)$, induced by the set of triangulations of P that admit a perfect matching. Notice that a set P may admit some triangulations which contain a matching while some others do not contain any (Figure 2).

Theorem 3.3. *$\mathcal{T}_{\mathcal{M}}(P)$ is a connected graph for any set P of $2m$ points in general position in the plane.*

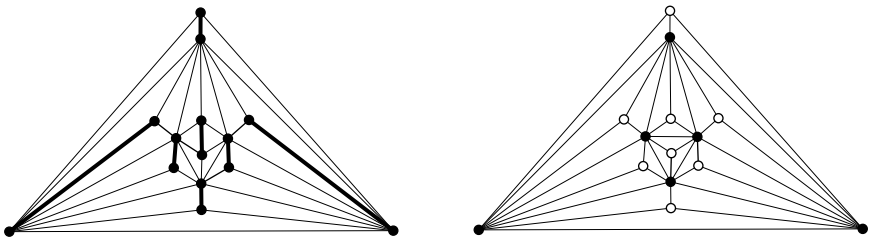


Figure 2: The triangulation on the left part of the figure contains a perfect matching (solid lines), but the triangulation on the right part does not contain any, because the 8 independent white nodes are adjacent only to the 6 black nodes.

4 Conclusions

Our definition of adjacency of the graph of non-crossing matchings $\mathcal{M}(\mathcal{P})$ of P via a single alternating non-crossing cycle exchange contains no constraint on the length of the cycle. Nevertheless, as pointed out in [3], for the purposes of optimization, enumeration, and random generation, it is desirable that the transformation making a class connected is as local as possible, which somehow amounts to use an exchange of constant size at each step. Therefore it is natural to consider a graph of matchings $\mathcal{M}'(P)$ in which only exchanges in cycles of length $\ell = 4$ (say) are considered. It is an open problem to decide whether such graph is connected for some constant value of ℓ. For $\ell = 4$ we have been able to prove that the corresponding graph contains no isolated point; yet even this modest fact required quite a long proof.

Finally, there other subgraphs of $\mathcal{T}(P)$ for which it would be interesting to know whether they induced a connected subgraph or not. For instance, the set of 3-connected triangulations of P (see [1] for a related problem), or the set of triangulations with minimum degree at least k.

References

[1] D. Avis, Generating rooted triangulations without repetitions, *Algorithmica* 16 (1996), 618–632.

[2] G. Chartrand and L. Lesniak, *Graphs and digraphs, 3d edition*, Chapman and Hall (1996).

[3] C. Hernando, M. Houle and F. Hurtado, On Local Transformation of Polygons with Visibility Properties, *Theoretical Computer Science*, to appear.

[4] C. Hernando, F. Hurtado and M. Noy, Graphs of non-crossing perfect matchings, *Graphs and Combinatorics*, to appear.

[5] F. Hurtado and M. Noy, Graph of triangulations of a convex polygon and tree of triangulations, *Computational Geometry: Theory and Applications* 13 (1999), 179–188.

[6] F. Hurtado, M. Noy and J. Urrutia, Flipping edges in triangulations, *Discrete and Computational Geometry* 22 (1999) 333–346.

Computing the Detour of Polygons

Ansgar Grüne Rolf Klein Elmar Langetepe

Abstract

Let P be a simple polygon in R^2 with n vertices. The detour of P between two points, $x, y \in P$, is the length of a shortest path contained in P and connecting x to y, divided by the distance of these points. The detour of the whole polygon is the maximum detour between any two points in P. We first analyze properties of pairs of points with maximum detour. Next, we use these properties to achieve a deterministic $O(n^2)$-algorithm for computing the maximum Euclidean detour and a deterministic $O(n \log n)$-algorithm which calculates a $(1+\varepsilon)$-approximation. Finally, we consider the special case of monotone rectilinear polygons. Their L^1-detour can be computed in time $O(n)$.

1 Introduction

Let P be a connected set in R^d. For any two points $x, y \in P$ let $d_P(x, y)$ denote the infimum of the lengths of all curves which are contained in P and connect x to y. The length of the curves is measured using a given norm $\|.\|$. The *detour* $\delta_P(x, y)$ between x and y in P with respect to $\|.\|$ and the detour $\delta(P)$ of P are defined as

$$\delta_P(x, y) := \frac{d_P(x, y)}{\|y - x\|}, \qquad \delta(P) := \sup_{x,y \in P, x \neq y} \delta_P(x, y).$$

Narasimhan and Smid [8] examined the problem of computing a value similar to $\delta(G)$ for a given Euclidean graph G. They restricted the maximum to pairs of vertices. Thus, their problem is slightly different (but not necessarily easier). The maximum of all points was first considered by Ebbers-Baumann et al. [3]. They presented an $O(n \log n)$ approximation algorithm for n-link chains in E^2. Later, Agarwal et al. [1] gave a randomized $O(n \log^3 n)$ and a deterministic $O(n \log^4 n)$ exact algorithm. Simultaneously, Langerman, Morin and Soss [6] constructed an $O(n \log n)$ randomized algorithm for solving the same problem.

In this abstract we present algorithms computing the detour of a simple polygon $P \subset \mathrm{R}^2$ where P denotes the union of the interior and the boundary. We first analyze some general properties of detour maxima, then we develop an algorithm for the Euclidean metric, and finally, we present a faster algorithm for the L^1-detour of monotone rectilinear polygons.

2 Properties of Maxima

Ebbers-Baumann et al. [3] showed for the Euclidean norm that every polygonal chain C in R^2 has a co-visible[1] *detour maximum* (p, q), i.e. $\delta_C(p, q) = \delta(C)$. The proof can easily be extended to polygons with detour $\delta(P) > 1$ and arbitrary norms.[2]

Lemma 1 *Let $P \subset \mathrm{R}^2$ be a simple polygon with $\delta(P) > 1$ where the detour is measured with respect to an arbitrary norm $\|.\|$. Then, there always exists a detour maximum $(p, q) \in P \times P$ which is co-visible in $P^{\mathbf{C}}$* [3].

[1] In this setting, (p, q) is *co-visible* iff $\overline{pq} \cap C = \{p, q\}$.
[2] Note that for Euclidean distances $\delta(P) = 1$ iff P is convex.
[3] $P^{\mathbf{C}} = \mathrm{R}^2 \setminus P$; (p, q) is *co-visible in* $P^{\mathbf{C}}$ iff $\overline{pq} \cap P = \{p, q\}$.

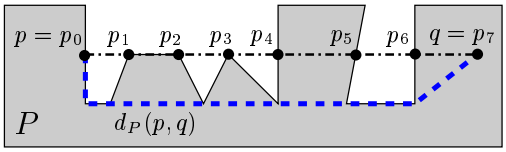


Figure 1: Boundary intersection points of $\overline{pq}$

Proof. Let (p, q) be a detour maximum. Due to $\delta(p, q) = \delta(P) > 1$, (p, q) cannot be co-visible in P. If (p, q) is co-visible in $P^{\mathbf{C}}$, the proof is done. Otherwise, let $p_0 := p$, $p_n := q$ and let $p_1, \ldots, p_{n-1}$ be the boundary intersection points of $\overline{pq}$ apart from p, q (see Fig. 1), i.e. $p_i \in \overline{pq} \cap \partial P \setminus \{p, q\}$ and p_i touches $\overline{pq} \cap P^{\mathbf{C}}$.

If the points p_i are ordered by their distance to p, we get $\|\overline{pq}\| = \sum_{i=0}^{n-1} \|\overline{p_i p_{i+1}}\|$. Additionally applying the triangle inequality of $d_P(.,.)$ yields:

$$(1) \qquad \begin{aligned} \delta_P(p, q) &= \frac{d_P(p, q)}{|\overline{pq}|} \overset{\Delta\text{-inequ.}}{\leq} \frac{\sum_{i=0}^{n-1} d_P(p_i, p_{i+1})}{\sum_{i=0}^{n-1} \|\overline{p_i p_{i+1}}\|} \\ &\leq \max_{0 \leq i \leq n-1} \frac{d_P(p_i, p_{i+1})}{\|\overline{p_i p_{i+1}}\|} = \max_{0 \leq i \leq n-1} \delta_P(p_i, p_{i+1}) \end{aligned}$$

The maximum on the right hand side is attained by a pair of points being co-visible in $P^{\mathbf{C}}$. □

For the Euclidean norm one can even show that every detour maximum of any non-convex polygon must be co-visible in $P^{\mathbf{C}}$. For the L^1-norm we will give an stricter statement in section 4.

3 Euclidean Detour of Simple Polygons

In this section, we introduce an algorithm which computes the exact Euclidean detour of a given polygon P with n vertices. Lemma 1 already allows us to restrict the search for detour maxima to the boundary of P. The following lemma further reduces the number of candidates.

Lemma 2 *Any simple polygon $P \subset \mathrm{R}^2$ has a detour maximum (p, q) which is a* vertex-boundary cut*, i.e. at least one of the points p, q is a vertex and the other one lies on the boundary ∂P.*

Lemma 2 suggests the following strategy: For every vertex p of P and every edge e of the boundary compute the local maximum $\max_{q \in e} \delta_P(p, q)$ and return the maximum of these values. However, this does not lead directly to a quadratic upper time bound because the local maximum cannot be found in constant time.

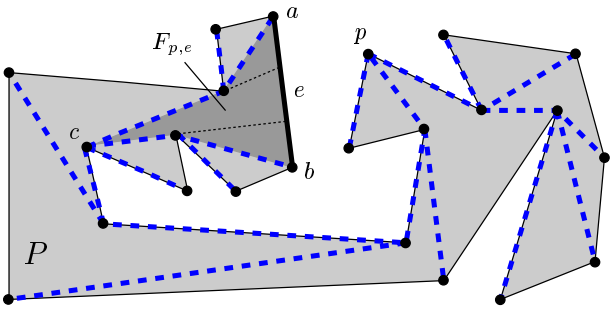


Figure 2: Shortest path tree $\mathrm{SPT}(p)$, funnel $F_{p,e}$ and its regions

To find a local maximum we consider the *funnel* $F_{p,e}$ of p and e (see Fig. 2) first examined by Lee and Preparata [7]. Let a and b be the vertices incident to e, then $F_{p,e}$ is the polygon bounded

by e and the shortest paths $\pi(a,c)$ and $\pi(b,c)$, where c is the first common vertex of $\pi(a,p)$ and $\pi(b,p)$. This vertex c is called the *cusp* of the funnel, and both paths $\pi(a,c)$ and $\pi(a,b)$ are outward convex (see [5]).

For every point $q \in e$ the shortest path $\pi(q,p)$ can be divided into $\pi(q,c)$ and $\pi(c,p)$, the first one completely contained within $F_{p,e}$. We associate with q the first vertex of $F_{p,e}$ hit by $\pi(q,p)$. Thus, if k is the number of vertices of $F_{p,e}$, the edge e will be divided into k regions $R_1, \ldots, R_k$ including the degenerate cases $R_1 := \{a\}$ and $R_k := \{b\}$ (see Fig. 2). For each such region a local maximum can be computed in $O(1)$ if $F_{p,e}$ and $|\pi(p,c)|$ are known.

Hence, a local maximum of any point p and any edge e can be computed in $O(k)$ where k is the number of vertices (or edges) of $F_{p,e}$. The funnel $F_{p,e}$ can easily be computed from the shortest path tree $\mathrm{SPT}(p)$ in $O(k)$ time by looking for the first common vertex of $\pi(a,p)$ and $\pi(b,p)$. Because every edge of the shortest path tree $\mathrm{SPT}(p)$ (see Fig. 2) can be at most on the boundary of two funnels and $\mathrm{SPT}(p)$ has $n-1$ edges, we get the value $\max_{q \in \partial P} \delta_P(p,q)$ in time $O(n)$ if $\mathrm{SPT}(p)$ is known.

Guibas et al. [5] have shown how to construct $\mathrm{SPT}(p)$ in linear time in any triangulated simple polygon. Since we can use Chazelle's [2] well-known algorithm to triangulate P in linear time, our idea leads to an algorithm computing $\max_{q \in \partial P} \delta_P(p,q)$ in $O(n)$. Thus, applying Lemma 2 yields a way to get the detour of P in $O(n^2)$.

Theorem 3 *Let $P \subset \mathrm{R}^2$ be a simple polygon with n vertices. Its maximum Euclidean detour value $\delta(P)$ and a pair of points (p,q) attaining the maximum can be computed in time $O(n^2)$.*

However, this result might not be best possible. One can transfer the approximation algorithm of Ebbers-Baumann et al. [3] to the setting of simple polygons achieving a $(1+\varepsilon)$-approximation in $O(n \log n)$. This hints that there could be a sub-quadratic solution. The complete proofs of the previous results can be found in [4].

4 L^1-Detour of Monotone Rectilinear Polygons

Within the simpler setting of monotone rectilinear[4] polygons, we can compute the L^1-detour in linear time. The main reason is a stricter statement about detour maxima proven similarly to Lemma 1:

Lemma 4 *Let $P \subset \mathrm{R}^2$ be a simple rectilinear polygon and let $(p,q) \in P \times P$ be a L^1-detour maximum. If $R(p,q)$ denotes the* bounding rectangle[5] *of p and q, its intersection with P must be empty apart from p and q, i.e. $R(p,q) \cap P = \{p,q\}$.*

It follows immediately that any L^1-detour maximum (p,q) must either be a pair of vertices or an axis-parallel pair of boundary points. In both cases, (p,q) must be co-visible in $P^{\mathbf{C}}$. If P is x-monotone[6] (y-monotone), further arguments yield that every maximum must be a horizontal (vertical) vertex-boundary cut.

W.l.o.g. let P be x-monotone. We describe an algorithm examining all upper maximum candidates, i.e. horizontal vertex-boundary cuts of the upper boundary which are co-visible in $P^{\mathbf{C}}$. The lower boundary can be treated in the same way.

The algorithm starts at the left-most vertex of the upper boundary and proceeds to the right (see Fig. 3). While moving on the boundary chain, a stack holds every previously visited *left vertical segment* s (i.e. s is vertical and P lies to the left of s) for which there has not been found any opposite right segment, yet. If the current boundary point is moving upward a vertical (right) edge, the algorithm pops the corresponding left segments and examines a horizontal pair each time it pops a vertex of a left segment or finds a vertex on the current right segment.

[4] A polygon is *rectilinear* iff every edge is either horizontal or vertical.
[5] $R(p,q) := \{r \in \mathrm{R}^2 \,|\, \min(p_x,q_x) \le r_x \le \max(p_x,q_x) \wedge \min(p_y,q_y) \le r_y \le \max(p_y,q_y)\}$.
[6] P is *x-monotone* iff its intersection with any vertical line is connected.

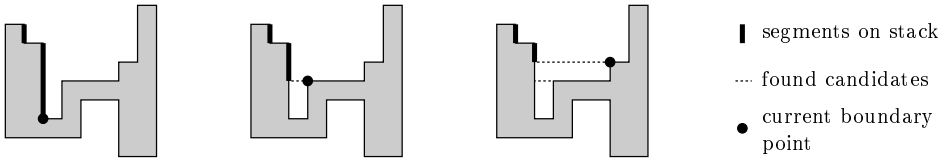


Figure 3: Some states of the algorithm for monotone rectilinear polygons

When the algorithm has found a maximum candidate (p, q), it has to calculate its detour. Let $\pi(p, q)$ be a rectilinear shortest path connecting p and q within P. The y-length $l_y(\pi(p, q))$ is the summed up length of all vertical segments of $\pi(p, q)$. The x-length $l_x(\pi(p, q))$ is defined analogously. Obviously, $d_P^{L^1}(p, q) = l_x(\pi(p, q)) + l_y(\pi(p, q))$ where $l_x(\pi(p, q)) = |p_x - q_x|$ due to P being x-monotone. Thus, for computing $\delta_P^{L^1}(p, q)$ we just need the coordinates of p and q and the value $l_y(\pi(p, q))$. The additional path information for calculating $l_y(\pi(p, q))$ can also be stored on the stack without increasing the linear time bound of the algorithm. Further details are omitted in this abstract.

Theorem 5 *Let $P \subset \mathrm{R}^2$ be an x-monotone (or y-monotone) rectilinear polygon with n vertices. An L^1-detour maximum (p, q) and its value $\delta_P^{L^1}(p, q) = \delta^{L^1}(P)$ can be computed in time $O(n)$.*

5 Open Questions

One main questions remains open: Is there a sub-quadratic algorithm computing exactly the Euclidean detour of any simple polygon or is there a quadratic lower bound? The same problem is not solved for the presumably easier setting of simple rectilinear polygons and the L^1-norm.

References

[1] P. K. Agarwal, R. Klein, C. Knauer, and M. Sharir. Computing the detour of polygonal curves. Technical report, Freie Universität Berlin, Fachbereich Mathematik und Informatik, 2002.

[2] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[3] A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas. A fast algorithm for approximating the detour of a polygonal chain. *ESA 2001 - European Symposium on Algorithms*, 2001.

[4] A. Gruene. Umwege in Polygonen. Diplomarbeit, Universität Bonn, 2002.

[5] 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.

[6] S. Langerman, P. Morin, and M. A. Soss. Computing the maximum detour and spanning ratio of planar paths, trees and cycles. *STACS 2002*, pages 250–261, 2002.

[7] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14:393–410, 1984.

[8] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM J. Comput.*, 30:978–989, 2000.

More results about spanners in the l_1-metric.*

J. Cáceres[†], C. I. Grima, A. Márquez[‡] and A. Moreno-González[§]

Abstract

In this work we study more questions about spanners in the l_1-metric. Concretely, we will see that adding some Steiner points to a set of sites the metrically complete graph of the new set has a linear number of edges. We will also characterize the free dilation trees. Finally, inspired in the work for the l_1-metric, we will study points in general position for other metrics, the λ-metrics.

1 Introduction.

There are many applications in geometric network design in which it would be interesting to find graphs with few edges that approximate shortest paths between all pair of vertices. Since in many problems, as the design of VLSI circuits, the metric that reflexes the actual distance between the vertices is the l_1-metric, in previous works [2, 3] we presented some results about the first questions that arise in the study of these graphs. Given a set of sites S in the plane, the *dilation* of a subgraph of the complete geometric graph is the largest ratio between the length of the shortest path from a pair of points of S to the distance of those points in the plane. In this way, we have presented the next results:

- It is possible to construct graphs approximating the complete Euclidean graph closely in the l_1-metric. Moreover, we found graphs that are not the complete graph but they have dilation 1 (dilation free graphs). More precisely, given a set of sites S in the plane, we call the *metrically complete graph* of S (denoted $M(S)$) to the minimal dilation free graph.
- The metrically complete graph is strictly smaller than the complete graph in the l_1-metric; in fact, if $K(S)$ denotes the complete geometric graph on S, then $|K(S)-M(S)| \in O(N^{3/2})$.
- There exists a characterization of the set of sites with a planar metrically complete graph. Also, we have found some necessary conditions for a planar graph in order to be isomorphic to a metrically complete graph.

In this work we present some additional results that continue those two mentioned works. Firstly, given a set of sites S in the plane we try to reduce the size of $M(S)$ and we will see that adding some Steiner points to S the metrically complete graph of the new set of sites has a linear number of edges. Secondly, we try to find which trees have dilation 1 in the l_1-metric, obtaining a characterization for these graphs. Finally, we try to generalize some of our first results for other metrics, the λ-metrics. In fact, we will see that the metrically complete graph of a set of sites is smaller than the complete Euclidean graph for those metrics.

*Partially supported by MCyT project BFM2001-2474

[†]Departamento de Matemática Aplicada y Estadística. Universidad de Almería. E-mail: jcaceres@ual.es

[‡]Departamento de Matemática Aplicada I. Universidad de Sevilla. E-mail: {grima,almar}@us.es

[§]Departamento de Matemáticas. Universidad de Huelva. E-mail: maria.moreno@dmat.uhu.es

2 It is possible to reduce the size of a metrically complete graph.

As we have said above, given a set of sites S in the plane, $|K(S) - M(S)| \in O(N^{3/2})$ in the l_1-metric, but, in general, $M(S)$ has a quadratic number of edges. Thus, the first question we consider is to reduce the size of $M(S)$ adding some new points to S. In order to find these points we only have to make a partition of the initial set of sites that leads to a *kd*-tree [1], (see Figure 1). Then, we add one Steiner point in the intersections of the lines used to make the partition. Then, we can prove the next result.

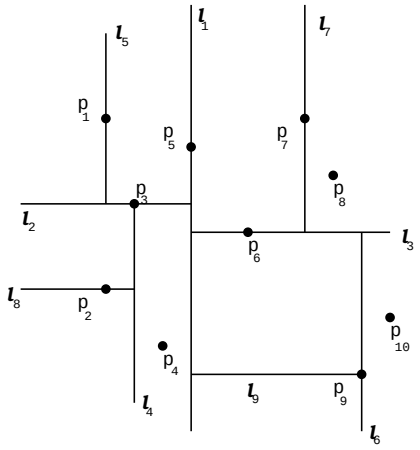


Figure 1: A partition of a set of sites.

Theorem 1 *Given a set of n sites S in the plane, there exists a linear number of Steiner points S_t verifying that $|M(S \cup S_t)| \in O(n)$.*

3 Free dilation trees.

As we have said in the Introduction, the second question we try to solve is to find which are the trees with dilation 1. In the Euclidean metric the answer to this question is very simple: we can only construct a free dilation tree when the sites are in a straight line. In the l_1-metric, some new cases appear.

Theorem 2 *If T is a free dilation tree, then T is isomorphic to one of the trees in Figure 2.*

4 Points in general position for a λ-metric.

Given a λ-metric, a ball centered in x and radio r is a regular polygon of λ edges verifying that the Euclidean distance between x and the vertices of the polygon is r.

Observe that for any value of λ there exist infinite regular polygons centered in x, so a λ-metric is not only characterized by the number of edges, but also by their orientation. However, it is only necessary to solve the question for one of them.

One of the 4-metrics is the l_1-metric, so it is natural to consider the question of constructing free dilation graphs for other values of λ. In fact, we will see that the metrically complete graph of a set of points has less edges that the complete graph in a λ-metric. In order to solve this result

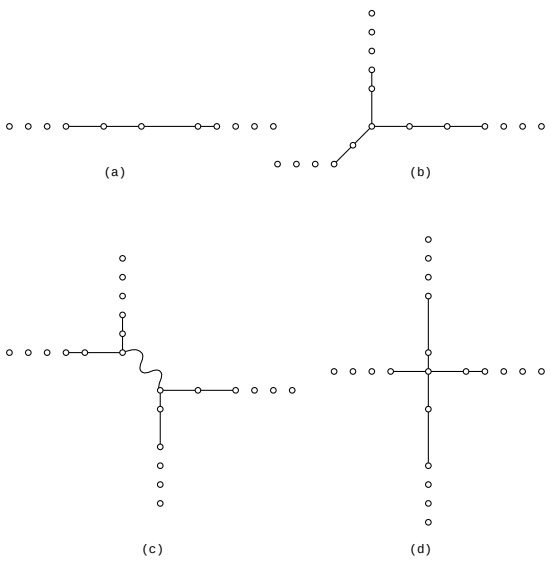


Figure 2: Free dilation trees in the l_1-metric.

we will prove that for every λ, there exists a number $n(\lambda)$ verifying that any set of points with more than $n(\lambda)$ points in general position for the Euclidean metric, is not in general position in the λ-metric. We consider that a set of points is in general position if there are not three consecutive points in a straight line.

Then, the first question we must solve is to find the minimum arc between two points. Then, let u_1, u_2 be two points in the plane and for each point u_i we consider a neighbor E_i. These neighbors make a partition of the plane in sectors centered in the initial points. If we call Sij the sector centered in u_i that contains u_j, the minimum arcs between u_i and u_j are all the arcs not decreasing parallel to the border of $Sij \bigcap Sji$, [7, 5], (see Figure 3).

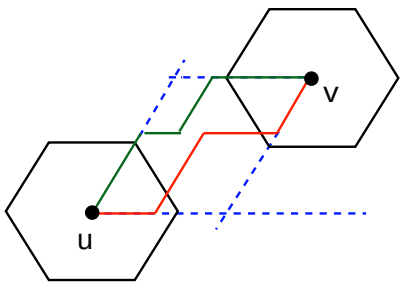


Figure 3: Two minimum arcs between u_i and u_j in a 6-metric.

Lemma 1 *Given a λ-metric, there exist, at most, λ points in convex position in general position.*

Now, in 1935 Erdös y Szekeres [4] proved that for every natural number n there exists an integer $g(n)$ verifying that for every set with more than $g(n)$ points, there are n in convex position. Then, we can prove the next result.

Theorem 3 *For any value of λ there exists $n(\lambda)$ verifying that every set of points with, at least, $n(\lambda)$ points is not in general position.*

Now, our objective is to find bounds for $n(\lambda)$. It is obvious that

$$\lambda + 1 \leq n(\lambda) \leq g(\lambda + 1),$$

and it is known [6] that

$$g(n) \leq \binom{2n-5}{n-2} + 2$$

However, this bound does not seem to be tight because for $n = 4$ we obtain 12 as upperbound and we know that $n(4) = 5$. In fact, for small values of λ, it is easy to prove that $n(\lambda) = \lambda + 1$.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517. 1975.

[2] J. Cáceres, C. I. Grima, A. Márquez and A. Moreno-González. Dilation free graphs in l_1-metric. *17th European Workshop on Computational Geometry*, Berlin. 2001.

[3] J. Cáceres, C. I. Grima, A. Márquez and A. Moreno-González. Planar graphs and metrically complete graphs. *18th European Workshop on Computational Geometry*, Warsaw. 2002.

[4] P. Erdös and G. Szekeres. A combinatorial problem in geometry. em Compositio Math., 2:463–470. 1935.

[5] R. Klein. Concrete and Abstract Voronoi Diagrams. *Lecture Notes in Computer Science. Springer-Verlag.* 1989.

[6] G. Toth and P.Valtr. Note on the Erdös-Szekeres theorem. *Rutger University. Technical Report DIMACS TR:97-31.* 1997.

[7] P. Widmayer, Y. F. Yu and C. K. Wong. Distance problems in computational geometry for fixed orientations. *Proceedings 1st ACM Symposium on Computational Geometry*, 186–195. 1985.

Approximately Matching Polygonal Curves under Translation, Rotation and Scaling with Respect to the Fréchet-Distance

Michael Clausen* and Axel Mosig
Institut für Informatik III, Universität Bonn

January 3, 2003

Abstract

Let $P : [0, m] \to \mathbb{R}^2$ and $Q : [0, n] \to \mathbb{R}^2$ be polygonal curves in the plane, G a subgroup of the affine group $\mathrm{AGL}(2, \mathbb{R})$, and $\varepsilon \geq 0$. By definition, a transformation $g \in G$ yields a (G, ε)-Fréchet-match of P and Q if the Fréchet-distance of P and the transformed version gQ of Q is at most ε. In this paper we design a c-approximation algorithm, $c > 2$, that constructs such $(G, c\varepsilon)$-Fréchet-matches for both the group G_r of rigid motions and the group G_s generated by translations and uniform scalings. We associate to P, Q and ε a certain acyclic digraph $\mathcal{M}_{m,n}$, see Fig. 1, whose edges are either weighted by closed intervals in $\mathbb{R}_{>0}$ ($G = G_s$) or by circular arcs ($G = G_r$). All maximal paths in $\mathcal{M}_{m,n}$ correspond to discrete reparametrization pairs; such a pair yields a c-approximate solution if the intervals assigned to the edges along the path have a non-empty intersection. To decide whether such a path exists, we use a dynamic programming approach, whose time complexity is $O(m^2 n^2)$. There is related work dealing with smaller subgroups of $\mathrm{AGL}(2, \mathbb{R})$: Alt and Godau [1] investigated the case $G = \{1\}$, whereas both Alt, Knauer and Wenk [2] and Efrat, Indyk and Venkatasubramanian [4] studied the case $G = T_2$, which denotes the group of translations.

1 Fréchet-Matches

A *polygonal curve of length* $m \in \mathbb{N}$ in $\mathbb{R}^2$ is defined as a continuous mapping $P : [0, m] \to \mathbb{R}^2$ with the property that for all $i \in [0 : m-1] := \{0, 1, \ldots, m-1\}$ the curve $P|_{[i,i+1]}$ is affine, i.e., $P(i + \lambda) = (1 - \lambda)P(i) + \lambda P(i + 1)$ for $\lambda \in [0, 1]$. A polygonal curve P is completely described by the sequence of its vertices $\langle p_0, \ldots, p_m \rangle$, where $p_i := P(i)$. For real numbers $x < y$ and $x' < y'$, let $\mathrm{Mon}([x, y], [x', y'])$ denote the set of all continuous, weakly increasing and surjective functions $\varphi : [x, y] \to [x', y']$; note that the surjectivity implies $\varphi(x) = x'$ and $\varphi(y) = y'$ for all $\varphi \in \mathrm{Mon}([x, y], [x', y'])$. Let P and Q be polygonal curves of lengths m and n, respectively. The *Fréchet-distance* $d_F(P, Q)$ of P and Q is defined as $d_F(P, Q) := \inf_{\alpha,\beta} \max_{t \in [0,1]} d(v(\alpha(t)), w(\beta(t)))$, where the infimum is taken over all $\alpha \in \mathrm{Mon}([0, 1], [0, m])$ and $\beta \in \mathrm{Mon}([0, 1], [0, n])$. Any subgroup G of $\mathrm{AGL}(2, \mathbb{R})$ acts on $\mathbb{R}^2$ as well as on the set of all polygonal curves P by $(gP)(t) := gP(t)$ for $g \in G$. Moreover, if $P = \langle p_0, \ldots, p_m \rangle$, then $gP = \langle gp_0, \ldots, gp_m \rangle$.

We are now ready to describe a typical question in pattern matching: given a subgroup G of $\mathrm{AGL}(2, \mathbb{R})$, two polygonal curves P and Q, and $\varepsilon \geq 0$, is there a $g \in G$ such that $d_F(P, gQ) \leq \varepsilon$? Motivated by this question, we define the set of all (G, ε)-*Fréchet-matches of* P *and* Q as

$$\mathcal{F}_G^{\varepsilon}(P, Q) := \{g \in G \mid d_F(P, gQ) \leq \varepsilon\}. \tag{1}$$

Given two polygonal curves P and Q and $\varepsilon \geq 0$, a decision algorithm for this pattern matching task outputs 1 if $\mathcal{F}_G^{\varepsilon}(P, Q) \neq \emptyset$ and 0 otherwise. Letting $c > 1$, a *c-approximation algorithm* also outputs 1 if $\mathcal{F}_G^{\varepsilon}(P, Q) \neq \emptyset$. However, the output is guaranteed to be 0 only if $\mathcal{F}_G^{c\varepsilon}(P, Q) = \emptyset$. In case $\mathcal{F}_G^{c\varepsilon}(P, Q) \setminus \mathcal{F}_G^{\varepsilon}(P, Q) \neq \emptyset$, the algorithm may answer either 0 or 1. The algorithm proposed in this paper yields c-approximate solutions for arbitrary $c > 2$. The algorithm will also be able to compute specific elements $g \in \mathcal{F}_G^{c\varepsilon}(P, Q)$ in case of output 1.

*This work was supported in part by Deutsche Forschungsgemeinschaft under grant CL 64/3

2 Approximating Fréchet-Matches by Transporter Sets

In this section $P = \langle p_0, \ldots, p_m \rangle$ and $Q = \langle q_0, \ldots, q_n \rangle$ will always denote polygonal curves of lengths m and n, respectively. P is called *reducible* if and only if, for some i, the vertex p_i is contained in the line segment $[p_{i-1}, p_{i+1}]$. Eliminating p_i from the sequence yields another curve P' with $d_F(P, P') = 0$. This elimination process finally yields an irreducible curve. In general, two polygonal curves P and P' are called *equivalent* if and only if their Fréchet-distance is zero. The Fréchet-distance defines a metric on the equivalence classes of polygonal curves. Obviously, in each equivalence class there is a unique irreducible curve. All other members of this class can be viewed as *oversamplings* of this irreducible version. In what follows, oversampling will play a crucial role. Let $\delta > 0$. A polygonal curve P is said to be *δ-sampled* if and only if $d(p_{i-1}, p_i) \leq 2\delta$, for all $i \in [1 : m]$. Given a polygonal curve P, an equivalent, δ-sampled curve P' can be constructed in an obvious way.

The notion of δ-sampled curves is a first step towards discretizing the reparametrizations α and β. We will replace $(\alpha, \beta) \in \mathrm{Mon}([0,1],[0,m]) \times \mathrm{Mon}([0,1],[0,n])$ by discrete reparametrizations $(\kappa, \lambda) \in \mathcal{I}_{m,n}$, where

$$\mathcal{I}_{m,n} := \{(\kappa, \lambda) \mid \begin{array}{l} \kappa : [0 : m+n] \to [0 : m] \text{ and } \lambda : [0 : m+n] \to [0 : m] \\ \text{are both weakly increasing and surjective}\}. \end{array} \tag{2}$$

From the facts that the index sequences κ and λ are surjective and weakly increasing, we may conclude that $\{\kappa_{s+1} - \kappa_s, \lambda_{s+1} - \lambda_s\} \subseteq \{0, 1\}$ for all $s \in [0 : n + m - 1]$. To approximate sets of Fréchet-matches we use certain transporter subsets of the group G. If d denotes the Euclidean distance in $\mathbb{R}^2$ and if P and Q have equal length, then $\tau^{G,\varepsilon}_{P,Q} := \{g \in G \mid \max_i d(p_i, gq_i) \leq \varepsilon\}$ is called the *(G, ε)-transporter of Q to P*. Similarly, $\tau^{G,\varepsilon}_{p,q} := \{g \in G \mid d(p, gq) \leq \varepsilon\}$ denotes the (G, ε)-transporter of $q \in \mathbb{R}^2$ to $p \in \mathbb{R}^2$. Obviously, $\tau^{G,\varepsilon}_{P,Q} = \cap_i \tau^{G,\varepsilon}_{p_i,q_i}$.

Theorem 2.1 *Let P and Q be δ-sampled polygonal curves of lengths m and n, respectively, with $\mathcal{F}^{\varepsilon}_G(P, Q) \neq \emptyset$. Then there exists a pair $(\kappa, \lambda) \in \mathcal{I}_{m,n}$ such that $\emptyset \neq \tau^{G,\varepsilon+\delta}_{P\circ\kappa, Q\circ\lambda} \subseteq \mathcal{F}^{\varepsilon+\delta}_G(P, Q)$.*

Unfortunately, our proof of the theorem is not completely constructive, since we require some $g \in \mathcal{F}^{\varepsilon}_G(P, Q)$ for computing (κ, λ). When matching two curves, however, such a g is what we are looking for. Thus, our algorithm for deciding whether $\mathcal{F}^{\varepsilon}_G(P, Q)$ is non-empty has to find suitable integer sequences κ and λ in a different way. A naive method that enumerates all surjective and weakly increasing candidate sequences and checks if $\tau^{G,\varepsilon+\delta}_{P\circ\kappa, Q\circ\lambda}$ is non-empty for each candidate sequence only yields an exponential-time algorithm.

3 Intersecting Projected Transporter Sets

Regarding the last theorem, both $P \circ \kappa$ and $Q \circ \lambda$ are in $(\mathbb{R}^2)^{m+n+1}$. Thus $\tau^{G,\varepsilon+\delta}_{P\circ\kappa, Q\circ\lambda}$ is the intersection of $m + n + 1$ individual transporters of the form $\tau^{G,\varepsilon+\delta}_{P(\kappa_s), Q(\lambda_s)}$. Unfortunately, these individual transporters have a rather complicated structure. In order to simplify the intersection problem we use the fact that our groups are semidirect products: $G = T_2 \rtimes H$ with $H = H_r := \mathrm{SO}(2)$ for $G = G_r$ and $H = H_s := \{\sigma E_2 \mid \sigma > 0\}$ for $G = G_s$, where E_2 denotes the 2×2 unit matrix. Thus the projection η of G onto H with kernel T_2, i.e., $\eta(th) := h$, for $t \in T_2$ and $h \in H$, is well-defined. Instead of $\tau^{G,\varepsilon}_{P,Q}$ we work with its η-image:

$$\eta^{G,\varepsilon}_{P,Q} := \eta[\tau^{G,\varepsilon}_{P,Q}] = \{h \in H \mid \exists t \in T_2 : th \in \tau^{G,\varepsilon}_{P,Q}\}.$$

Note that an analogous statement to $\tau^{G,\varepsilon}_{P,Q} = \cap_i \tau^{G,\varepsilon}_{p_i,q_i}$ does not hold for the η-images. Furthermore, for $p, q \in \mathbb{R}^2$ we always have $\eta^{G,\varepsilon}_{p,q} = H$, thus to obtain non-trivial transporters we use projected transporter sets of the form $\eta^{G,\varepsilon}_{\langle p_0,p_1\rangle,\langle q_0,q_1\rangle}$ as building blocks. To simplify notation, we let $k := m + n + 1$ and write P and Q instead of $P \circ \kappa$ and $Q \circ \lambda$.

Theorem 3.1 *Let $G \in \{G_r, G_s\}$, $k \in \mathbb{N}$, and $\varepsilon \geq 0$. For $P, Q \in (\mathbb{R}^2)^k$ and every $j \in [1 : k-1]$ define*

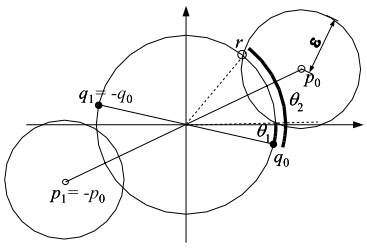
$$H_j := \bigcap_{i \in [1:j]} \eta^{G,\varepsilon}_{\langle p_{i-1}, p_i \rangle, \langle q_{i-1}, q_i \rangle} \cap \eta^{G,\varepsilon}_{\langle p_0, p_i \rangle, \langle q_0, q_i \rangle}.$$

Then $\tau^{G,\varepsilon}_{P,Q} \neq \emptyset$ implies $H_{k-1} \neq \emptyset$, whereas $\tau^{G,2\varepsilon}_{P,Q} = \emptyset$ forces $H_{k-1} = \emptyset$.

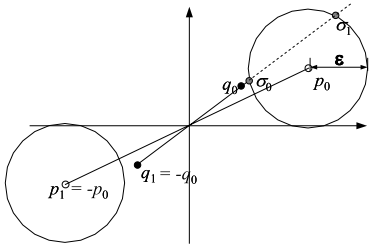
Expressed in simple terms, the preceding result states that deciding whether the intersection H_{k-1} is non-empty yields an approximate solution to the decision problem asking if the intersection $\tau^{G,\varepsilon}_{P,Q} = \cap_i \tau^{G,\varepsilon}_{p_i,q_i}$ is non-empty. Next we take a closer look at the projected (G,ε)-transporters $\eta^{G,\varepsilon}_{\langle p_0,p_1\rangle,\langle q_0,q_1\rangle}$ for $G \in \{G_r, G_s\}$.

Theorem 3.2 *Each $\eta^{G_r,\varepsilon}_{\langle p_0,p_1\rangle,\langle q_0,q_1\rangle}$ can be viewed as a circular arc on the unit circle S^1, whereas each $\eta^{G_s,\varepsilon}_{\langle p_0,p_1\rangle,\langle q_0,q_1\rangle}$ can be regarded as a closed interval on $\mathbb{R}_{>0}$.*

Sketch of Proof. One shows that it suffices to compute $\eta^{G,\varepsilon}_{\langle p_0,p_1\rangle,\langle q_0,q_1\rangle}$ for line segments $\langle p_0, p_1\rangle$ and $\langle q_0, q_1\rangle$ centered at the origin. The construction of the circular arc and the interval is illustrated in the figure below. □



$G = G_r$. Obviously, $\eta^{G,\varepsilon}_{P,Q} = [\theta_1, \theta_2]$, where $\theta_2 - \theta_1 = 2\sphericalangle(r, p_0)$ and $\theta_2 = \sphericalangle(r, q_0)$. Hence, we get $\theta_1 = \theta_2 - (\theta_2 - \theta_1) = \sphericalangle(r, q_0) - 2\sphericalangle(r, p_0)$.



$G = G_s$. The intersection of the ray Hq_0 with the disc $U_\varepsilon(p_0)$ is a line segment, and thus $\eta^{G,\varepsilon}_{P,Q}$ can be identified with the closed interval $[\|\sigma_0\|_2/\|q_0\|_2, \|\sigma_1\|_2/\|q_0\|_2]$.

In case $G = G_s$, we can easily decide whether the intersection of finitely many projected transporters is non-empty as $\cap_i [x_i, y_i] \neq \emptyset$ iff $\max_i x_i \leq \min_j y_j$.
For $G = G_r$, the projected (G,ε)-transporters are circular arcs, hence intervals on the unit circle. Such intervals differ in some respects from real intervals. For example, the intersection of two intervals on S^1 may consist of up to two disjoint intervals. An easy way to avoid the difficulties in conjunction with circular-arc intersections is to unroll S^1 — and intervals on S^1 — to the interval $[0, 2\pi]$. Unrolling an interval that covers the angle 0 requires the interval to be split into two intervals on $[0, 2\pi]$. Thus, unrolling $\eta^{G,\varepsilon}_{\langle p_0,p_i\rangle,\langle q_0,q_i\rangle} \cap \eta^{G,\varepsilon}_{\langle p_{i-1},p_i\rangle,\langle q_{i-1},q_i\rangle}$ yields up to three (disjoint) intervals in $[0, 2\pi]$.

4 An Efficient Approximate Matching Algorithm

We are now prepared to design an efficient algorithm for approximately matching two polygonal curves with respect to the Fréchet-distance under a transformation group $G \in \{G_r, G_s\}$. To this end, we introduce for δ-sampled polygonal curves P and Q of lengths m and n, respectively, the acyclic digraph $\mathcal{M}_{m,n} := (V_{m,n}, E_{m,n})$ together with a function that assigns a real interval ($G = G_s$) or up to two circular arcs ($G = G_r$) to each edge of the graph. (Efrat et al. [4] also use a graph for finding paths in free-space. However, our graph differs substantially from their construction.) The digraph $\mathcal{M}_{m,n}$, defined by

$$V_{m,n} := [0 : m] \times [0 : n] \quad \text{and} \quad E_{m,n} := \{((a,b),(c,d)) \in V^2_{m,n} \mid \{1\} \subseteq \{c-a, d-b\} \subseteq \{0,1\}\},$$

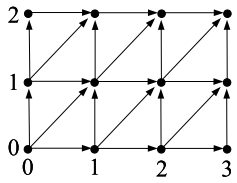


Figure 1: The digraph $\mathcal{M}_{3,2}$.

only depends on m and n, see Fig. 1, whereas the weight of the edge $e = ((a,b),(c,d)) \in E_{m,n}$ depends on $P, Q, \delta, \varepsilon$, and G:

$$I_{P,Q,\delta,\varepsilon,G}((a,b),(c,d)) := \eta^{G,\varepsilon+\delta}_{\langle p_a,p_c\rangle,\langle q_b,q_d\rangle} \cap \eta^{G,\varepsilon+\delta}_{\langle p_0,p_c\rangle,\langle q_0,q_d\rangle}.$$

Obviously, every pair $(\kappa,\lambda) \in \mathcal{I}_{m,n}$ defines (after eliminating loops) a path in $\mathcal{M}_{m,n}$ with source $(0,0)$ and sink (m,n). Conversely, every such path can be transformed into an element of $\mathcal{I}_{m,n}$ as follows: turn every vertex along the path into a pair (κ_s,λ_s) and repeat the target node of every diagonal edge, i.e., replace $(a,b) \to (a+1,b+1)$ by the subsequence $(a,b),(a+1,b+1),(a+1,b+1)$.

Now we take a closer look at the case $G = G_s$. (The case $G = G_r$ is similar, but a little bit more technical.) Here, each edge e is assigned the empty set or a closed real interval $[\ell_e, r_e]$, see Theorem 3.2. We have to find a path from $(0,0)$ to (m,n) in $\mathcal{M}_{m,n}$ such that the intersection of the involved intervals is non-empty. To decide whether such a path exists, we use the facts that $\cap_{i=1}^{N}[x_i,y_i] = [\max_i x_i, \min_i y_i]$ and that $\max_i x_i \in \{x_1,\ldots,x_N\}$. In particular, as $\mathcal{M}_{m,n}$ has $3mn+m+n$ edges, we have at most $3mn+m+n$ different left borders to consider. For each possible left border ℓ we define a new $0-1$ weight on the edges: edge e has weight 1 iff ℓ is contained in $[\ell_e, r_e]$. By dynamic programming one can test in time $O(mn)$ whether there is a path from $(0,0)$ to (m,n) involving only edges with weight 1.

Theorem 4.1 *For $G = G_s$, there is an algorithm that on input $P, Q, m, n, \delta, \varepsilon$ (with the above meaning) computes an element $g \in \mathcal{F}_G^{2(\varepsilon+\delta)}(P,Q)$ if $\mathcal{F}_G^{\varepsilon}(P,Q) \neq \emptyset$ and computes the output 0 if $\mathcal{F}_G^{2(\varepsilon+\delta)}(P,Q) = \emptyset$. Its running time is $O(m^2n^2)$.*

Thus there is a c-approximation algorithm for determining whether the set of (G_s,ε)-Fréchet-matches is non-empty, for $c = 2(1+\delta/\varepsilon)$. The same result holds for $G = G_r$.

5 Final Remarks and Future Work

The systematic use of group transporter sets is the basis of a new technique that generalizes the concept of inverted files from full-text retrieval. It has been successfully applied to content-based multimedia retrieval, see [3]. In the present work this concept has been extended to (G,ε)-transporters. We are currently investigating variants of the described algorithm, including matching curves partially as well as matching under other subgroups of $\mathrm{AGL}(2,\mathbb{R})$, in particular the group of similarity transformations generated by translations, rotations and uniform scalings.

References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[2] H. Alt, C. Knauer, and C. Wenk. Matching polygonal curves with respect to the Fréchet distance. In *Proc. 18th Int. Symp. on Theoretical Aspects of Computer Science*, pages 63–74, 2001.

[3] M. Clausen and F. Kurth. A Unified Approach to Content-Based and Fault Tolerant Music Recognition. *IEEE Transactions on Multimedia*, 2003. To appear.

[4] A. Efrat, P. Indyk, and S. Venkatasubramanian. Pattern matching with sets of segments. In *Proc. 12th ACM Symp. on Discrete Algorithms*, January 2001.

The polytope of non-crossing graphs on a planar point set *

David Orden, Francisco Santos

Universidad de Cantabria. Departamento de Matemáticas, Estadística y Computación.
Av. Los Castros, s/n E-39005 Santander, SPAIN. {ordend,santos}@matesco.unican.es

Key Words: pseudo-triangulation, flip, graph, crossingness, marks, polytope.

1 Introduction

The set of (straight-line, or geometric) non-crossing graphs with a given set of vertices $\mathcal{A}$ in the plane is of interest in Computational Geometry, Geometric Combinatorics, and related areas. In particular, much effort has been directed towards enumeration, counting and optimization on the set of maximal such graphs, that is to say, triangulations of $\mathcal{A}$. But little is known about the poset structure of the set of all non-crossing subgraphs under inclusion. In this paper we associate to $\mathcal{A}$ a polytope whose face poset contains the poset of non-crossing graphs on $\mathcal{A}$ embedded in a very nice way.

The construction is based on [5], where a polyhedron $\overline{X_f}(\mathcal{A})$ of dimension $2n-3$ with face poset (opposite to) that of *pointed* non-crossing graphs on $\mathcal{A}$ is constructed. A straight-line graph embedded in the plane is called pointed if the edges incident to every vertex span an angle smaller than 180 degrees. Let n denote the size of $\mathcal{A}$, n_b and n_i the number of points in the boundary and the interior of its convex hull, respectively. The polyhedron $\overline{X_f}(\mathcal{A})$ has a unique maximal bounded face $X_f(\mathcal{A})$, of dimension $2n_i + n_b - 3$, there called the *polytope of pointed pseudo-triangulations* of $\mathcal{A}$.

Our main new ingredient is that we consider "marked" non-crossing graphs, meaning non-crossing graphs together with the specification of a subset of their pointed vertices. With similar ideas but with n extra coordinates for the n possible marks, we get a polyhedron $\overline{Y_f}(\mathcal{A})$ of dimension $3n-3$ with a unique maximal bounded face $Y_f(\mathcal{A})$ of dimension $3n_i + n_b - 3$. The face F in the statement of Theorem 2.5 is precisely the polytope $X_f(\mathcal{A})$, which arises by setting to 0 the n new coordinates, corresponding to marks.

The technical tools both in our construction and in [5] are *pseudo-triangulations* of planar point sets and their relation to structural rigidity of non-crossing graphs. Pseudo-

*Research partially supported by project BMF2001-1153 of Spanish *Dirección General de Investigación Científica*.

triangulations, first introduced by Pocchiola and Vegter around 1995 (see [3]), have by now been used in many Computational Geometry applications, among them visibility [4, 3], ray shooting [1], and kinetic data structures [2]. Streinu [6] introduced the *minimum* or *pointed pseudo-triangulations*, and used them to prove the Carpenter's Rule Theorem. Pointed pseudo-triangulations turn out to coincide with the maximal non-crossing and pointed graphs; that is to say, with the vertices of the polytope $X_f(\mathcal{A})$. Our method extends that construction and $Y_f(\mathcal{A})$ is the *polytope of pseudo-triangulations*, since its vertices correspond to all pseudo-triangulations of $\mathcal{A}$.

2 Overview of the method and results

In the sequel we assume our point set $\mathcal{A}$ to be in general position, although the results are proved in the full paper for point sets in degenerate position as well. Let n_i and n_b be the number of points of $\mathcal{A}$ in the interior and boundary of $\text{conv}(\mathcal{A})$, respectively, and let $n = n_i + n_b$. We define:

Definition 2.1 (Flips in pseudo-triangulations) Let T be a pseudo-triangulation of $\mathcal{A}$. We call *flips* in T the following three types of operations, all producing pseudo-triangulations.

- (*Deletion flip*). The removal of an edge $e \in T$, if $T \setminus e$ is a pseudo-triangulation.
- (*Insertion flip*). The insertion of an edge $e \notin T$, if $T \cup e$ is a pseudo-triangulation.
- (*Diagonal flip*). The exchange of an edge $e \in T$, if $T \setminus e$ is not a pseudo-triangulation, for the unique edge e' such that $(T \setminus e) \cup e'$ is a pseudo-triangulation.

The *graph of pseudo-triangulations* of $\mathcal{A}$ has as vertices all the pseudo-triangulations of $\mathcal{A}$ and as edges all flips of any of the types.

Proposition 2.2 *The graph of pseudo-triangulations of $\mathcal{A}$ is connected and regular of degree $3n_i + n_b - 3 = 3n - 2n_b - 3$.*

As happened with pointed pseudo-triangulations, Proposition 2.2 suggests that the graph of pseudo-triangulations of $\mathcal{A}$ may be the skeleton of a simple polytope of dimension $3n_i + n_b - 3$. As a step towards this result we look at what the face poset of such a polytope should be. The polytope being simple means that we want to regard each pseudo-triangulation T as the upper bound element in a Boolean poset of order $3n - 3 - 2n_b$. This number equals the number of interior edges plus interior pointed vertices in T:

Definition 2.3 A *marked graph* on $\mathcal{A}$ is a geometric graph with vertex set $\mathcal{A}$ together with a subset of its vertices, that we call "marked". We call a marked graph *non-crossing* if it is non-crossing as a graph and marks arise only in pointed vertices.

We call a non-crossing marked graph *fully-marked* if it is marked at all pointed vertices. If, in addition, it is a pseudo-triangulation, then we call it a *fully-marked pseudo-triangulation*, abbreviated as *f.m.p.t.*

We start defining a linear cone $\overline{Y_0}(\mathcal{A})$ by one inequality for each possible edge and each point of $\mathcal{A}$. Its $\binom{n+1}{2}$ facets are then translated using the entries of a vector f in $\mathbb{R}^{\binom{n+1}{2}}$ to produce a polyhedron $\overline{Y_f}(\mathcal{A})$ which has as unique maximal bounded face a polytope $Y_f(\mathcal{A})$. Our proof goes by analyzing the necessary and sufficient conditions for f to produce a polytope with the desired properties. We get the next result, where flips in fully-marked pseudo-triangulations are defined in the natural way from those in pseudo-triangulations:

Theorem 2.4 (The polyhedron of marked non-crossing graphs) *If f is a valid choice of parameters, then there is a simple polyhedron $\overline{Y_f}$ of dimension $3n-3$ whose face poset equals (the opposite of) the poset of non-crossing marked graphs on $\mathcal{A}$. In particular:*

(a) *Vertices of the polyhedron are in 1-to-1 correspondence with fully-marked pseudo-triangulations of $\mathcal{A}$.*

(b) *Bounded edges correspond to flips of interior edges or marks in fully-marked pseudo-triangulations, i.e., to fully-marked pseudo-triangulations with one interior edge or mark removed.*

(c) *Extreme rays correspond to fully-marked pseudo-triangulations with one convex hull edge or mark removed.*

We prove valid choices of f to be the interior of a convex polyhedron defined by $\binom{n}{4}$ strict inequalities and give an explicit choice. Then, from the existence of a valid f and the Theorem above, the following result is concluded:

Theorem 2.5 (The polytope of all pseudo-triangulations) *Let $Y_f(\mathcal{A})$ be the face of $\overline{Y_f}(\mathcal{A})$ defined turning into equalities the equations which correspond to convex hull edges or convex hull points of $\mathcal{A}$, and assume f to be a valid choice. Then:*

1. *$Y_f(\mathcal{A})$ is a simple polytope of dimension $3n-2n_b-3$ whose 1-skeleton is the graph of pseudo-triangulations of $\mathcal{A}$. (In particular, $Y_f(\mathcal{A})$ is the unique maximal bounded face of $\overline{Y_f}(\mathcal{A})$).*

2. *Let F be the face of $Y_f(\mathcal{A})$ defined by turning into equalities the equations corresponding to interior points. Then, the complement of the star of F in the proper face-poset of $Y_f(\mathcal{A})$ equals the poset of non-crossing graphs on $\mathcal{A}$ which use all the convex hull edges.*

This result, which proves the claims advanced in the introduction, deserves some words of explanation:

- Since convex hull edges are irrelevant to crossingness, the poset of *all* non-crossing graphs on $\mathcal{A}$ is the direct product of the poset in the statement and a Boolean poset of rank n_b.

- The equality of posets in Theorem 2.5.2 reverses inclusions. Maximal non-crossing graphs (triangulations of $\mathcal{A}$) correspond to minimal faces (vertices of $Y_f(\mathcal{A})$).

- By "proper" face poset of a polytope we mean that the polytope itself is not considered a face. We remind the reader that the *star* of a face F is the union of all the facets (maximal proper faces) containing F.

- We give a fully explicit facet description of $Y_f(\mathcal{A})$. It lives in $\mathbb{R}^{3n}$ and is defined by 3 linear equalities and $\binom{n}{2} + n$ linear inequalities in which those $2n_b$ corresponding to convex hull edges and vertices of $\mathcal{A}$ have to be turned into equalities, thus providing an affine subspace of dimension $3n-3-2n_b$, as stated. The face F is the one obtained turning into equalities also the n_i equations corresponding to interior points.

It is worth mentioning that our results have some rigid-theoretic consequences. Namely:

Theorem 2.6 *Let T be a pseudo-triangulation of a planar point set A. Let G be its underlying graph. Then:*

1. *G is infinitesimally rigid, hence rigid and generically rigid.*
2. *There are at least $2k+3l$ edges of T incident to any subset of k pointed plus l non-pointed vertices of T.*

If the pseudo-triangulation is pointed, then it has $2n-3$ edges and parts (2) is just the Laman characterization of isostatic graphs in the plane as graphs with $2n-3$ edges with every k vertices incident to at least $2k$ edges. In particular, Theorem 2.6 generalizes Ileana Streinu's result [6] that pointed pseudo-triangulations are isostatic graphs.

References

[1] M. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *J. Algorithms*, **23** (1997) 51–73.

[2] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry and Applications*, 12:3–27, 2002.

[3] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete and Computational Geometry*, 16:419–453, 1996.

[4] M. Pocchiola and G. Vegter. The visibility complex. *Int. J. Comput. Geom. Appl.*, 6:279–308, 1996.

[5] G. Rote, F. Santos, and I. Streinu. Expansive motions and the polytope of pointed pseudo-triangulations, preprint 2002, `http://arXiv.org/abs/math.CO/0206027`. To appear in *Discrete and Computational Geometry – The Goodman-Pollack Festschrift*, (B. Aronov, S. Basu, J. Pach, M. Sharir, eds), Algorithms and Combinatorics, Springer Verlag, Berlin.

[6] I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proc. 41st Ann. Symp. on Found. of Computer Science (FOCS 2000), Redondo Beach, California*, pages 443–453, 2000.

Affine representations of abstract convex geometries

Kenji Kashiwabara* Masataka Nakamura† Yoshio Okamoto‡

Abstract

A convex geometry is a combinatorial abstract model introduced by Edelman and Jamison which captures a combinatorial essence of "convexity" shared by some structures including finite point sets, partially ordered sets, trees, rooted graphs. In this paper, we introduce a generalized convex shelling, and we show that any convex geometry can be represented as a generalized convex shelling. This is "the representation theorem for convex geometries" similar to "the representation theorem for oriented matroids" by Folkman and Lawrence. An important feature is that our representation theorem is affine-geometric while that for oriented matroids is topological. Namely our representation theorem indicates the intrinsic simplicity of convex geometries.

1 Introduction

Some abstract models of geometric concepts are known to be useful. For example, a matroid is considered as the abstraction of linear dependence and plays important roles in finite geometry, coding theory, combinatorial optimization and so on [11]. Another example is an oriented matroid, which is considered as the abstraction of affine (and linear) dependence and which captures essences of convex polytopes, point configurations, hyperplane arrangements and so on [1]. Oriented matroids play an important role in theory of convex polytopes, discrete geometry, computational geometry and so on, and they are known to be quite powerful models.

One of the most important theorems in oriented matroid theory is the "topological representation theorem" by Folkman and Lawrence [7]. The topological representation theorem states that: any simple oriented matroid can be represented as a "pseudohyperplane arrangement." So, in principle, when we investigate an oriented matroid, we only have to look at the corresponding pseudohyperplane arrangement. A recent study by Swartz [12] revealed the topological representation of matroids, saying that every simple matroid can be represented as the arrangement of homotopy spheres.

In this paper, we will study yet another example of combinatorial abstraction of geometric concepts, namely a convex geometry. A convex geometry was introduced by Edelman and Jamison [6] as an abstraction of convexity, and it can be seen as a "dual" (or a "polar" or a "complement") of an antimatroid [4]. A convex geometry has been appearing in papers not only on discrete geometry or combinatorics but also on social choice theory ([10] for example) or mathematical psychology ([5] for a detailed treatment). Also, convex geometries form a greedily solvable special case of a certain optimization problem [2].

In this paper, we will show a representation theorem for convex geometries. The theorem says that any convex geometry can be represented as a "generalized convex shelling." Since a generalized convex shelling is defined in a purely affine-geometric manner, this theorem gives an affine-geometric representation of a convex geometry. Since neither an affine-geometric representation theorem for matroids nor for oriented matroids is known, our affine-

*Department of Systems Science, Graduate School of Arts and Sciences, The University of Tokyo, 3–8–1, Komaba, Meguro, Tokyo, 153–8902, Japan. E-mail: kashiwa@graco.c.u-tokyo.ac.jp.

†Department of Systems Science, Graduate School of Arts and Sciences, The University of Tokyo, 3–8–1, Komaba, Meguro, Tokyo, 153–8902, Japan. E-mail: nakamura@klee.c.u-tokyo.ac.jp.

‡Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich, ETH Zentrum, CH-8092, Zürich, Switzerland. E-mail: okamotoy@inf.ethz.ch. Supported by the Berlin-Zürich Joint Graduate Program "Combinatorics, Geometry, and Computation" (CGC), financed by ETH Zürich and the German Science Foundation (DFG).

geometric representation theorem for convex geometries indicates the intrinsic simplicity of convex geometries. As well as the topological representation theorem for oriented matroids plays a significant role in theory of oriented matroids, our theorem will play a similar role in theory of convex geometries.

2 Convex geometries and the representation theorem

In this section, we will give a definition of a convex geometry, which was introduced by Edelman and Jamison [6], and will state our theorem precisely.

Let E be a nonempty finite set. A family $\mathcal{L}$ of subsets of E is called a *convex geometry* on E if $\mathcal{L}$ satisfies the following three axioms:

(L1) $\emptyset \in \mathcal{L}$ and $E \in \mathcal{L}$;

(L2) if $X, Y \in \mathcal{L}$, then $X \cap Y \in \mathcal{L}$;

(L3) if $X \in \mathcal{L} \setminus \{E\}$ then there exists some $e \in E \setminus X$ such that $X \cup \{e\} \in \mathcal{L}$.

Two convex geometries $\mathcal{L}_1$ on E_1 and $\mathcal{L}_2$ on E_2 are *isomorphic* if there exists a bijection $\psi : E_1 \to E_2$ such that $\psi(X) \in \mathcal{L}_2$ if and only if $X \in \mathcal{L}_1$.

Let us look at some examples of convex geometries.

Example 2.1 (convex shelling). Let Q be a finite set of points in $\mathbb{R}^d$, and define

$$\mathcal{L} = \{X \subseteq Q : \mathrm{conv}(X) \cap Q = X\}.$$

Then, we can see that $\mathcal{L}$ is a convex geometry on Q, and we say this kind of convex geometries is a *convex shelling*. A convex geometry isomorphic to some convex shelling on a finite point set Q is also called a convex shelling.

Example 2.2 (poset shelling). Let E be a partially ordered set endowed with a partial order $\preceq$, and define $\mathcal{L} = \{X \subseteq E : e \in X$ and $f \preceq e$ imply $f \in X\}$. Then we can see that $\mathcal{L}$ is a convex geometry on E, and we say this kind of convex geometries is a *poset shelling*.

Example 2.3 (tree shelling). Let V be the vertex set of a (graph-theoretic) tree T, and define $\mathcal{L} = \{X \subseteq V :\ u, v \in X \Rightarrow$ a unique path connecting u and v only uses vertices in $X\}$. Then we can see that $\mathcal{L}$ is a convex geometry on V, and we say this kind of convex geometries is a *tree shelling*.

Example 2.4 (graph search). Let $G =(V,E)$ be a rooted connected graph with root $r \in V$, and define $\mathcal{L} = \{X \subseteq V \setminus \{r\} : v \in V \setminus X \Rightarrow$ v can be reached from r by a path only using vertices in $V \setminus X\}$. Then we can see that $\mathcal{L}$ is a convex geometry on $V \setminus \{r\}$, and we say this kind of convex geometries is a *graph search*.

For other various examples of convex geometries, see [6] or [9].

Here, we will give yet another example of convex geometries, which was not given explicitly before.

Example 2.5 (generalized convex shelling). Let P and Q be finite point sets in $\mathbb{R}^d$. Assume that $\mathrm{conv}(P) \cap Q = \emptyset$ and particularly that $P \cap Q = \emptyset$. Then define

$$\mathcal{L} = \{X \subseteq Q : \mathrm{conv}(X \cup P) \cap Q = X\}.$$

We say $\mathcal{L}$ is the *generalized convex shelling on Q with respect to P*. If $P = \emptyset$, this just gives a convex shelling. So, as the name indicates, a generalized convex shelling is a generalization of a convex shelling. While at first sight it is not so obvious that a generalized convex shelling is indeed a convex geometry, that can be shown. (Here we omit the proof.)

Our main theorem will be as follows. This says that the class of convex geometries coincides with the class of generalized convex shellings, although convex geometries arise from diverse objects as we saw.

Theorem 2.1. *Any convex geometry is isomorphic to some generalized convex shelling.*

The main concern of this paper is the proof of Theorem 2.1. In the next section, for the proof of Theorem 2.1, we will construct finite sets P_0 and Q_0 of points from a given convex geometry $\mathcal{L}$ so that $\mathcal{L}$ can be isomorphic to the generalized convex shelling on Q_0 with respect to P_0.

3 Construction of point sets

For our construction, we will use rooted circuits of a convex geometry. So at the beginning of this section, we will introduce rooted circuits.

A rooted circuit of a convex geometry was originally introduced by Korte and Lovász [8].

In order to define a rooted circuit, we need some more technical words. For a convex geometry $\mathcal{L}$ on E and $A \subseteq E$, the *trace* of $\mathcal{L}$ on A is defined as $\mathrm{Tr}(\mathcal{L}, A) = \{X \cap A : X \in \mathcal{L}\}$. A *rooted set* is a pair (X, r) of a set X and an element r of X. A *rooted subset* of E is a rooted set (X, r) such that $X \subseteq E$.

Here comes the definition of a rooted circuit. Let $\mathcal{L}$ be a convex geometry on E. A rooted subset (C, r) of E is called a *rooted circuit* of $\mathcal{L}$ if $\mathrm{Tr}(\mathcal{L}, C) = 2^C \setminus \{C \setminus \{r\}\}$. We denote the family of rooted circuits of a convex geometry $\mathcal{L}$ by $\mathcal{C}(\mathcal{L})$.

Now we are ready for our construction. We will construct point sets P_0 and Q_0 from a given convex geometry $\mathcal{L}$ on E so that $\mathcal{L}$ can be isomorphic to the generalized convex shelling on Q_0 with respect to P_0.

Let us say that $|E| = n$. We will take the $(n-1)$-dimensional space $\mathbb{R}^{n-1}$. For each element $e \in E$, we take a point $\boldsymbol{q}(e) \in \mathbb{R}^{n-1}$ such that the points $\boldsymbol{q}(e) \in \mathbb{R}^{n-1}$ $(e \in E)$ can be affinely independent. Namely, it should hold that for any $\{\mu_e \in \mathbb{R} : e \in E\}$ with $\sum_{e \in E} \mu_e = 0$,

$$\sum_{e \in E} \mu_e \boldsymbol{q}(e) = \boldsymbol{0} \;\Rightarrow\; \mu_e = 0 \text{ for all } e \in E.$$

(So $\mathrm{conv}(\{\boldsymbol{q}(e) : e \in E\})$ is an $(n-1)$-dimensional simplex.) Also for each rooted circuit $(C, r) \in \mathcal{C}(\mathcal{L})$ of $\mathcal{L}$ we put a point $\boldsymbol{p}(C, r) \in \mathbb{R}^{n-1}$ determined as

$$\boldsymbol{p}(C, r) = |C|\boldsymbol{q}(r) - \sum_{e \in C \setminus \{r\}} \boldsymbol{q}(e). \qquad (1)$$

Note that $\boldsymbol{q}(r)$ lies in the relative interior of $\mathrm{conv}(\{\boldsymbol{q}(e) : e \in C \setminus \{r\}\} \cup \{\boldsymbol{p}(C, r)\})$ for any rooted circuit $(C, r) \in \mathcal{C}(\mathcal{L})$. In this way, we have set up $|E| + |\mathcal{C}(\mathcal{L})|$ points in $\mathbb{R}^{n-1}$.

Let $P_0 = \{\boldsymbol{p}(C, r) : (C, r) \in \mathcal{C}(\mathcal{L})\}$ and $Q_0 = \{\boldsymbol{q}(e) : e \in E\}$. Then $P_0 \cap Q_0 = \emptyset$. Now our claim is as follows.

Claim 3.1. *For P_0 and Q_0 constructed above, the generalized convex shelling on Q_0 with respect to P_0 is isomorphic to $\mathcal{L}$.*

This claim proves Theorem 2.1.

To illustrate the construction, we will look at examples for $n = 3$. For $n = 3$ we have six non-isomorphic convex geometries. Let $E = \{1, 2, 3\}$ for example. Below we enumerate all of the six non-isomorphic convex geometries on $\{1, 2, 3\}$ together with their rooted circuits. $\mathcal{L}_1 = 2^{\{1,2,3\}}$ and $\mathcal{C}(\mathcal{L}_1) = \emptyset$; $\mathcal{L}_2 = \mathcal{L}_1 \setminus \{\{1, 3\}\}$ and $\mathcal{C}(\mathcal{L}_2) = \{(\{1, 2, 3\}, 2)\}$; $\mathcal{L}_3 = \mathcal{L}_2 \setminus \{\{3\}\}$ and $\mathcal{C}(\mathcal{L}_3) = \{(\{2, 3\}, 2)\}$; $\mathcal{L}_4 = \mathcal{L}_3 \setminus \{\{2, 3\}\}$ and $\mathcal{C}(\mathcal{L}_4) = \{(\{1, 3\}, 1), (\{2, 3\}, 2)\}$; $\mathcal{L}_5 = \mathcal{L}_3 \setminus \{\{1\}\}$ and $\mathcal{C}(\mathcal{L}_5) = \{(\{1, 2\}, 2), (\{2, 3\}, 2)\}$; $\mathcal{L}_6 = \mathcal{L}_4 \setminus \{\{2\}\}$ and $\mathcal{C}(\mathcal{L}_6) = \{(\{1, 2\}, 1), (\{1, 3\}, 1), (\{2, 3\}, 2)\}$.

Figure 1 depicts the construction of the point sets for these examples.

4 Idea of the proof

Because of the limitation of the pages, we will just describe an idea of the proof of Claim 3.1. The entire proof will appear in the full-paper version. In this section, any proof will be omitted.

Let $\mathcal{L}'$ be the generalized convex shelling on Q_0 with respect to P_0. The first thing that we should care about is that the constructed point sets P_0 and Q_0 actually satisfy the precondition of generalized convex shellings, namely $\mathrm{conv}(P_0) \cap Q_0 = \emptyset$. In fact, we can show that this is the case.

Next, we want to establish a bijection ψ from E to Q_0 such that ψ can be an isomorphism between $\mathcal{L}$ and $\mathcal{L}'$. As it is natural, we will set $\psi(e) = \boldsymbol{q}(e)$ for each $e \in E$. We want to show that ψ is an expected isomorphism between $\mathcal{L}$ and $\mathcal{L}'$.

To show that, we will use a result by Dietrich [3, 4] which is a characterization of a convex geometry in terms of the family of rooted circuits. Therefore, in order to show that ψ is an isomorphism, we will show that ψ maps a rooted circuit of $\mathcal{L}$ to a rooted circuit of $\mathcal{L}'$ bijectively. From the characterization by Dietrich [3, 4], we can find that it suffices to show the following two lemmas for our purpose.

Lemma 4.1. *1. In the setting above, for any rooted circuit $(C, r) \in \mathcal{C}(\mathcal{L})$, there exists $(C', \boldsymbol{r'}) \in \mathcal{C}(\mathcal{L}')$ such that $C' \subseteq \psi(C)$ and $\boldsymbol{r'} = \psi(r)$.*

2. In the setting above, for any rooted circuit $(C', \boldsymbol{r'}) \in \mathcal{C}(\mathcal{L}')$, there exists $(C, r) \in \mathcal{C}(\mathcal{L})$ such that $C \subseteq \psi^{-1}(C')$ and $r = \psi^{-1}(\boldsymbol{r'})$.

We need more facts to prove Lemma 4.1. Actually, for a proof of Lemma 4.1.2 we use the concept of a closure operator which appears

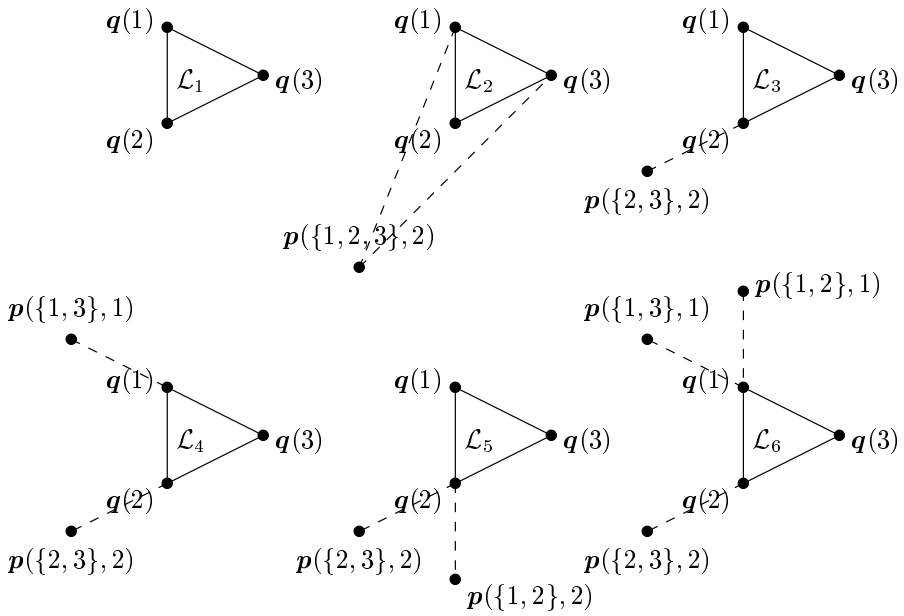


Figure 1: Construction of point sets for $n = 3$.

in the theory of convex geometries (or closure spaces more generally).

Acknowledgements

The authors are grateful to Masahiro Hachimori and Tadashi Sakuma for discussion and useful comments.

References

[1] A. Björner, M. Las Vergnas, B. Sturmfels, N. White and G. Ziegler: Oriented Matroids (2nd Edition). Cambridge University Press, Cambridge, 1999.

[2] E.A. Boyd and U. Faigle: An algorithmic characterization of antimatroids. Discrete Applied Mathematics **28**, 1990, 197–205.

[3] B.L. Dietrich: A circuit set characterization of antimatroids. Journal of Combinatorial Theory Series B **43**, 1987, 314–321.

[4] B.L. Dietrich: Matroids and antimatroids — a survey. Discrete Mathematics **78**, 1989, 223–237.

[5] J.-P. Doignon and J.-C. Falmagne: Knowledge spaces. Springer Verlag, Berlin, 1999.

[6] P.H. Edelman and R.E. Jamison: The theory of convex geometries. Geometriae Dedicata **19**, 1985, 247–270.

[7] J. Folkman and J. Lawrence: Oriented matroids. Journal of Combinatorial Theory Series B **25**, 1978, 199–235.

[8] B. Korte and L. Lovász: Shelling structures, convexity, and a happy end. In: B. Bollóbas, ed., Graph Theory and Combinatorics: Proceedings of the Cambridge Combinatorial Conference in Honour of Paul Erdős, Academic Press, London New York San Francisco, 1984, 219–232.

[9] B. Korte, L. Lovász and R. Schrader: Greedoids. Springer-Verlag, Berlin Heidelberg, 1991.

[10] G.A. Koshevoy: Choice functions and abstract convex geometries. Mathematical Social Sciences **38**, 1999, 35–44.

[11] J. Oxley: Matroid Theory. Oxford University Press, New York, 1992.

[12] E. Swartz: Topological representations of matroids. Preprint, arXiv:math.CO/0208157, 2002.

Maximum subsets in Euclidean position in Euclidean 2-orbifolds and the sphere

M. Abellanas*, C. Cortés†, G. Hernández*,
A. Márquez† and J. Valenzuela‡

Abstract

Intuitively, a set of sites on a surface is in Euclidean position, if points are so close each other that planar algorithms can be easily adapted in order to solve classical problems of Computational Geometry. In this work we focus in a very relevant class of metric surfaces, the Euclidean 2-orbifolds in addition to the sphere. To seek for maximum sets (in terms of cardinal) in Euclidean position of a given set of sites is, in most of these surfaces, equivalent to compute the maximum depth of an arrangement of convex sets determined by the geometry of the surface. We present algorithms for finding either one or all the maximum subsets as well as the number of such subsets and their minimum cardinal and show that this problem is equivalent to the search of the maximum clique for a natural class of geometric graphs generated from these surfaces.

1 Introduction. Euclidean position

Most people along the human history have believed in a *flat earth*. Even nowadays there exit persons that still hold the flatness of the world. This is because most of our daily experience takes place in a restricted region of a sphere-like surface, so that there are no significant errors if it is considered as a plane. This idea can be easily extended to the Computational Geometry context, where in multiple applications it is assumed that if a given data set is constrained to a small portion of a surface it presents a planar behavior.

The notion of Euclidean position was introduced in [7] for the sphere, the cylinder, the cone, and the torus, and it was extended in [4] to a wider class of surfaces, the Euclidean 2-orbifods, where methods for determining whether or not a point set is in Euclidean position are developed (see first column in Table 1). In this paper we extend those previous works and focus on finding the subsets with highest cardinal in Euclidean position either on an Euclidean 2-orbifold or on the sphere.

As it is known, any Euclidean 2-orbifolds is obtained as a quotient space $\varphi : \mathbb{R}^2 \longrightarrow \mathbb{R}^2/\Gamma \simeq S$, being φ the quotient map and Γ a discrete group of planar motions, and where an equivalent class (the *orbit*) of a point of $\mathbb{R}^2$ is given by all its images by elements of Γ. The four locally Euclidean surfaces (the cylinder, the twisted cylinder, the torus and the Klein bottle) and others well-known surfaces as the Moebius strip or the projective plane are 2-orbifolds. A more complete study of them can be found in [8].

Given two points, X and Y, the geodesics joining them in the quotient metric correspond to straight-line segments matching one point of the orbit of X with all the ones of the orbit of Y. The shortest of these line segments will be called the *segment* between X and Y. On the sphere,

*Dpto. de Matemática Aplicada, Fac. de Informática, Univ. Politécnica de Madrid, {mabellanas,gregorio}@fi.upm.es. Partially supported by projects MCyT TIC2002-04486-C02-01 and CAM 07T/0014/2001

†Dpto. de Matemática Aplicada I, Univ. de Sevilla, {ccortes,almar}@us.es. Partially supported by MCyT project BFM2001-2474

‡Dpto. de Matemáticas, Univ. de Extremadura, jesusv@unex.es. Partially supported by MCyT project BFM2001-2474

the *segment* corresponds to the shortest geodesic joining the points. The distance between X and Y is the length of the segment joining them.

To get a simple representation of an Euclidean 2-orbifold, a very useful tool is the *fundamental domain*: a closed region in the plane containing one element of each orbit, that is unique except for the points of the boundary (*double points*). If we delete all double points of a fundamental domain and consider its image by φ, we obtain what we call a *fundamental region*.

A set of sites P either on a 2-orbifold or the sphere is said to be in *Euclidean position* if there exists a fundamental region so that all segments joining points of P (the clique generated by P) are contained inside it (a hemisphere in the case of the sphere).

A subset Q of P is a *maximum subset of P for the Euclidean position* (MSEP for short) if it is in Euclidean position and there is no other higher cardinal subset having that property. In this work we develop several methods to find either *one* or *all* the MSEPs of a given set of sites both on the Euclidean 2-orbifolds and the sphere and determine how large *all* means. Complexities of these procedures are summarized in Table 1 together with the minimum number of points of P that can be assured to be in Euclidean position on each surface.

Given a set P on an Euclidean 2-orbifold or the sphere, the *segment graph* of P is the geometric graph having the points of its orbit as vertices, and as edges all the possible segments, that coincides with the image by φ^{-1} of the complete graph with nodes in P. It is easy to realize the following result:

Lemma 1 *$Q \subset P$ is in Euclidean position if and only if the connected components of the segment graph of Q are cliques.*

This turns our problem to the searching of cliques in segment graphs, a problem that is known to be NP-hard for general graphs [3, 6]. However, for segment graphs on the sphere and most of 2-orbifolds (the ones without glide reflections) we prove that this can be solved in polynomial time. This is due to the geometric properties of these surfaces which allow to transform this problem to the one of computing the region of maximum depth in an arrangement of convex polygons (or maximum circles in the sphere).

We will make use of known algorithms for computing the maximum depth of an arrangement of convex polygons [5, 2] as well as the ones for determining the halfspace depth of a point in both the plane [1] and the space [9]. This reasoning does not work properly in surfaces with glide reflections, where no polynomial time algorithms have been found at once. In fact, sets with $O(2^n)$ MSEPs can be constructed in these surfaces.

2 Surfaces without glide reflections

It is known a set of sites P on the cone, the cylinder, the torus, or the sphere is in Euclidean position if and only if it is contained in certain sets whose shape depend on the surface considered (between opposite generatrices of the cylinder or the cone; a quadrant -the region between two opposite parallels and two opposite meridians- of the flat torus; or an hemisphere of the sphere). By taking the counter image by the quotient map of these sets in the plane, it can be proven [7, 4] that P is in Euclidean position if and only if its orbit is contained in some particular convex sets (according on the surface considered) in the plane.

Thus, the searching of MSEPs turns to the optimal placement problem of a certain convex set containing the maximum number of points of $\varphi^{-1}(P)$. This can be solved by computing the maximum depth of the arrangement of sets centered at the points of the orbit of P, so the next result holds:

Theorem 1 *Let $S = \mathbb{R}^2/\Gamma$ be an Euclidean 2-orbifold (respectively the sphere), with Γ a discrete group of motions containing no glide reflections, and a set of sites P on S. Then it can be constructed an arrangement of convex polygons (respect. maximum circles) such as any region of the arrangement with maximum depth is associated to a MSEP of P.*

The region with maximum depth in an arrangement of n convex sets can be computed in $O(nk \log n)$ time, where k is the depth of the arrangement, for translations of a fixed convex set in the plane [2]; or, as it is noted in [5], by modifying the algorithm given in [10], that provides a computational time of $O(n^{d/2} \log n)$ for an arrangement of isotetic hipercubes in $\mathbb{R}^d$. These algorithms together with Theorem 1 give rise to the following assertion:

Corolary 1 *A maximal clique in a segment graph generated from a set in a 2-orbifold without glide reflection or the sphere can be found in polynomial time.*

The convex polygons cited in Theorem 1 depend on the 2-orbifold considered, and it is

- a strip for sets given on the cylinder,
- an angular sector with vertex on the center of the rotation for the cone,
- an isotetic rectangle for the flat torus (generated from two orthogonal translations) and the pillow-like surfaces (generated from rotations),
- a hexagon for the skew torus (non-orthogonal translations), or
- an isotetic cube for the Pillow (generated by two π radians rotations and a translation or by four π radians rotations).

The time needed to compute the maximum depth of arrangements of such sets are summarized in second column of Table 1. The optimum for the cylinder and the cone are due to the MSEP search in these surfaces can be connected to the halspace depth problem; that is, to determine the halfspace, whose boundary contains a fixed point (the center of the circumference in our case), having less points of a given set [1]. This also assures the optimum for the flat torus and the pillow-like surfaces.

Note as the angle between the generating translations in the skew torus cause a change of the polygon considered that increases the computing time. This $O(n^{1.5} \log n)$ time improves the $O(nk \log n)$ time given in [2] for arrangement whose depth k is greater than $\sqrt{n}$.

In spite of what happen on the 2-orbifolds, the searching of a MSEP on the sphere can be done by computing the maximum depth region of an arrangement of maximum circles on the sphere itself. This problem is equivalent either to determine the subset of highest cardinal of a point set in $\mathbb{R}^3$ whose convex hull does not contain a fixed point (the center of the sphere) or to compute the halfspace depth of the center, and improves the $O(n^2 \log n)$ time given in [9].

Theorem 2 *Given a set of n sites on $\mathbb{R}^3$, the halspace depth of a given point can be computed in $O(n^2)$ time.*

Finally, in Table 1 are also listed the number of MSEPs on each surface and the time necessary to find all of them, together with their minimum cardinal.

3 Surfaces with glide reflections

If a glide reflection is involved, there is not a unique region G such as a set is in Euclidean position if and only if it is contained in G. 2-orbifolds generated by this motion are non-orientables, and it includes the Moebius strip, the Klein bottle or the projective plane. In this surfaces it is not possible to make use of Theorem 1. If fact, opposite the other 2-orbifolds, there can be constructed sets with $O(2^n)$ MSEPs, and $\Theta(2^n)$ time is required to report all of them. We are actually working in determining if it is possible to find *one* MSEPs (and as a consequence, a clique in the segment graph) in polynomial time.

4 Conclusions and open problems

Problem we have worked on are summarized in Table 1; computing either one or all MSEPs in Euclidean position and giving bounds for the number of such sets and for the minimum number of points that take part of any of them.

	Determine [4]	Find a max. set	Num. of max. sets	Find all max. sets	Min. num. of points
Cylinder	$\Theta(n)$	$\Theta(n \log n)$	$O(n)$	$\Theta(n \log n)$	$n/2$
Cone	$\Theta(n)$	$\Theta(n \log n)$	$O(n)$	$\Theta(n \log n)$	$n/2$
Torus	$\Theta(n)$	$\Theta(n \log n)$	$O(n^2)$	$\Theta(n^2)$	$n/4$
Skew Torus	$\Theta(n)$	$O(n^{1.5} \log n)$	$O(n^2)$	$\Theta(n^2)$	$n/4$
Pillow	$\Theta(n)$	$O(n^{1.5} \log n)$	$O(n^3)$	$\Theta(n^3)$	$n/2$
Pillow-like surfaces	$\Theta(n)$	$\Theta(n \log n)$	$O(n^2)$	$\Theta(n^2)$	$n/2$
Surfaces with glide reflections	$O(n \log n)$	?	$O(2^n)$	$\Theta(2^n)$	$n/4$
Sphere	$\Theta(n)$	$O(n^2)$	$O(n^2)$	$\Theta(n^2)$	$n/2$

Table 1: Scheme of problems and the current cost of solutions.

References

[1] G. Aloupis, C. Cortés, F. Gómez, M. Soss, and G. Toussaint. Lower bounds for computing statistical depth. *Computational Statistics and Data Analysis*, 40:223–229, 2002.

[2] G. Barequet, M. Dickerson, and P. Pau. Translating a convex polygon to containn a maximum number of points. *Computational Geometry: Theory and Applications*, 8:167–179, 1997.

[3] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization.*, Supplement volume A, pages 1–74. Kluwer Academic Publisher, 1999.

[4] C. Cortés, A. Márquez, and J. Valenzuela. Euclidean position in 2-orbifolds. In *18th European Workshop in Computational Geometry*, April 2002.

[5] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Computational Geometry*, 11:321–350, 1994.

[6] M. Garey and D. Johnson. *Computers and Intractability.* W. H. Freeman, 1979.

[7] C. I. Grima and A. Márquez. *Computational Geometry on Surfaces.* Kluwer Academic Publisher, 2001.

[8] V. V. Nikulin and I. R. Shafarevich. *Geometries and Groups.* Springer Series in Soviet Mathematics. Springer, Berlin, 1987.

[9] P. J. Rousseeuw and A. Struyf. Computing location depth and regression depth in higher dimensions. *Statistics and Computing*, 8:193-203, 1998.

[10] M. H. Overmars and C. Yap. New upper bounds in Klee's measure problem. *SIAM J. Computing*, 20:1034–1045, 1991.

Optimal Pants Decompositions and Shortest Freely Homotopic Loops on an Orientable Surface

Éric Colin de Verdière* Francis Lazarus†

1 Introduction

Let $\mathcal{M}$ be a compact orientable combinatorial surface of genus g with b boundaries. A *pants decomposition* of $\mathcal{M}$ is a maximal set of pairwise disjoint, non-isotopic, essential loops on $\mathcal{M}$; a loop being *essential* if it is simple and neither contractible nor homotopic to a boundary of $\mathcal{M}$. A pants decomposition is made of $3g-3+b$ loops and cuts $\mathcal{M}$ into *pairs of pants*, *i.e.*, spheres with three boundaries (see [4]).

We describe a conceptually simple, polynomial, iterative scheme which takes a given pants decomposition and outputs a shorter homotopic pants decomposition. We prove that, at the end of the process, each loop is a shortest loop in its homotopy class (in this paper, we consider homotopy of loops without basepoint, *i.e.*, free homotopy). In particular, the resulting decomposition is *optimal* in the sense that it is as short as possible among all homotopic decompositions.

Furthermore, given a simple, essential loop ℓ, it is not difficult to extend ℓ to a pants decomposition of $\mathcal{M}$ (see [3]). This decomposition, after optimization, contains a shortest loop homotopic to ℓ which is simple. Even the existence of such a simple loop is non-obvious.

The problem of optimizing a pants decomposition was raised in the conclusion of [3]; to our knowledge, we present the first algorithm which solves it. It also somehow extends [5] to more general surfaces. This is a natural extension of our former paper [1] where we treat the case of optimal simple loops in a given class of homotopy with fixed basepoint as opposed to free homotopy.

*Laboratoire d'informatique de l'École normale supérieure, UMR 8548 (CNRS), Paris, France. Eric.Colin.de.Verdiere@ens.fr

†Laboratoire IRCOM-SIC, UMR 6615 (CNRS), Poitiers, France. Francis.Lazarus@sic.univ-poitiers.fr

2 Framework and Result

The framework we use in this paper is very close to the one used in [1], see this paper for details. The surface $\mathcal{M}$ is assumed to be a polyhedral 2-manifold, whose edges have positive weights. Let G be the vertex-edge graph of $\mathcal{M}$, and G^* be its dual (embedded into $\mathcal{M}$). We consider sets of disjoint, simple, piecewise linear (PL) curves drawn on $\mathcal{M}$ that intersect G^* in an *admissible* way (*i.e.*, the intersections are generic). Throughout this paper, we always assume admissibility. If a curve crosses the edges $e_1^*, \ldots, e_k^*$ of G^*, then its *length* is defined to be the sum of the weights of $e_1, \ldots, e_k$. This notion of length coincides with the usual length if we retract all curves on G.

Let s^{init} be a pants decomposition of $\mathcal{M}$ to be optimized[1]. To simplify the computation and the proof of correctness, we first augment s^{init} to form a *doubled pants decomposition*, which we call s. Is is obtained by taking a copy of each loop in s^{init} and of each boundary[2] of $\mathcal{M}$, slightly translated, in the same homotopy class, such that s is still a set of pairwise disjoint simple loops. $s = (s_1, \ldots, s_N)$ is thus composed of $N = 6g-6+3b$ loops. A loop of s or a boundary of $\mathcal{M}$ and its translated copy are called *twins*. For a loop s_j in s, the connected component of $\mathcal{M} \setminus \{s \setminus s_j\}$ that contains s_j is a pair of pants, and one of its three boundaries is the twin of s_j. We note $\mathcal{P}_j$ this pair of pants.

Definition 1 *An* Elementary Step $f_j(s)$ *consists in replacing the jth loop s_j by a shortest simple homotopic loop in $\mathcal{P}_j$. A* Main Step $f(s)$ *is the application of $f = f_N \circ f_{N-1} \circ \ldots \circ f_2 \circ f_1$ to*

[1] In fact, we allow s^{init} to be a decomposition of $\mathcal{M}$ with pairs of pants and annuli, as it must be the case if $\mathcal{M}$ is a torus or a cylinder. This technicality does not change anything for the rest of the paper.

[2] This to allow shortening of loops that would be homotopic to a boundary.

s. These operations transform a doubled pants decomposition into another one, keeping the homotopy class of the decomposition.

Here is our main theorem:

Theorem 2 *Let s^0 be a doubled pants decomposition of $\mathcal{M}$, and let $s^{n+1} = f(s^n)$. For some $m \in \mathbb{N}$, s^m and s^{m+1} have the same length and, in this situation, s^m is a doubled pants decomposition homotopic to s^0 made of loops which are individually as short as possible among all loops in their (free) homotopy class. In particular, s^m is an optimal doubled pants decomposition of $\mathcal{M}$.*[3]

Since it is easy to extend a simple loop to a pants decomposition, and since a pants decomposition is made of simple loops, we have:

Corollary 3 *Let ℓ be a simple loop in $\mathcal{M}$. There exists a* simple *loop ℓ' homotopic to ℓ which is as short as possible among all loops homotopic to ℓ.*

The following section aims at proving Theorem 2. Note that Lemma 7 explains how to perform algorithmically the computations of f_i.

3 Proof of Theorem 2

Let π be the projection from the universal cover $\tilde{\mathcal{M}}$ of $\mathcal{M}$ onto $\mathcal{M}$. In this section, we fix $i \in [1, N]$; let s be a doubled pants decomposition, and let t_i be a loop which is homotopic to s_i and as short as possible among all loops homotopic to s_i. We may assume that no lift of t_i self-intersects in $\tilde{\mathcal{M}}$ (see below for the definition of a lift). This fact, which is not trivial, will be used in the proof of Proposition 9. Our goal is to prove that, after a finite number of steps, the ith loop of the doubled pants decomposition has the same length as t_i.

3.1 Lifts and translations in $\tilde{\mathcal{M}}$

Let ℓ be a loop on $\mathcal{M}$. We view ℓ as a 1-periodic mapping from $\mathbb{R}$ into $\mathcal{M}$. A *lift* of ℓ is a mapping $\tilde{\ell} : \mathbb{R} \to \tilde{\mathcal{M}}$ such that $\pi \circ \tilde{\ell} = \ell$. A *part* of a lift $\tilde{\ell}$ is the restriction of $\tilde{\ell}$ to an interval of the form $[a, a+1)$.

Let $\tilde{\ell}_1$ be a part of a lift $\tilde{\ell}$. Let v be a point in $\tilde{\mathcal{M}}$, and let β be a path from the source of $\tilde{\ell}_1$ to v. Consider the target v' of the lift of $\pi(\beta)$ starting at the target of $\tilde{\ell}_1$. It is readily seen that v' does not depend on the path β, nor on the part $\tilde{\ell}_1$ of $\tilde{\ell}$ chosen. We have $\pi(v) = \pi(v')$; intuitively, v' is the translated of v by $\tilde{\ell}$. We define $\tau_{\tilde{\ell}}(v)$ to be v'.

Let s_j and $s_{j'}$ be two twins of s. Let $(s_j^\alpha)^{\alpha \in \mathbb{N}}$ be an enumeration of the lifts of s_j in $\tilde{\mathcal{M}}$. By [2, Lemma 2.4], s_j and $s_{j'}$ bound a cylinder in $\mathcal{M}$. It follows that, for each $\alpha \in \mathbb{N}$, s_j^α is one boundary of an infinite strip which contains no lift of s in its interior, and is bounded from the other side by a lift of $s_{j'}$. We call $s_{j'}^\alpha$ this other boundary.

Let $j \in [1, N]$. For a lift $\tilde{t}_i$ of t_i, $\tau_{\tilde{t}_i}$ induces a permutation σ_j of $\mathbb{N}$ as follows: the image by $\tau_{\tilde{t}_i}$ of s_j^α is also a lift of s_j, which we call $s_j^{\sigma_j(\alpha)}$. The σ_j's, which depend on $\tilde{t}_i$ and on the enumeration of the lifts of s_j, will remain fixed in the rest of this paper.

3.2 Crossing words

Let A be the set of *symbols* of the form k^α or $\bar{k}^\alpha$, where $k \in [1, N]$ and $\alpha \in \mathbb{N}$. The set A^* of *words* on A is the set of finite sequences of elements in A. Let $\tilde{p}$ be a path in $\tilde{\mathcal{M}}$; $\tilde{p}$ crosses the lifts s_k^α (for $k \in [1, N]$ and $\alpha \in \mathbb{N}$) at a finite number of points. We walk along $\tilde{p}$ and, at each crossing encountered with a lift s_k^α of s, we write the symbol k^α or $\bar{k}^\alpha$, according to the orientation of the crossing (with respect to a fixed orientation of $\tilde{\mathcal{M}}$). The resulting element of A^* is called the *crossing word* of $\tilde{p}$ with s, and denoted by $s/\tilde{p}$.

Lemma 4 *Let $a^1 < a^2$ be two real numbers such that exactly one crossing occurs between all lifts of s and $\tilde{t}_i|_{[a^1, a^2)}$. For $k = 1, 2$, let $w^k = s/\tilde{t}_i|_{[a^k, a^k+1)}$.*

If $w^1 = j^\alpha.w$ (resp. $w_1 = \bar{\jmath}^\alpha.w$), then $w^2 = w.j^{\sigma_j(\alpha)}$ (resp. $w^2 = w.\bar{\jmath}^{\sigma_j(\alpha)}$).

Let $w \in A^*$. We define the relation $\sim$ to be the equivalence relation generated by $j^\alpha.w \sim w.j^{\sigma_j(\alpha)}$ and $\bar{\jmath}^\alpha.w \sim w.\bar{\jmath}^{\sigma_j(\alpha)}$ (for any j and α). Let $[A^*]$ be A^* quotiented by the relation $\sim$. If $w \in A^*$, we denote by $[w]$ its equivalence class in $[A^*]$.

[3] **Remark.** The proof of Theorem 2 extends to the case where we consider the real *length* of PL loops drawn on $\mathcal{M}$ (and not on its vertex-edge graph), provided that the suitable definition of a crossing is used: we have to take into account that two loops can partly overlap without crossing.

Let $\tilde{t}_i^1$ be a part of $\tilde{t}_i$. It follows from the previous lemma that $[s/\tilde{t}_i^1]$ does not depend on $\tilde{t}_i^1$; hence we define $[s/\tilde{t}_i]$ to be their common equivalence class in $[A^*]$.

Let $j \in [1, N]$, and let $[w] \in [A^*]$. The *j-reductions* of $[w]$ are defined as follows. If w has the form $w_1 j^\alpha \bar{\jmath}^\alpha w_2$ or $w_1 \bar{\jmath}^\alpha j^\alpha w_2$, then we say that $[w]$ *j-reduces* to $[w_1 w_2]$; if w has the form $j^\alpha w_1 \bar{\jmath}^{\sigma_j(\alpha)}$ or $\bar{\jmath}^\alpha w_1 j^{\sigma_j(\alpha)}$, then we also say that $[w]$ *j-reduces* to $[w_1]$. Obviously, this definition does not depend on the particular choice of the word w in $[w]$.

A *reduction* is a j-reduction for some j. $[w]$ is *j-irreducible* (resp. *irreducible*) if it can be applied no j-reduction (resp. reduction).

Lemma and Definition 5 *Let $[w] \in [A^*]$. There is only one j-irreducible (resp. irreducible) element of $[A^*]$ which can be obtained from $[w]$ by successive j-reductions (resp. reductions). We define $g_j([w])$ (resp. $g([w])$ to be this word.*

3.3 Reducibility of $[s/\tilde{t}_i]$

Proposition 6 *$g([s/\tilde{t}_i]) = \varepsilon$, where ε is the class of the empty word in $[A^*]$.*

PROOF. Let s_i' be a loop homotopic to s_i and slightly translated such that it does not cross any loop s_k. Let $\dot{s}_i$ and $\dot{t}_i$ be the restrictions of s_i' and t_i to $[0,1]$. There exists a path β joining $s_i'(0)$ to $t_i(0)$ such that the path $p := \beta.\dot{t}_i.\beta^{-1}.\dot{s}_i^{-1}$ is a null-homotopic loop in $\mathcal{M}$. We subdivide p into four paths $p_1 = \beta$, $p_2 = \dot{t}_i$, $p_3 = \beta^{-1}$, and $p_4 = \dot{s}_i^{-1}$. Let $\tilde{p} = \tilde{p}_1.\tilde{p}_2.\tilde{p}_3.\tilde{p}_4$ be a lift of p such that $\tilde{p}_2$ is on $\tilde{t}_i$.

It can be proved that $s/\tilde{p}$ is a parenthesized expression (it reduces to the empty word by successive removals of subwords of the form $j^\alpha \bar{\jmath}^\alpha$ and $\bar{\jmath}^\alpha j^\alpha$). Hence $g([s/\tilde{p}]) = \varepsilon$.

$\tilde{p}_1$ and $\tilde{p}_3^{-1}$ are parts of lifts of β, and $\tau_{\tilde{t}_i}(\tilde{p}_1)$ is equal to $\tilde{p}_3^{-1}$. Hence, if the kth symbol of $s/\tilde{p}_1$ is equal to j^α (resp. $\bar{\jmath}^\alpha$), then the kth symbol of $s/\tilde{p}_3^{-1}$ (which equals $s/\tilde{p}_3$ in reverse order) is $j^{\sigma_j(\alpha)}$ (resp. $\bar{\jmath}^{\sigma_j(\alpha)}$). Since $s/\tilde{p}_4$ is empty, it follows that $g([s/\tilde{p}_2]) = g([s/\tilde{p}])$. The left handside equals $g([s/\tilde{t}_i])$ and the right handside equals ε. □

3.4 Uncrossing the loops

Lemma 7 *Let $r = f_j(s)$. r_j is, in $\mathcal{P}_j$, a shortest loop homotopic to s_j.*

PROOF (SKETCH). Let b_k, $k = 1, 2, 3$, be the boundaries of $\mathcal{P}_j$, such that b_1 is homotopic to s_j. Let p_1 (resp. p_2) be a shortest path between b_2 and b_3 (resp. b_1 and b_3); we can make these paths simple and disjoint. Let ℓ be a shortest loop homotopic to s_j in $\mathcal{P}_j$, and $\tilde{\ell}$ be a lift of ℓ in the universal cover of $\mathcal{P}_j$. By analyzing the way $\tilde{\ell}$ crosses the lifts of p_1 and p_2, we can prove that we can change ℓ to a loop ℓ', which is also a shortest loop homotopic to s_j in $\mathcal{P}_j$, but does not cross p_1 and crosses p_2 once.

Cut $\mathcal{P}_j$ along p_1 and p_2; for each pair of vertices corresponding to a single vertex of p_2 before cutting, compute a shortest path whose endpoints are this pair of vertices; take the shortest of these shortest paths. By the preceding paragraph, this path yields a shortest loop homotopic to s_j in $\mathcal{P}_j$, and it is simple. As a byproduct, this describes a way to compute $f_j(s)$. □

Let $r = f_j(s)$. If $k \neq j$, let r_k^α be equal to s_k^α. To get an enumeration of the lifts of r_j, we proceed as follows. Let $s_{j'}$ be the twin of s_j. Note that r_j and $s_{j'}$ bound a cylinder by [2, Lemma 2.4]. We let r_j^α to be the lift of r_j which bounds the lift of this cylinder whose other boundary is $s_{j'}^\alpha$. It follows that $\tau_{\tilde{t}_i}(r_j^\alpha)$ is equal to $r_j^{\sigma_j(\alpha)}$ (in other words, the permutation σ_j remains unchanged).

Lemma 8 *$g_j([r/\tilde{t}_i]) = g_j([s/\tilde{t}_i])$.*

PROOF (SKETCH). Let $[r/\tilde{t}_i]_j$ and $[s/\tilde{t}_i]_j$ be obtained by deleting j-symbols from $[r/\tilde{t}_i]$ and $[s/\tilde{t}_i]$. Since r and s only differ in their jth loop, these two words are identical. Consider two consecutive symbols σ_1 and σ_2 in $[u/\tilde{t}_i]_j$, where u stands for either r or s. These two symbols are replaced in $[u/\tilde{t}_i]$ by an expression $\sigma_1 w_j \sigma_2$, where w_j is a word on j-symbols. We only need to show that w_j reduces (with parenthesized reductions) to a same expression for $u = r$ and $u = s$. This obviously implies the lemma. The proof uses the fact that $u_{j'}$ $(= s_{j'})$ and u_j bound a cylinder in $\mathcal{P}_j$, and this cylinder is crossed by no other loops of u. □

Proposition 9 *We can replace t_i by a loop t_i' (homotopic to t_i, no longer than t_i, and such that its lifts are simple) so that $[r/\tilde{t}_i'] = g_j([s/\tilde{t}_i])$ for some lift $\tilde{t}_i'$ of t_i'.*

PROOF. By Lemma 8, we may only consider the case where $[r/\tilde{t}_i']$ is j-reducible; this implies

that there is a disk D in $\tilde{\mathcal{M}}$ bounded by an arc $\tilde{r}_j^{ab}$ of a lift $\tilde{r}_j$ of r_j, and an arc $\tilde{t}_i^{ab}$ of $\tilde{t}_i$ with the same endpoints a and b.

D intersects $\tilde{t}_i$ in a set of pairwise disjoint arcs with endpoints on $\tilde{r}_j^{ab}$ (recall $\tilde{t}_i$ is simple). Consider an innermost such arc $\tilde{t}_i^{cd}$, *i.e.*, such that it sustains a subarc $\tilde{r}_j^{cd}$ of $\tilde{r}_j^{ab}$ that does not intersect $\tilde{t}_i$.

If $\tilde{t}_i^{cd}$ were shorter than $\tilde{r}_j^{cd}$, we could shorten r_j as follows: in $\tilde{\mathcal{M}}$, replace the part $\tilde{r}_j^{cd}$ of $\tilde{r}_j$ by a path with the same endpoints going along $\tilde{t}_i^{cd}$, and project it onto $\mathcal{M}$. The resulting loop, r'_j, is shorter than r_j; moreover, no lift of any loop other than t_i can cross D, so the projection $\pi(D)$ lies entirely in $\mathcal{P}_j$. It follows that r'_j is homotopic *in* $\mathcal{P}_j$ to r_j, while being shorter; this contradicts Lemma 7.

We modify $\tilde{t}_i$ as follows: replace the part $\tilde{t}_i^{cd}$ of $\tilde{t}_i$ by a path with the same endpoints going along $\tilde{r}_j^{cd}$, on the other side of $\tilde{r}_j^{cd}$ (to remove the two crossings). The projection t'_i of the resulting path is a loop homotopic to t_i, and no lift of this new loop self-intersects in $\tilde{\mathcal{M}}$. It cannot be longer than t_i by the preceding paragraph, hence t'_i is a shortest loop homotopic to s_i whose lifts are simple. Moreover, $[r/\tilde{t}'_i]$ is deduced from $[r/\tilde{t}_i]$ by a j-reduction. We finish the proof by induction. □

3.5 Conclusion of the proof

Lemma 10 *Assume t_i does not cross any loop of s; let $\mathcal{P}$ be the pair of pants delimited by s in which t_i is. Then one of the boundaries of $\mathcal{P}$ is homotopic,* in $\mathcal{P}$, *to t_i.*

PROOF. Omitted in this abstract. □

Lemma 11 *Assume that $[s/\tilde{t}_i] = \varepsilon$. Let $r = f^2(s)$. Then r_i and its twin have the same length as t_i.*

PROOF. By Lemma 10, t_i is inside a pair of pants bounded by some s_k which is either s_i or its twin. By Proposition 9, we may replace t_i by t'_i such that $s' := f_{k-1} \circ \ldots \circ f_1(s)$ does not cross t'_i, and (in fact) that t'_i is in a pair of pants bounded by s'_k. Hence, by Lemma 7, the kth loop of $f_k(s')$ has the same length as t_i. After one more iteration of f the same is true for the twin of the kth loop. □

PROOF OF THEOREM 2. Fix i; let t_i^0 be a shortest loop homotopic to s_i^0. By Propositions 9 and 6, one can construct a sequence $(t_i^n)_{n\in\mathbb{N}}$ of shortest homotopic loops such that the length of $[s^n/t_i^n]$ strictly decreases. Then for some n, $[s^n/t_i^n] = \varepsilon$. By Lemma 11, s_i^{n+2} has the same length as t_i^0. Hence the length of s^n becomes stationary. It remains to prove that the lengths remain unchanged once s^n and s^{n+1} have the same lengths. □

4 Complexity

We present a sketch of the complexity analysis (which is similar to and simpler than the one in [1]). Let n be the number of edges of $\mathcal{M}$, g its genus and b its number of boundaries. Let α be the longest-to-shortest edge ratio of $\mathcal{M}$. Let S be a combinatorial doubled pants decomposition of $\mathcal{M}$ composed of $N = O(g+b)$ loops, and μ be the maximal multiplicity of any vertex of $\mathcal{M}$ in a loop of S. Hence the number of edges of a loop at the beginning of the algorithm is $O(\mu n)$, and, since loops can only get shorter in length, their maximal number of edges is $O(\alpha\mu n)$. We can prove that the lengths of the crossing words is $O((g+b)\alpha\mu^2 n)$, and compute the time spent by an Elementary Step, using Dijkstra's algorithm and the proof of Lemma 7. Finally:

Theorem 12 *This algorithm computes an optimal pants decomposition homotopic to S in $O(\mu^4\alpha^3(g+b)^2n^3\log\mu\alpha n)$ time.*

References

[1] É. Colin de Verdière and F. Lazarus. Optimal system of loops on an orientable surface. In *IEEE Symp. Found. Comput. Sci.*, pages 627–636, 2002.

[2] D. Epstein. Curves on 2-manifolds and isotopies. *Acta Mathematica*, 115:83–107, 1966.

[3] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. In *Proc. 18th Annu. ACM Symp. Comput. Geom.*, pages 244–253, 2002.

[4] A. Hatcher. Pants decompositions of surfaces. http://www.math.cornell.edu/~hatcher/Papers/pantsdecomp.pdf, 2000.

[5] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.

Geometric Games on Triangulations
Extended Abstract

Oswin Aichholzer[1] David Bremner[2] Erik D. Demaine[3]
Ferran Hurtado[4] Evangelos Kranakis[5] Hannes Krasser[6]
Suneeta Ramaswami[7] Saurabh Sethia[8] Jorge Urrutia[9]

1 Introduction

Let S be a set of n points in the plane, which we assume to be in general position, i.e., no three points of S lie on the same line. A *triangulation* of S is a simplicial decomposition of its convex hull having S as vertex set.

In this work we consider several perfect-information combinatorial games involving the vertices, edges (straight-line segments) and faces (triangles) of some triangulation. We describe main broad categories of these games and provide in various situations polynomial-time algorithms to determine who wins a given game under optimal play, and ideally, to find a winning strategy.

We present games where two players $\mathcal{R}$(ed) and $\mathcal{B}$(lue) play in turns, as well as *solitaire* games for one player. In some *bichromatic* versions, player $\mathcal{R}$ will use red and player $\mathcal{B}$ will use blue, respectively, to color some element of the triangulation. In *monochromatic* variations, all players (maybe the single one) use the same color, green.

Games on triangulations come in three main flavors:

- *Constructing (a triangulation).* The players construct a triangulation $T(S)$ on a given point set S. Starting from no edges, players $\mathcal{R}$ and $\mathcal{B}$ play in turn by drawing one or more edges in each round. In some variations, the game stops as soon as some structure is achieved. In other cases, the game stops when the triangulation is complete, the last move or possibly some counting decides then who is the winner.

- *Transforming (a triangulation).* A triangulation $T(S)$ on top of S is initially given, all edges originally being black. In each turn, a player applies some local transformation to the current triangulation, resulting in a new triangulation. The game stops when a specific configuration is achieved or no more moves are possible.

- *Marking (a triangulation).* A triangulation $T(S)$ on top of S is initially given, all edges and nodes originally being black. In each turn, some of its elements are marked (e.g. colored) in a game-specific way. The game stops when some configuration of marked elements is achieved (possibly the whole triangulation) or no more moves are possible.

For each of the variety of games described in Section 2, we are interested in characterizing who wins the game, and designing efficient algorithms to determine the winner and compute a winning strategy. More details about the games can be found in the full papers [1] and [2].

1 Institute for Softwaretechnology, Graz University of Technology; 2 Faculty of Computer Science, University of New Brunswick; 3 Laboratory for Computer Science, Massachusetts Institute of Technology; 4 Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya; 5 School of Computer Science, Carleton University; 6 Institute for Theoretical Computer Science, Graz University of Technology; 7 Computer Science Department, Rutgers University; 8 Department of Computer Science, Oregon State University; 9 Instituto de Matemáticas, Universidad Nacional Autónoma de México.

2 Examples of Games

We describe next the rules of several specific games that we have studied. Our intention here is to make clear which kind of games we are dealing with.

2.1 Constructing

2.1.1 Monochromatic Complete Triangulation. The players construct a triangulation $T(S)$ on a given point set S. Starting from no edges, players $\mathcal{R}$ and $\mathcal{B}$ play in turn by drawing one edge in each round. Each time a player completes one or more empty triangle(s), it is (they are) given to this player and it is again her turn (an "extra move"). Once the triangulation is complete, the game stops and the player who owns more triangles is the winner.
2.1.2 Monochromatic Triangle. Starts as in 2.1.1, but has a different stopping condition: the first player who completes one empty triangle is the winner.
2.1.3 Bichromatic Complete Triangulation. As in 2.1.1, but the two players use red and blue edges. Only monochromatic triangles count.
2.1.4 Bichromatic Triangle. As in 2.1.2, but with red and blue edges. The first empty triangle must be monochromatic.

2.2 Transforming

2.2.1 Monochromatic Flipping. Two players start with a triangulation whose edges are initially black. Each move consists of choosing a black edge, flipping it, and coloring the new edge green. The winner is determined by normal play, meaning that the goal is to make the last complete move.
2.2.2 Monochromatic Flipping to Triangle. Same rules as for 2.2.1, except now the winner is who completes the first empty green triangle.
2.2.3 Bichromatic Flipping. Two players play in turn, selecting a flippable black edge e of $T(S)$ and flipping it. Then e as well as any still-black boundary edges of the enclosing quadrilateral become red if it was player $\mathcal{R}$'s turn, and blue if it was player $\mathcal{B}$'s move. The game stops if no more flips are possible. The player who owns more edges of her color wins.
2.2.4 All-Green Solitaire. In each move, the player flips a flippable black edge e of $T(S)$; then e becomes green, as do the four boundary edges of the enclosing quadrilateral. The goal of the game is to color all edges green.
2.2.5 Green-Wins Solitaire. As in 2.2.4, but the goal of the game is to obtain more green edges than black edges.

2.3 Marking

2.3.1 Triangulation Coloring Game. Two players move in turn by coloring a black edge of $T(S)$ green. The first player who completes an empty green triangle wins.
2.3.2 Bichromatic Coloring Game. Two players $\mathcal{R}$ and $\mathcal{B}$ move in turn by coloring red respectively blue a black edge of $T(S)$. The first player who completes an empty monochromatic triangle wins.
2.3.3 Four-Cycle Game. Same as 2.3.1 but the goal is to get an empty quadrilateral.
2.3.4 Nimstring Game. Nimstring is a game defined in Winning Ways [4] as a special case of the classic children's (but nonetheless deep) combinatorial game Dots and Boxes [3, 4]. In the context of triangulations, players in Nimstring alternate *marking* one-by-one the edges of a given triangulation (i.e., coloring green an edge, initially black), and whenever a triangle has all three of its edges marked, the completing player is awarded an extra move and must move again. The winner is determined by normal play. Thus, the player marking the last edge of the triangulation actually loses, because that last edge completes one or two triangles, and the player is forced to move again, which is impossible.

Besides beauty and entertainment, games keep attracting the interest of mathematicians and computer scientists because they also have applications to modeling several areas and because they often reveal deep mathematical properties of the underlying structures, in our case the combinatorics of planar triangulations.

Games on triangulations belong to the more general area of *combinatorial games* which typically involve two players, $\mathcal{R}$(ed) and $\mathcal{B}$(lue). A *game position* consists of a set of options for Red's moves and a set of options for Blue's moves, where each option is itself a game, representing the game position resulting from the move. We define next a few more terms from combinatorial game theory that we will use in this paper. For more information, refer to the books [4, 5] and the survey [6]. The paper [7] contains a list of more than 900 references.

We consider games with *perfect information* (no hidden information as in many card games) and there are no chance moves (like rolling dice). Most of the games we consider (the monochromatic games) are also *impartial* in the sense that the options for Red are the same as the options for Blue. In this case, a game is simply a set of games, and can thus be viewed as a tree. The leaves of this tree correspond to the empty-set game, meaning that no options can be played; this game is called the *zero game*, denoted 0.

In general, each leaf game might be assigned a label of whether the current player reaching that node is a winner or loser, or the players tied. However, a common and natural assumption is that the zero game is a losing position, because the next player to move has no move to make. We usually make this assumption, called *normal play*, so that the goal is to make the last move. In contrast, *misère play* is just the opposite: the last player able to move loses. In more complicated games, the winner is determined by comparing scores.

Any impartial perfect-information combinatorial game without ties has one of two outcomes under optimal play (when the players do their best to win): a *first-player win* or a *second-player win*. In other words, whoever moves first can force herself to reach a winning leaf, or else whoever moves second can force herself to reach a winning leaf, no matter how the other player moves throughout the game. Such forcing procedures are called *winning strategies*. For example, under normal play, the game 0 is a second-player win, and the game having a single move to 0 is a first-player win, in both cases no matter how the players move. More generally, impartial games may have a third outcome: that one player can force a tie.

The Sprague-Grundy theory of impartial games (see e.g. [4], Chapter 3) says that, under normal play, every impartial perfect-information combinatorial game is equivalent to the classic game of Nim. In (single-pile) Nim, there is a pile of $i \geq 0$ beans, denoted $*i$, and players alternate removing any positive number of beans from the pile. Only the empty pile $*0$ results in a second-player win (because the first player has no move); for any other pile, the first player can force a win by removing all the beans. If a game is equivalent to $*i$, then i is called the *Nim value* of the game.

3 Overview of Results

In this section we briefly summarize some of our results from the papers [1] and [2], where all proofs and details can be found.

Theorem 1. *Deciding whether the Triangulation Coloring Game on a simple-branching triangulation (no two inner triangles share a common diagonal) on n points in convex position is a first-player win or a second-player win, as well as finding moves leading to an optimal strategy, can be solved in time linear in the size of the triangulation.*

Theorem 2. *The Monochromatic Triangle Game on n points in convex position is an incarnation of a known game called* Dawson's Kayles *[4]. It is thus a second-player win when $n \equiv 5, 9, 21, 25, 29 \pmod{34}$ and for the special cases $n = 15$ and $n = 35$; otherwise it is a first-player win. Each move in a winning strategy can be computed in time linear in the size of the triangulation.*

Theorem 3. *The outcome of the Monochromatic Complete Triangulation Game on n points in convex position is a first-player win for n odd, and a tie for n even.*

Theorem 4. *Whether a player can win the All-Green Solitaire Game for a given triangulation of n points in convex position can be decided in time $O(n)$. When the player can win, a winning sequence of moves can be found within the same time bound.*

Theorem 5. *The player of the Green-Wins Solitaire Game can obtain from any given triangulation on n points at least $1/6$ of the edges to be green at the end of the game. There are triangulated point sets such that no sequence of flips of black edges provides more than $5/9$ of the edges to be green at the end. (In the above fractions we don't pay attention to additive constants).*

Theorem 6. *The player of the Green-Wins Solitaire Game can always win for any given triangulation on $n \geq 4$ points in convex position.*

Theorem 7. *Nimstring in a fan with an even number of vertices is a first-player win.*

Theorem 8. *Nimstring in a wheel with an odd number of vertices is a second-player win.*

Theorem 9. *Four-Cycle in a triangulation whose dual is a path is a first-player win.*

Theorem 10. *Four-Cycle in a wheel with more than four triangles is a second-player win.*

Theorem 11. *Monochromatic Flipping on top of n points in convex position is a first-player win if n is even and a second-player win if n is odd.*

Theorem 12. *There is a constant N such that Monochromatic Flipping to Triangle in a triangulation (whose dual is a path) of $n \geq N$ points in convex position is a first-player win for n even, and a second-player win for n odd.*

Acknowledgments

David Bremner is supported by the AvH Foundation and NSERC Canada. Ferran Hurtado is partially supported by Projects MEC-DGES-SEUID PB98-0933, MCYT-FEDER BFM2002-0557, Gen. Cat 2001SGR00224 and Accion Integrada España-Austria HU2002-0010. Research of Evangelos Kranakis is supported in part by NSERC and MITACS grants. Research of Hannes Krasser is supported by the FWF (Austrian Fonds zur Förderung der Wissenschaftlichen Forschung). Research of Oswin Aichholzer and Hannes Krasser ist partially supported by Acciones Integradas 2003-2004, Proj.Nr.1/2003. Suneeta Ramaswami is partially supported by a Rutgers University ISATC pilot project grant. Jorge Urrutia is supported in part by CONACYT grant 37540-A and grant PAPIIT.

References

[1] O. Aichholzer, D. Bremner, E. D. Demaine, F. Hurtado, E. Kranakis, H. Krasser, S. Ramaswami, S. Sethia, J. Urrutia: Playing with triangulations. Manuscript in preparation.

[2] O. Aichholzer, D. Bremner, E. D. Demaine, F. Hurtado, E. Kranakis, H. Krasser, S. Ramaswami, S. Sethia, J. Urrutia: Games on Triangulations: Several Variations. Manuscript in preparation.

[3] E. R. Berlekamp: The Dots and Boxes Game: Sophisticated Child's Play. Academic Press (1982)

[4] E. R. Berlekamp, J. H. Conway, R. K. Guy: Winning Ways for your Mathematical Plays. Academic Press (1982). Second edition in print, A K Peters Ltd., (2001)

[5] J. H. Conway: On Numbers and Games. Academic Press (1976). Second edition, A K Peters Ltd. (2002)

[6] E. D. Demaine: Playing games with algorithms: Algorithmic combinatorial game theory. In: Proc. 26th Symp. on Math Found. in Comp. Sci., Lect. Notes in Comp. Sci., Springer-Verlag (2001)

[7] A. S. Fraenkel: Combinatorial Games: Selected Bibliography with a Succinct Gourmet Introduction. Electronic Journal of Combinatorics, `http://www.wisdom.weizmann.ac.il/~fraenkel`

[8] J. Galtier, F. Hurtado, M. Noy, S. Pérennes, J. Urrutia: Simultaneous edge flipping in triangulations. Submitted.

Cutting Triangular Cycles of Lines in Space*

Boris Aronov† Vladlen Koltun‡ Micha Sharir§

Abstract

We show that a collection of lines in 3-space can be cut into a sub-quadratic number of pieces, such that all depth cycles defined by triples of lines are eliminated. This partially resolves a long-standing open problem in computational geometry, motivated by hidden-surface removal in computer graphics.

1 Introduction

Historical background. The chief goal of most computer graphics applications is to correctly depict ('render') a synthetic 3-dimensional scene onto the computer screen. The geometry of the scene is often represented by a collection of triangles. Correct rendering means, in particular, resolving situations where some object partly occludes another; we want to correctly draw the objects that lie closer to the viewpoint, and avoid drawing the occluded parts.

The importance of determining which parts of the scene objects are occluded was recognized in computer graphics from its very beginning. Until the 1970s, 'Hidden-Surface Removal' (HSR) was considered one of computer graphics' most important problems, and has received a substantial amount of attention; see [12] for a survey of the ten leading HSR algorithms circa 1974.

A commonly used HSR technique is the *z-buffer* [3], which produces a 'discrete' solution to the problem. Given a computer screen with a specific resolution, the z-buffer heuristically determines for each pixel on the screen the object that is closest to the viewpoint inside the area represented by the pixel. Since the z-buffer yields to efficient implementations in hardware, it is usually the HSR method of choice. It is not, however, applicable in all situations. Since its output consists of a finite number of samples, instead of an analytic description of the visible part of the scene, it does not provide the data necessary for vector-based output devices, and is highly inefficient in terms of memory consumption when dealing for example with high-quality large-scale printing tasks, which require producing images at exceedingly high resolutions. An analytic solution requires little memory and storage space, and can be used to produce images of arbitrary resolution.

These considerations motivated a long study of hidden-surface removal in computational geometry, culminating in the early 1990s with a number of algorithms that provide both conceptual simplicity and satisfactory running-time bounds. See de Berg [2] and Dorward [6] for overviews of these developments, and Overmars and Sharir [9] for a simple HSR algorithm with good theoretical running-time bounds.

A common feature of most HSR algorithms is that they rely on the existence of a consistent *depth order* for the input objects. For example, if object A occludes part of object B, and object B partially occludes object C, it is assumed that C will not occlude any part of A;

*Part of the work on this paper has been carried out at the U.S.-Israeli Workshop on Geometric Algorithms, held in Jackson Hole, WY, in the summer of 2002. Work on the paper by Boris Aronov and Micha Sharir has been supported by a joint grant from the U.S.-Israeli Binational Science Foundation. Work by Vladlen Koltun and Micha Sharir has also been supported by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing). Work by Boris Aronov was also supported by NSF Grants CCR-99-72568 and ITR CCR-00-81964. Work by Vladlen Koltun was also supported by NSF Grant CCR-01-21555 and by the Rothschild Post-doctoral Fellowship. Work by Micha Sharir was also supported by NSF Grants CCR-97-32101 and CCR-00-98246, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

†Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201-3840, USA; *aronov@ziggy.poly.edu*.

‡Computer Science Division, University of California, Berkeley, CA 94720-1776, USA; *vladlen@cs.berkeley.edu*.

§School of Computer Science, Tel Aviv University, Tel-Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA; *michas@post.tau.ac.il*.

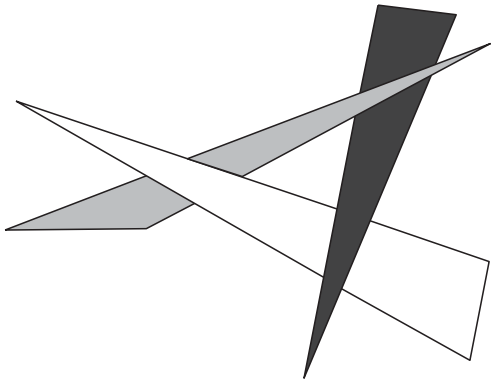


Figure 1: A depth cycle defined by three triangles.

the contrary situation is termed a *depth cycle*, or, simply, a *cycle*; see Figure 1. More precisely, it is assumed that the transitive closure of the relationship $A \prec B$, defined as A occluding part of B, is a partial order. This assumption is not always satisfied in practice, where depth cycles are easily encountered in real-world scenes involving tree branches, industrial pipes, etc. Nevertheless, the reliance on a consistent depth order is crucial to HSR algorithms, most of which begin by sorting the objects either front-to-back (e.g., the Overmars-Sharir algorithm [9]) or back-to-front (e.g., the classical Painter's Algorithm [12]).

A large number of algorithms have been developed for *testing* whether the depth relationship in a collection of triangles contains a cycle with respect to a specific viewpoint; see de Berg [2] and the references therein. However, while these algorithms help detect cycles, they do not provide strategies for dealing with them.

One such strategy is to eliminate all depth cycles, with respect to a specific viewpoint, by cutting the objects into pieces that do not form cycles, and running an HSR algorithm on the resulting collection of pieces. In 1980, Fuchs et al. [7] introduced *Binary Space Partition (BSP) trees*, which can be used to perform the described cutting. However, a BSP tree may force up to a quadratic number of cuts [10], which is problematic in light of the fact that virtually all of the research into hidden-surface removal has concentrated on the development of output-sensitive algorithms that run in subquadratic time whenever possible [2, 6].

It has been open since 1980 whether one can devise an algorithm that, given a specific viewpoint and a collection of n triangles in $\mathbb{R}^3$, removes all depth cycles defined by this collection with respect to the viewpoint using a subquadratic number of cuts. The work of Solan [11] and of Har-Peled and Sharir [8] implies that this is indeed possible, provided a subquadratic number of cuts is known to be sufficient. In particular, these works present algorithms that, given a collection $\mathcal{L}$ of n lines in 3-space, perform close to $O(n\sqrt{C})$ cuts that eliminate all cycles defined by $\mathcal{L}$ as seen from $z = -\infty$, where C is the minimal required number of such cuts.[1] That is, if we can provide a subquadratic bound on the minimum number of cuts that suffice to eliminate all cycles defined by a collection of lines, then the aforementioned algorithms are guaranteed to find a collection of such cuts of (potentially larger but still) subquadratic size.

Such an upper bound has however remained elusive. The only progress in this direction is due to Chazelle et al. [4], who in 1992 have analyzed the following special case of the problem. A collection of line segments in the plane is said to form a *grid* if it can be partitioned into two subcollections of 'red' and 'blue' segments, such that all red (resp., blue) segments are pairwise disjoint, and all red (resp., blue) segments intersect all blue (resp., red) segments in the same order; see Figure 2. Chazelle et al. [4] have shown that if the xy-projections of a collection of n segments in 3-space form a grid, then all cycles defined by this collection (again, as seen from $z = -\infty$) can be eliminated with $O(n^{9/5})$ cuts.

Our contribution. This paper describes the first step towards obtaining subquadratic general upper bounds on the number of cuts that are sufficient to eliminate all cycles defined by a collection of lines in space. Specifically, we

[1] It can be easily shown that stating the problem in terms of collections of lines, instead of the original setting of triangles, does not diminish the problem complexity but does simplify the exposition of the results. Moreover, we can assume without loss of generality that the viewpoint lies at $z = -\infty$, relying on an appropriate transformation of the 3-dimensional space. All previous work on cutting cycles has thus been done with regard to collections of lines or line segments that are viewed from $z = -\infty$ [4, 8, 11]. Since any cycle defined by a collection of line segments is also a cycle in the collection of lines spanned by these segments, we will concentrate on the case of lines.

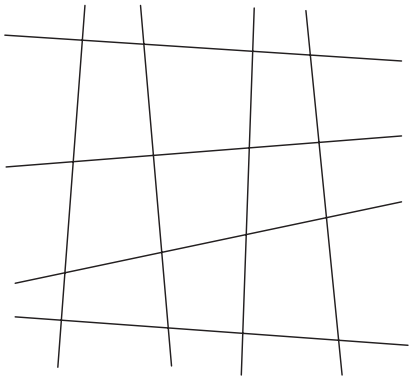


Figure 2: A collection of line segments that forms a grid.

show that all *triangular cycles*, which are cycles formed by triples of lines, can be eliminated with $O(n^{2-1/69+\varepsilon})$ cuts, for an arbitrarily small $\varepsilon > 0$. While this bound is still far from the lower bound $\Omega(n^{3/2})$ that Chazelle et al. [4] have provided for this quantity, and does not immediately apply to cycles defined by an arbitrary number of lines, it is an essential first step towards the complete solution. As the first nontrivial general upper bound for this problem, since the problem's conception more than 20 years ago, we expect it to be generalized and improved, and the techniques we introduce to be extended and simplified. A central component in our proof is a result of independent interest concerning the unrealizability of a certain weaving pattern of lines; see full version for details [1].

2 Cutting Triangular Cycles

Let us provide a formal definition for the problem of cutting cycles. Let $\mathcal{L}$ be a set of n non-vertical lines in 3-space in general position. Define the *depth order* $\prec$ on $\mathcal{L}$ to be such that $\ell \prec \ell'$ if ℓ *passes below* ℓ'; that is, the unique vertical line λ that connects ℓ and ℓ' meets them at two respective points p, p' so that the z-coordinate of p is smaller than that of p'. The relationship $\prec$ can have cycles, and our challenge is to obtain nontrivial bounds on the number of cuts that need to be applied to the lines of $\mathcal{L}$, so that the depth order among the resulting segments and rays (defined in exactly the same manner as for lines) has no cycles.

Let ℓ^* denote the xy-projection of a line ℓ, and let $\mathcal{L}^* = \{\ell^* \mid \ell \in \mathcal{L}\}$ denote the set of the projections of the lines in $\mathcal{L}$. A cycle c in $\mathcal{L}$ of the form $\ell_1 \prec \ell_2 \prec \cdots \prec \ell_j \prec \ell_1$ can be represented as a closed oriented (possibly self-intersecting or even self-overlapping) polygonal path $c^* = p_1 p_2 \ldots p_n p_1$, where p_i is the intersection point of ℓ_i^* and $\ell_{i+1 \pmod j}^*$.

The simplest kind of a cycle in the depth order is a *triangular cycle* defined by three lines ℓ_1, ℓ_2, ℓ_3, satisfying $\ell_1 \prec \ell_2 \prec \ell_3 \prec \ell_1$. We call a triangular cycle c a *clockwise* (resp., *counterclockwise*) cycle if the resulting orientation of c^* (as we trace it in the order $\ell_1^* \to \ell_2^* \to \ell_3^* \to \ell_1^*$) is clockwise (resp., counterclockwise); see Figure 3.

In this paper we confine our study to triangular cycles; thus from now on, the unqualified term 'cycle' will always refer to a triangular cycle. We therefore wish to cut the lines in $\mathcal{L}$ so that all such cycles are eliminated. Here is a simple procedure that achieves this goal. Fix a parameter k to be determined later. For each $\ell \in \mathcal{L}$, cut ℓ at (the points projecting on) every k-th vertex of $\mathcal{A}(\mathcal{L}^*)$ lying on ℓ^*. The total number of cuts is $O(n^2/k)$. It is easy to see that after these cuts are performed, any cycle c that has not been eliminated has the property that c^* is crossed by at most $3k/2$ lines of $\mathcal{L}^*$. Using the probabilistic analysis technique of Clarkson and Shor [5], the overall number of these 'light' triangular cycles is $O(k^3 \nu_0(n/k))$, where $\nu_0(m)$ is the maximum number of triangular cycles c in a collection of m lines in space, such that c^* is a *face* in the arrangement of the projected lines. (We refer to cycles of the latter type as *empty*.) Hence, we can certainly eliminate all triangular cycles in $\mathcal{L}$ using

$$O\left(\frac{n^2}{k} + k^3 \nu_0\left(\frac{n}{k}\right)\right) \qquad (1)$$

cuts.

Let C be a family of triples (ℓ_1, ℓ_2, ℓ_3) of distinct lines of $\mathcal{L}$, such that each triple in C forms a counterclockwise triangular cycle whose xy-projection is a face of $\mathcal{A}(\mathcal{L}^*)$. It suffices to obtain a bound on $|C|$, since the overall number of triangular face cycles is at most twice this bound. Such a bound is given in the following theorem, whose proof constitutes the main technical part of the full version of this paper [1].

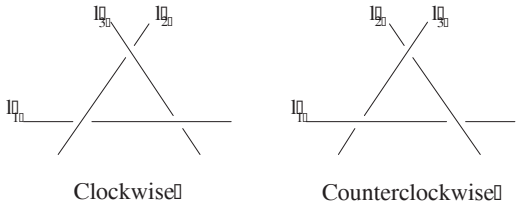


Figure 3: The two kinds of triangular cycles.

Theorem 2.1. *Given a set $\mathcal{L}$ of n nonvertical lines in $\mathbb{R}^3$ in general position, the number of empty triangular counterclockwise cycles defined by $\mathcal{L}$ is $O(n^{2-1/34+\varepsilon})$, for any $\varepsilon > 0$.*

This theorem states that $|C|$ is bounded by $O(n^{2-1/34+\varepsilon})$, for any $\varepsilon > 0$, which implies that $\nu_0(n) = O(n^{2-1/34+\varepsilon})$. Plugging this estimate into (1) we conclude that the number of cuts needed to eliminate all triangular cycles in $\mathcal{L}$ is

$$O\left(\frac{n^2}{k} + k^3\left(\frac{n}{k}\right)^{2-1/34+\varepsilon}\right) = O\left(\frac{n^2}{k} + k^{35/34-\varepsilon}n^{2-1/34+\varepsilon}\right).$$

Choosing $k = n^{1/69}$, and replacing ε by an appropriate multiple, we obtain the main result of this paper.

Theorem 2.2. *A set $\mathcal{L}$ of n nonvertical lines in $\mathbb{R}^3$ in general position can be cut into $O(n^{2-1/69+\varepsilon})$ segments and rays, for any $\varepsilon > 0$, such that no triangular cycles are present in the depth order of these portions of the lines.*

The interested reader is referred to the full version [1] for the missing technical details.

Acknowledgments

The authors wish to express their gratitude to Pankaj Agarwal, Sariel Har-Peled and Shakhar Smorodinsky for insightful suggestions concerning the material presented in this paper.

References

[1] B. Aronov, V. Koltun and M. Sharir. Cutting triangular cycles of lines in space. http://www.cs.berkeley.edu/~vladlen/cycles-conf.zip

[2] M. de Berg, *Ray Shooting, Depth Orders and Hidden Surface Removal*, Lecture Notes Comput. Sci., 703, Springer Verlag, Berlin, 1993.

[3] E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. Thesis, UTEC-CSC-74-133, Dept. Comput. Sci., University of Utah, 1974.

[4] B. Chazelle, H. Edelsbrunner, L.J. Guibas, R. Pollack, R. Seidel, M. Sharir and J. Snoeyink, Counting and cutting cycles of lines and rods in space, *Comput. Geom. Theory Appls.* 1 (1992), 305–323.

[5] K. Clarkson and P. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.* 4 (1989), 387–421.

[6] S. E. Dorward, A survey of object-space hidden surface removal, *Internat. J. Comput. Geom. Appl.* 4 (1994), 325–362.

[7] H. Fuchs, Z. M. Kedem and B. Naylor, On visible surface generation by a priori tree structures, *Comput. Graph.* 14 (1980), 124–133.

[8] S. Har-Peled and M. Sharir, On-line point location in planar arrangements and its applications, *Discrete Comput. Geom.* 26 (2001), 19–40.

[9] M. H. Overmars and M. Sharir, A simple output-sensitive algorithm for hidden surface removal, *ACM Transactions on Graphics* 11 (1992), 1–11.

[10] M. S. Paterson and F. F. Yao, Efficient binary space partitions for hidden-surface removal and solid modeling, *Discrete Comput. Geom.* 5 (1990), 485–503.

[11] A. Solan, Cutting cycles of rods in space, *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, 1998, 135–142.

[12] I. E. Sutherland, R. F. Sproull and R. A. Schumacker, A characterization of ten hidden-surface algorithms, *ACM Comput. Surv.* 6 (1974), 1–55.

Red-Blue Separability Problems in 3D *

Ferran Hurtado † Carlos Seara† Saurabh Sethia‡

1 Introduction

Let B and R be two disjoint sets of points in 3D classified as *blue* and *red* points, respectively. Let n be the number of points in B and R. We consider the sets of points in general position, thus there are no four points in a plane and no three points on a line. Let $\mathcal{C}$ be a family of surfaces in 3D. The sets B and R are $\mathcal{C}$ *separable* if there exists a surface $S \in \mathcal{C}$ such that every connected component of $\mathbb{R}^3 - S$ contains points only from B or from R. If S is a plane, we have *linear separability*. The decision problem of linear separability for two disjoint sets of objects (points, segments, polygons or circles) in 2D or (points, segments, polyhedra or spheres) in 3D can be solved in linear time [8, 13].

In [1, 10, 11] the authors study the separability of two disjoint point sets in the plane by the following criteria: *wedge separability*, *strip separability* and *double wedge separability*. Optimal $\Theta(n \log n)$ time algorithms for deciding wedge, strip, and double wedge separability, as well as for constructing the locus of feasible apices of wedges and double wedges, and the interval of feasible slopes of strips are described in [1, 10, 11]. They have also shown how to find wedges and double wedges with maximum and minimum aperture angle, and the narrowest and the widest strips.

In this abstract we summarize results from our work on separability of two disjoint point sets in 3D that extends the criteria above and gives solutions to various separability problems. For each separability criterion we consider the problem of deciding whether that particular separability is feasible, which is probably equivalent to finding one solution to the problem; in some cases, we also consider the problem of finding partial descriptions of all feasible solutions. For some separability criteria we consider, the convex hull of either the red or blue points needs to be monochromatic. If it is so, we perform this check as a first step in our algorithms in $O(n \log n)$ time by computing $CH(R)$ and $CH(B)$ [14]. First, we consider the problem of computing all the feasible solutions for linear separability. Second, we study slice, wedge, and diwedge separability as the natural extensions in 3D for strip, wedge, and double wedge separability in 2D. Third, we study the decision problem for prismatic, pyramidal, and dipyramidal separability, which also can be considered as extensions in 3D for strip, wedge, and double wedge separability criteria, but allowing a linear number of planes. Finally, we study some separability criteria defined by a constant number of planes.

We provide proof for one of our results (slice separability) and state others without proof due to space constraint.

2 Linear separability

The decision problem of linear separability for two disjoint point sets in 3D can be solved in linear time [13]. The problem of computing all feasible solutions is solved by the next theorem.

Theorem 1. *The locus of all the planes separating B and R can be computed in $\Theta(n \log n)$ time. Once we have pre-computed the locus, to decide whether a given plane separates the point sets can be done in $O(\log n)$ time.*

The problem of computing the maximum Euclidean distance between two parallel separating planes has been solved by Houle [8] with the following theorem.

Theorem 2. [8] *Given two point sets in $\mathbb{R}^d$, then a separating hyperplane which minimizes*

*Partially supported by grants DGES-MEC-SEUID PB98-0933, MCYT-FEDER BFM2002-0557, DURSI 2001SGR00224 and Joint Commission USA-Spain for Scientific and Technological Cooperation Project 98191.

†Dpt. Matemàtica Aplicada II, U.P.C., Barcelona, Spain, {hurtado, seara}@ma2.upc.es

‡Department of Computer Science, Oregon State University, Corvallis, OR 97331, USA, saurabh@cs.orst.edu

the orthogonal Euclidean distance between the hyperplane and the point sets may be found, or its non-existence determined, in $O(n)$ time.

A consequence is the following corollary.

Corollary 1. *The widest separating slice defined by two parallel separating planes of B and R can be computed in $O(n)$ time.*

3 Separability by two planes

In this section we study slice, wedge, and di-wedge separabilities, which involve exactly two separating planes.

3.1 Slice separability

A slice is defined as the space between two parallel planes. The normal vector to the planes of the slice is called the *slice direction*. The slice separability problem asks the question: Is there a slice that contains all the red points but does not contain any blue point or vice versa?

Theorem 3. *Deciding whether the sets B and R are slice separable can be done in $O(n^3)$ time. The locus of slice directions of all feasible solutions can be computed in $O(n^3 \log n)$ time and the complexity of the locus is $\Theta(n^2)$.*

Proof. (Sketch) Let u be the *slice direction* of a separating slice which contains R. Let B_1 (B_2) be the set of blue points above (below) the slice. A plane π_1 with normal vector u passing through any red point r is a separating plane of B_1 and B_2. While keeping the point r on π_1, we can move u around with two degrees of freedom until it bumps into two blue points b_1 and b_2. Thus, a separating slice (if it exists) can be found by the following $O(n^3)$ time algorithm.

1. Choose a red point $r \in R$.
2. For each pair of blue points b_1 and b_2, compute the plane π_1 passing through b_1, b_2, and r. Compute B_1 (B_2), the set of blue points above (below) π_1.
3. By linear programming compute a separating slice (if it exists) defined by two parallel planes such that one separates B_1 from $R \cup B_2$ and, the other one separates $R \cup B_1$ from B_2.

In order to compute the set of slice directions of all the feasible solutions we proceed as follows.

3.1. Compute $CH(B_1)$, $CH(B_2)$, $CH(R \cup B_1)$, and $CH(R \cup B_2)$.

3.2. Compute the region on the unit sphere formed by the set of directions of the separating planes between $CH(B_1)$ and $CH(R \cup B_2)$. Proceed analogously with $CH(R \cup B_1)$ and $CH(B_2)$. We obtain two regions each bounded by at most two convex chains with $O(n)$ complexity. The projection of each region on the plane $z = 1$ is either a convex polygon or two unbounded convex polygons.

3.3. Compute the intersection of the two regions in $O(n)$ time. Its boundary corresponds to parallel planes which are always touching $CH(R)$ and some blue point.

Thus, in additional $O(n \log n)$ time, we can compute the convex region on the unit sphere defined by the set of slice directions of all feasible solutions for each good partition. In full paper, we also show that the locus on the unit sphere of the set of slice directions of all feasible solutions is formed by at most $O(n^2)$ connected components, each one with at most linear complexity, and the total complexity of the locus is $\Theta(n^2)$. □

The width of a separating slice is defined by the distance between the two parallel planes. It is natural to ask about the narrowest and the widest separating slices.

Theorem 4. *Once we have pre-computed the set of slice directions of the separating slices for B and R, the widest and the narrowest separating slices can be computed in $O(n^3)$ and $O(n^2 \log n)$ time, respectively.*

3.2 Wedge separability

Two intersecting planes divide the 3D space into four quadrants. Any one quadrant is called a 3D wedge. The wedge separability problem is the following: Is there a wedge that contains R but does not contain any blue point or vice versa?

Theorem 5. *Deciding whether B and R are wedge separable can be done in $O(n^3)$ time.*

For wedge separable point sets we ask about a separating wedge with maximum or minimum aperture angle. The wedge with minimum aperture angle is related to slice separability (a slice can be considerer as a wedge with aperture angle $0°$) and the wedge with maximum aperture angle is related to linear separability (if the aperture angle is close to $180°$ then, we have a good approximation to linear separability).

Theorem 6. *Computing a separating wedge for B and R with maximum or minimum aperture angle can be done in $O(n^3 \log n)$ time.*

A natural problem is to decide whether there exists a separating wedge with a fixed aperture angle θ, $0° \leq \theta \leq 180°$. Note that if we have pre-computed the separating wedges with maximum and minimum aperture angle, the problem is not solved because it may be that B and R are not wedge separable for all possible values of aperture angle between the minimum and the maximum.

Theorem 7. *Computing a separating wedge for B and R with fixed aperture angle can be done in $O(n^3 \log n)$ time.*

3.3 Diwedge separability

Two intersecting planes divide the 3D space into four quadrants. The union of a pair of opposite quadrants is called a *diwedge*. The diwedge separability problem is the following: Is there a diwedge that contains R but does not contain any point from B or vice versa?

Theorem 8. *Deciding whether the sets B and R are diwedge separable can be done in $O(n^4)$ time.*

We define the *aperture angle* of a diwedge to be the bigger of the two aperture angles defined by the planes of the diwedge. We consider the problem of computing the separating diwedge with maximum or minimum aperture angle. As per the definition, the aperture angle can take values between 90° and 180°.

Theorem 9. *Computing a separating diwedge for B and R with maximum or minimum aperture angle can be done in $O(n^4 \log n)$ time.*

We also consider the problem of deciding whether there exists a separating diwedge with a fixed aperture angle θ, $90° \leq \theta < 180°$.

Theorem 10. *Computing a separating diwedge for B and R with fixed aperture angle can be done in $O(n^4 \log n)$ time.*

4 Separability by a linear number of planes

In this section we study other separability criteria which can be consider also as extensions for slice, wedge and double wedge separability in the plane, but allowing a linear number of planes. More precisely, we extend the concepts above to prismatic, pyramidal and dipyramidal separability, respectively.

4.1 Prismatic separability

A prism is defined as the space swept by a convex polygon when it is moved along a line perpendicular to its plane; the direction of this line is called the *prism direction*. The prismatic separability problem asks the question: Is there an infinite prism which contains all red points but none of blue points or vice versa?

Theorem 11. *Deciding whether the sets B and R are prismatic separable and computing the locus of the prism directions of all feasible solutions can be done in $O(n^3)$ time. The locus is formed by at most $O(n^2)$ connected components and its total complexity is $O(n^2\alpha(n))$.*

A still open problem is how can we find a minimum (in number of faces) separating prism? The minimum prism has at least three faces, hence this must address the question of deciding triangular prismatic separability which will be consider in section 5. We can compute the minimum separating prism for a given direction of prism u. This problem is equivalent to computing the minimum (in number of edges) convex polygon which separates the projected red and blue points on a plane with normal vector u; this problem can be solved in $O(n \log n)$ optimal time [1, 7].

4.2 Pyramidal separability

Join all the vertices of a convex polygon to a point in space to get a pyramid. The convex polygon is called the base of the pyramid while the point in space is called its apex. An infinite pyramid is one whose base is at infinity. The pyramidal separability problem asks the question: Is there an infinite pyramid that contains all the red points but none of the blue points or vice versa?

Theorem 12. *Deciding pyramidal separability for B and R and computing the locus of apices of all the separating pyramids can be done with a randomized algorithm whose expected running time is $O(n^3 \log^2 n)$. The locus of apices of separating pyramids is formed by $O(n^3)$ connected components with $O(n^3 \log n)$ total complexity.*

Next theorem shows a deterministic algorithm for solving the decision problem.

Theorem 13. *Deciding whether the sets B and R are pyramidal separable can be done in $O(n^7)$ time.*

4.3 Dipyramidal separability

Dipyramidal separability has the same definition as pyramidal separability except that now we have two symmetrical pyramids having the same apex. Assume that the red points R are inside the possible separating dipyramid which produces a partition of R.

Theorem 14. *Deciding whether the sets B and R are dipyramidal separable can be done in $O(n^8 \log n)$ time.*

5 Separability by a constant number of planes

In this section we study some particular separability criteria which involve three to six planes. More precisely, we consider triplane, triangular prism, tetrahedral and box separability.

Three intersecting planes divide the 3D space into eight octants. The triplane separability problem asks the question: Are there three planes such that each of the octants they define has points of only one color?

Theorem 15. *Deciding whether the sets B and R are triplane separable can be done in $O(n^7)$ time.*

The triangular prismatic separability problem asks the question: Is there an infinite triangular prism which contains all the red points but none of the blue points or vice versa?

Theorem 16. *Deciding whether the sets B and R are triangular prismatic separable can be done in $O(n^5)$ time.*

The tetrahedral separability problem asks the question: Is there a tetrahedron that contains all the red points but none of the blue points or vice versa?

Theorem 17. *Deciding whether the sets B and R are tetrahedral separable can be done in $O(n^7)$ time.*

The box separability problem asks the question: Is there an orthogonal box which contains all the red points but none of the blue or vice versa?

Theorem 18. *Deciding whether the sets B and R are box separable can be done in $O(n^7)$ time.*

References

[1] E. M. Arkin, F. Hurtado, J. S. B. Mitchell, C. Seara, S. S. Skiena, *Some Lower Bounds on Geometric Separability Problems*, 11th Fall Workshop on Computational Geometry, 2001.

[2] B. Aronov, M. Sharir, *The Common Exterior of Convex Polygons in the Plane*, Computational Geometry: Theory and Applications, 8, 1997, pp. 139–149.

[3] B. Aronov, M. Sharir, B. Tagansky *The Union of Convex Polyhedral in Three Dimensions*, SIAM J. Comput., 26 (6), 1997, pp. 1670–1688.

[4] B. Chazelle, H. Edelsbrunner, *An Optimal Algorithm for Intersecting Line Segments*, Journal of ACM, 39, 1992, pp. 1–54.

[5] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, *Diameter, Width, Closest Line Pair, and Parametric Searching*, 8th Annual Symp. on Comput. Geom., 1992, pp. 120–129

[6] G. Davis, *Computing Separating Planes for a Pair of Disjoint Polytopes*, First Annual Symp. on Comput. Geom., 1985, pp. 8–14.

[7] H. Edelsbrunner, F. P. Preparata, *Minimum Polygonal Separation*, Information and Computation, 77, 1988, pp. 218–232.

[8] M. E. Houle, *Algorithms for Weak and Wide Separation of Sets*, Discrete Applied Mathematics, Vol. 45, 1993, pp. 139–159.

[9] M. E. Houle and G. T. Toussaint, *Computing the Width of a Set*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No. 5, 1988, pp. 761–765.

[10] F. Hurtado, M. Noy, P. A. Ramos, C. Seara, *Separating Objects in the Plane with Wedges and Strips*, Discrete Applied Mathematics, Vol. 109, 2001, pp. 109–138.

[11] F. Hurtado, M. Mora, P. A. Ramos, C. Seara, *Two Problems on Separability with Lines and Polygonal*, 15th European Workshop on Computational Geometry, 1999, pp. 33–35.

[12] G. O. Katona, *On a Problem of L. Fejes Tóth*, Stud. Sci. Math. Hung., 12, 1977, pp. 77–80.

[13] N. Megiddo, *Linear-time Algorithms for Linear Programming in $\mathbb{R}^3$ and Related Problems*, SIAM J. Comput., 12 (4), 1983, pp. 759–776.

[14] F. P. Preparata, S. J. Hong, *Convex Hulls of Finite Sets of Points in Two and Three Dimensions*, Comm. of ACM, 20, 1977, pp. 87–93.

The Maximum Number of Edges in a Three-Dimensional Grid-Drawing *

Prosenjit Bose[‡] Jurek Czyzowicz[§] Pat Morin[‡] David R. Wood[‡]

Abstract

An exact formula is given for the maximum number of edges in a graph that admits a three-dimensional grid-drawing contained in a given bounding box.

A *three-dimensional* (*straight-line*) *grid-drawing* of a graph represents the vertices by distinct points in $\mathbb{Z}^3$, and represents each edge by a line-segment between its endpoints that does not intersect any other vertex, and does not intersect any other edge except at the endpoints. A folklore result states that every (simple) graph has a three-dimensional grid-drawing (see [2]). We therefore are interested in grid-drawings with small 'volume'.

The *bounding box* of a three-dimensional grid-drawing is the axis-aligned box of minimum size that contains the drawing. By an $X \times Y \times Z$ *grid-drawing* we mean a three-dimensional grid-drawing, such that the edges of the bounding box contain X, Y, and Z grid-points, respectively. The *volume* of a three-dimensional grid-drawing is the number of grid-points in the bounding box; that is, the volume of an $X \times Y \times Z$ grid-drawing is XYZ. (This definition is formulated to ensure that a two-dimensional grid-drawing has positive volume.) Our main contribution is the following extremal result.

Theorem 1. *The maximum number of edges in an $X \times Y \times Z$ grid-drawing is exactly*

$$(2X-1)(2Y-1)(2Z-1) - XYZ \ .$$

Proof. Consider an $X \times Y \times Z$ grid-drawing of a graph G with n vertices and m edges. Let P be the set of points (x, y, z) contained in the bounding box such that $2x$, $2y$, and $2z$ are all integers. Observe that $|P| = (2X-1)(2Y-1)(2Z-1)$. The midpoint of every edge of G is in P, and no two edges share a common midpoint. Hence $m \leq |P|$. In addition, the midpoint of an edge does not intersect a vertex. Thus

$$m \ \leq \ |P| - n \ . \tag{1}$$

A drawing with the maximum number of edges has no edge that passes through a grid-point. Otherwise, sub-divide the edge, and place a new vertex at that grid-point. Thus $n = XYZ$, and $m \leq |P| - XYZ$, as claimed.

This bound is attained by the following construction. Associate a vertex with each grid-point in an $X \times Y \times Z$ grid-box B. As illustrated in Figure 1, every vertex (x, y, z) is adjacent to each of $(x \pm 1, y, z)$, $(x, y \pm 1, z)$, $(x, y, z \pm 1)$, $(x \pm 1, y \pm 1, z)$, $(x \pm 1, y, z \pm 1)$, $(x, y \pm 1, z \pm 1)$, and $(x \pm 1, y \pm 1, z \pm 1)$, unless such a grid-point is not in B. It is easily seen that no two edges intersect, except at a common endpoint. Furthermore, every point in P is either a vertex or the midpoint of an edge. Thus the number of edges is $|P| - XYZ$. □

[‡]School of Computer Science, Carleton University, Ottawa, Ontario, Canada.
E-mail: {jit,morin,davidw}@scs.carleton.ca
[§]Département d'informatique et d'ingénierie, Université du Québec en Outaouais, Gatineau, Québec, Canada.
E-mail: jurek@uqo.ca
*Research supported by NSERC.

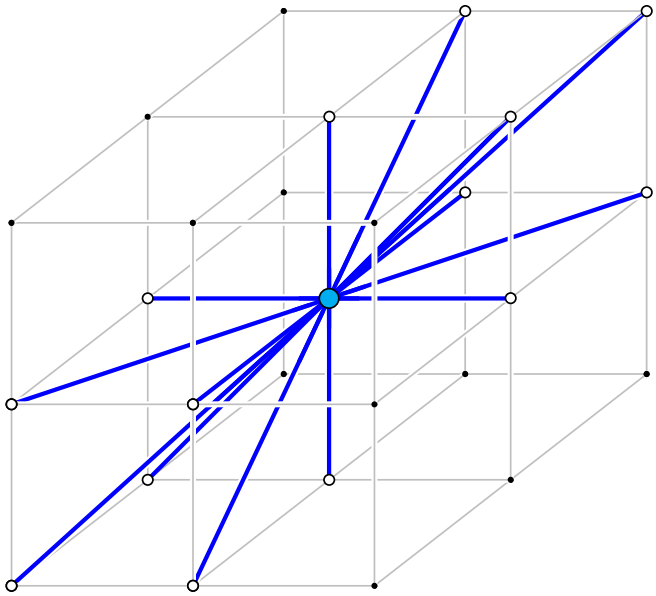


Figure 1: The neighbourhood of a vertex.

Theorem 1 can be interpreted as a lower bound on the volume of a three-dimensional grid-drawing of a given graph. Many upper bounds on the volume of three-dimensional grid-drawings are known [1–9]. There are two known non-trivial lower bounds for specific families of graphs. Cohen, Eades, Lin, and Ruskey [2] proved that the minimum volume of a three-dimensional grid-drawing of the complete graph K_n is $\Theta(n^3)$. The lower bound follows from the fact that K_5 is not planar, and hence at most four vertices can lie in a single grid-plane. The second lower bound is due to Pach, Thiele, and Tóth [7], who proved that the minimum volume of a three-dimensional grid-drawing of the complete bipartite graph $K_{n,n}$ is $\Theta(n^2)$. The proof of the lower bound is based on the observation that no two edges are parallel. The result follows since the number of distinct vectors between adjacent vertices is at most a constant times the volume. The following corollary of Theorem 1 generalises this lower bound for $K_{n,n}$ to all graphs.

Corollary 1. *A three-dimensional grid-drawing of a graph with n vertices and m edges has volume greater than $\frac{m+n}{8}$.*

Proof. Let v be the volume of an $X \times Y \times Z$ grid-drawing. By (1), $m \leq |P| - n < 8v - n$, and hence $v > \frac{m+n}{8}$. □

Obviously the bound in Corollary 1 can be slightly improved by considering the exact dimensions of the bounding box. Theorem 1 generalises to multi-dimensional polyline grid-drawings (where edges may bend at grid-points) as follows.

Theorem 2. *Let $B \subseteq \mathbb{R}^d$ be a convex set. Let $S = B \cap \mathbb{Z}^d$ be the set of grid-points in B. The maximum number of edges in a polyline grid-drawing with bounding box B is at most $(2^d - 1)|S|$. If B is an $X_1 \times \cdots \times X_d$ grid-box, then the maximum number of edges is exactly*

$$\prod_{i=1}^{d}(2X_i - 1) \; - \prod_{i=1}^{d} X_i \;\; .$$

Proof. Let $P = \{x \in B : 2x \in \mathbb{Z}^d\}$. Consider a polyline grid-drawing with bounding box B. The midpoint of every edge is in P, and no two edges share a common midpoint. A drawing with the maximum number of edges has no edge that passes through a grid-point. Otherwise, sub-divide the edge, and place a new vertex at that grid-point. Thus the number of edges is at most $|P| - |S| \leq (2^d - 1)|S|$.

If B is an $X_1 \times \cdots \times X_d$ grid-box, then $|P| - |S| = \prod_{i=1}^{d}(2X_i - 1) - \prod_{i=1}^{d} X_i$. To construct a grid-drawing with this many edges, associate one vertex with each grid-point in S. Observe that every point $x \in P \setminus S$ is in the interior of exactly one unit-sized d'-dimensional hypercube with corners in S, where $1 \leq d' \leq d$. For every point $x \in P \setminus S$, add an edge passing through x between opposite vertices of the unit-sized hypercube corresponding to x. Edges only intersect at common endpoints, since these unit-sized hypercubes only intersect along their boundaries. Every point in P contains a vertex or a midpoint of an edge. Thus the number of edges is precisely $|P| - |S|$. □

References

[1] T. Calamoneri and A. Sterbini, 3D straight-line grid drawing of 4-colorable graphs. *Inform. Process. Lett.*, **63(2)**:97–102, 1997.

[2] R. F. Cohen, P. Eades, T. Lin, and F. Ruskey, Three-dimensional graph drawing. *Algorithmica*, **17(2)**:199–208, 1996.

[3] E. di Giacomo, G. Liotta, and S. Wismath, Drawing series-parallel graphs on a box. In S. Wismath, ed., *Proc. 14th Canadian Conf. on Computational Geometry (CCCG '02)*, The University of Lethbridge, Canada, 2002.

[4] V. Dujmović, P. Morin, and D. R. Wood, Path-width and three-dimensional straight-line grid drawings of graphs. In M. T. Goodrich and S. G. Kobourov, eds., *Proc. 10th International Symp. on Graph Drawing (GD '02)*, vol. 2528 of *Lecture Notes in Comput. Sci.*, pp. 42–53, Springer, 2002.

[5] V. Dujmović and D. R. Wood, Tree-partitions of k-trees with applications in graph layout. Tech. Rep. TR-02-03, School of Computer Science, Carleton University, Ottawa, Canada, 2002.

[6] S. Felsner, S. Wismath, and G. Liotta, Straight-line drawings on restricted integer grids in two and three dimensions. In P. Mutzel, M. Jünger, and S. Leipert, eds., *Proc. 9th International Symp. on Graph Drawing (GD '01)*, vol. 2265 of *Lecture Notes in Comput. Sci.*, pp. 328–342, Springer, 2002.

[7] J. Pach, T. Thiele, and G. Tóth, Three-dimensional grid drawings of graphs. In G. Di Battista, ed., *Proc. 5th International Symp. on Graph Drawing (GD '97)*, vol. 1353 of *Lecture Notes in Comput. Sci.*, pp. 47–51, Springer, 1998.

[8] T. Poranen, A new algorithm for drawing series-parallel digraphs in 3D. Tech. Rep. A-2000-16, Dept. of Computer and Information Sciences, University of Tampere, Finland, 2000.

[9] D. R. Wood, Queue layouts, tree-width, and three-dimensional graph drawing. In M. Agrawal and A. Seth, eds., *Proc. 22nd Foundations of Software Technology and Theoretical Computer Science (FST TCS '02)*, vol. 2556 of *Lecture Notes in Comput. Sci.*, pp. 348–359, Springer, 2002.

Constrained Higher Order Delaunay Triangulations*

Joachim Gudmundsson Herman Haverkort Marc van Kreveld

Dept. of Computer Science, Utrecht University,
P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.
joachim@cs.uu.nl, herman@cs.uu.nl, marc@cs.uu.nl

1 Introduction

A previous paper by Gudmundsson et al. [3] studied a new type of triangulation called *higher-order Delaunay triangulation*. It is a class of well-shaped triangulations for a given point set. Such triangulations are useful in realistic terrain modeling on a set of points in the plane with known elevation. Often, in terrain modeling it is desirable to force a given set of edges to be part of the triangulation. These edges can come from contour lines or from the drainage network [2, 4, 6]. Motivated by this, we study *constrained higher-order Delaunay triangulations* in this paper. We first repeat the definition of higher-order Delaunay triangulations:

Definition 1 *A triangulation of a set P of points is an* order-k Delaunay triangulation *if for any triangle of the triangulation, the circumcircle of that triangle contains at most k points of P.*

So a normal Delaunay triangulation is an order-0 Delaunay triangulation, and for any positive integer k, there can be many different order-k Delaunay triangulations. By definition, any order-k Delaunay triangulation is also an order-k' Delaunay triangulation if $k' > k$.

Another important concept from Gudmundsson et al. [3] is the *useful order* of an edge:

Definition 2 *For a set P of points, the* order *of an edge between two points $p, q \in P$ is the minimum number of points inside any circle that passes through p and q. The* useful order *of an edge is the lowest order of a triangulation that includes that edge.*

In this paper we study constrained higher-order Delaunay triangulations, which must include a given set of edges in the triangulation. Note that the order of a Delaunay triangulation with only one constraining edge is exactly the useful order of that edge. This paper studies the case of more than one constraining edges. We study the following questions:

1. Given a triangulation T (all edges are constraining), determine its order.
2. Given a set P of n points and a set E of edges, determine the lowest order Delaunay triangulation of P that includes the edges of E.

The first question we can solve in two ways. Circular range counting gives an efficient algorithm for large orders, and higher-order Voronoi diagrams are the basis of an efficient algorithm for lower orders.

The main result we have for the second question is that if every edge in E has useful order k or less, then a triangulation of E and P exists that has order at most $2k - 1$. In fact, this triangulation is the constrained Delaunay triangulation. The bound is worst-case optimal: there are point sets with constraining edges, all of useful order k or less, for which any triangulation has order at least $2k - 1$.

Throughout this paper we assume general position, that is, no three points of a point set P lie on a line, and no four points of P lie on one circle.

*J.G. is supported by the Swedish Foundation for International Cooperation in Research and Higher Education. H.H. is supported by the Netherlands Organization for Scientific Research (NWO).

2 Determining the order of a triangulation

Given a triangulation T, we can determine its order k in one of two ways, based on the observations and algorithms given before in [3]. The first is efficient for any k, in particular, it is the best we can do if the unknown value k is at least $\sqrt{n}$ with some logarithmic factors. The second algorithm is more efficient when k is constant or a function that grows slower than $\sqrt{n}$ with logarithmic factors. Small values of k are expected to be most important in practical situations.

Both algorithms begin by determining the $O(n)$ circles through the three points of any triangle in the triangulation. Then we find out how many points lie in these circles. The circle containing the largest number of points determines the order of the triangulation.

The first algorithm is based on a circular range searching data structure on P that can answer point counting queries for query circles efficiently. For various storage requirements m, a data structure of space $O(m)$ exists that answers such circular range counting queries in $O(n/m^{1/3}\log(m/n))$ time [1]. The structure takes $O(m\log^{O(1)} m)$ time to construct. We choose m to be $n^{3/2}$. A triangulation gives rise to $O(n)$ circular range queries; the maximum count returned yields the order of the triangulation. So this solution takes $O(n^{3/2}\log^{O(1)} n)$ time in total.

The second solution comes down to choosing a value k' and testing whether the actual order k is less than k' or not. This can be done by computing the k'-th order VD and preprocessing it for point location queries. A query returns the k'-th closest point. To find out — for a query circle — whether it contains less than k' points, we query with the center of the circle and find the k'-th closest point, which is tested explicitly for containment in the circle. If for all $O(n)$ query circles the k'-th closest point lies outside, we know that the order is less than k'.

The k'-th order VD can be computed and preprocessed for planar point location in $O(nk'\log n)$ time [5]. We start with $k' = 1$, and if k appears to be larger, we double k' and test again. After at most $O(\log k)$ attempts, we find an interval of values $[2^i, 2^{i+1}]$ that must contain k. By binary search on this interval, we take another $O(\log k)$ steps to determine the exact order of the triangulation T. So in total, this method takes $O(nk\log n\log k)$ time.

Theorem 1 *Given a triangulation with n vertices, its order k can be determined in $O(n^{3/2}\log^{O(1)} n)$ time and in $O(nk\log n\log k)$ time.*

3 Completing to a Higher Order Delaunay Triangulation

Assume that a set P of n points and a set E of edges are given. P must include the endpoints from E. This section deals with computing a triangulation of P that includes the edges of E. We would like the triangulation to have the lowest possible order.

As mentioned in the introduction, a previous paper [3] includes the case $|E| = 1$. In case there is only one constraining edge $\overline{uv}$, we can determine the lowest k for which $\overline{uv}$ is a useful order-k Delaunay edge. Then we can complete it to a triangulation only using triangles whose circumcircle contains no more than k points. One of the triangles incident to $\overline{uv}$ has order k, or both, and no other triangle needs to have higher order. In the completion, $\overline{uv}$ will be part of triangles $\triangle uvs$ and $\triangle uvt$. Points s and t are the first points hit by a circle squeezed in between u and v from the one side and from the other side, see Figure 1(a).

The case with more constraining edges is more difficult than the case of one constraining edge. In [3] it was shown that if all edges of E are Delaunay or useful first order Delaunay, then a completion to a first order Delaunay triangulation exists and can be computed in $O(n\log n)$ time. It is simply the constrained Delaunay triangulation. But as soon as E contains edges that are useful k-order with $k > 1$, we cannot necessarily complete it to an order-k Delaunay triangulation anymore, as shown in the next theorem.

Theorem 2 *Let P be a set of points and let E be a set of edges that are all useful order-k Delaunay edges, with $k \geq 2$. Then we have:*

(i) For any sets P and E, the constrained Delaunay triangulation has order $2k-2$.

(ii) For some sets P and E, any constrained triangulation has order at least $2k-2$.

(iii) For some sets P and E, the constrained Delaunay triangulation does not have order smaller than $2k-2$, but some other constrained triangulation has order k.

Proof: We begin with (ii), which is shown by example. Figure 1(b) excluding point s shows a point set with 9 points and 2 constraining edges. Any constrained triangulation must contain $\triangle uvw$, and hence the number of points in the grey circle determines the order. The four other

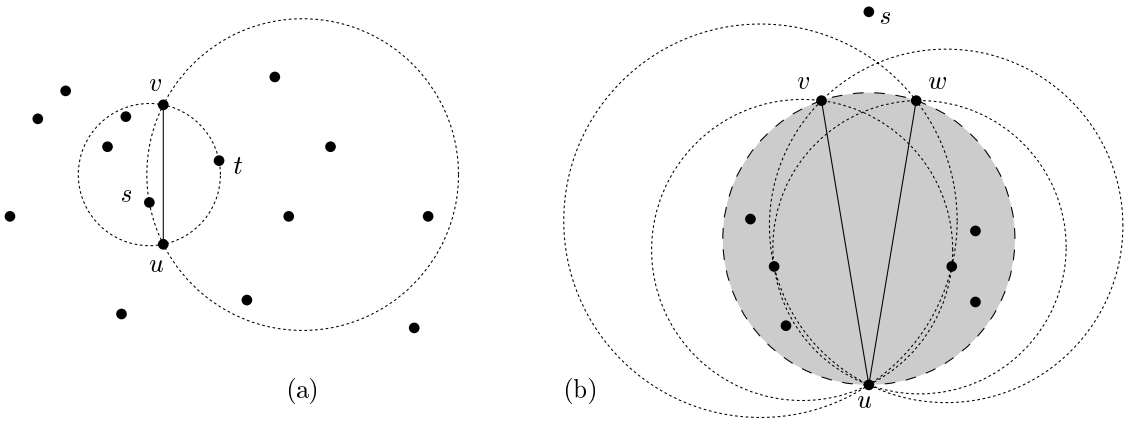


Figure 1: (a) Illustration of the first-points-hit (s and t). (b) Illustration of the proof.

circles show the useful order of the two constraining edges, which is 4. This example immediately generalizes to having $k-1$ points in each of the two circle parts left of $\vec{uv}$ and right of $\vec{uw}$. Then the edges $\overline{uv}$ and $\overline{uw}$ have useful order k, and the circle through u, v, w contains $2k-2$ points inside.

Part (iii) of the lemma also follows from Figure 1(b), now including point s. The constrained Delaunay triangulation has order $2k-2$, but flipping the edge $\overline{vw}$ to $\overline{us}$ reduces the order to k.

For part (i), consider the constrained Delaunay triangulation of P and E, and any triangle u, v, w of it. The circle through u, v, w can only contain points that are 'behind' edges of the CDT, see Figure 2. These edges must be constraining edges of E. (More correctly: for any point $p \in P$ inside the circle $C(u, v, w)$ there must be a constraining edge intersecting $C(u, v, w)$ twice and which has $\triangle uvw$ and point p on different sides.) Let $E' \subseteq E$ be the constraining edges that intersect $C(u, v, w)$ twice, separate a point of P inside $C(u, v, w)$ from $\triangle uvw$, and are closest to $\triangle uvw$ among these (that is, no other constraining edge lies in between: in Figure 2, the dashed edge is not in E').

If there is only one edge $e \in E'$, there can be at most k points behind it inside $C(u, v, w)$ because the first-point-hit for the edge e in the direction of $\triangle uvw$ will be point u, v, or w, or some point hit even before. That will give a circle with those same points inside. Since this circle is one of the two that determine the useful order of e, there can be at most k points inside. Hence, $C(u, v, w)$ can contain at most k points as well.

If E' contains at least two edges, consider any two of them, say e_1 and e_2. Let C_1 and C_2 be the circles through the endpoints of e_1 and e_2 and the first-point-hit behind the edges e_1 and e_2, respectively, see Figure 2. These two circles together cover the whole of $C(u, v, w)$. Since these circles are also the ones that determine the useful order of the constraining edges e_1 and e_2, which is at most k, the circles C_1 and C_2 can contain at most k points each. These include the points u, v, w, unless the endpoints of e_1 (or e_2) happen to be u, v, or w. But both C_1 and C_2 contain at least one of u, v, w. Hence, at most $k-1$ other points of P can lie inside each. It follows that at most $2k-2$ points of P can lie inside $C(u, v, w)$, which shows that the order of $\triangle uvw$ is at most $2k-2$. Since this triangle was any triangle of the CDT, the part (i) of the lemma follows. □

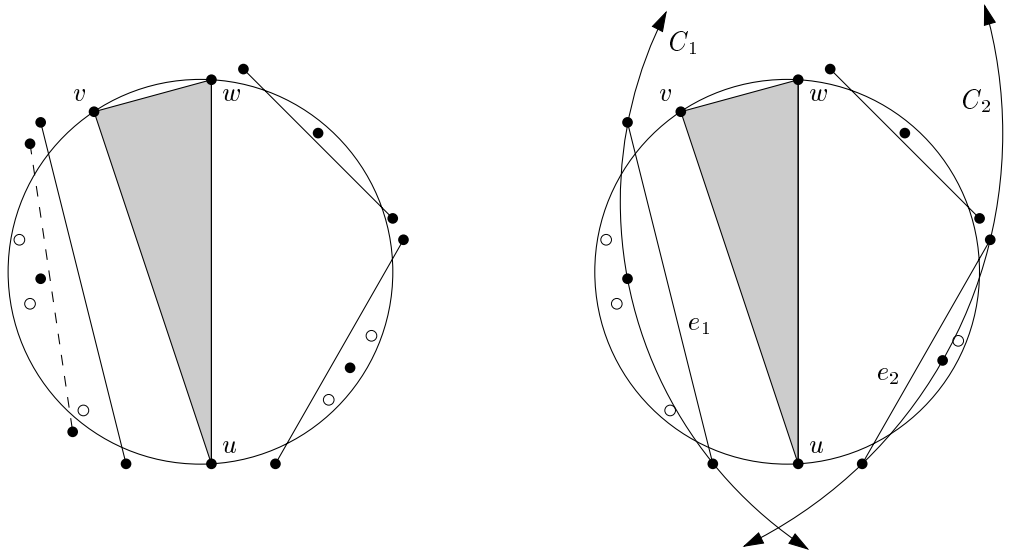


Figure 2: The order of a triangle in a constrained Delaunay triangulation.

4 Conclusions

We have extended results on higher-order Delaunay triangulations and generalized them to constrained higher-order Delaunay triangulations. The application of constrained higher order Delaunay triangulations lies in realistic terrain modeling, where a known river network gives the set of constraining edges. The next research issue is to integrate other criteria for realistic terrain modeling [6] by optimizing over the constrained higher-order Delaunay triangulations.

An open problem that arises in this paper is the computation of the lowest order completion of a set of useful order-k Delaunay edges to a triangulation. The constrained Delaunay triangulation only gives a 2-approximation of the lowest order.

References

[1] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[2] L. De Floriani and E. Puppo. A survey of constrained Delaunay triangulation algorithms for surface representaion. In G. G. Pieroni, editor, *Issues on Machine Vision*, pages 95–104. Springer-Verlag, New York, NY, 1989.

[3] J. Gudmundsson, M. Hammar, and M. van Kreveld. Higher order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 23:85–98, 2002.

[4] M. McAllister and J. Snoeyink. Extracting consistent watersheds from digital river and elevation data. In *Proc. ASPRS/ACSM Annu. Conf.*, 1999.

[5] E.A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th Annu. ACM Symp. on Computational Geometry*, pages 390–399, 1999.

[6] B. Schneider. Geomorphologically sound reconstruction of digital terrain surfaces from contours. In T.K. Poiker and N. Chrisman, editors, *Proc. 8th Int. Symp. on Spatial Data Handling*, pages 657–667, 1998.

An approach to exhaustive generation of objects without testing on isomorphisms. Application of the method to the cell growth problem

Lyuba Alboul
Sheffield Hallam University, UK
e–mail: L.Alboul@shu.ac.uk

Alexandre Netchaev*
University of Twente, the Netherlands
e–mail: netchaev@math.utwente.nl

1 Introduction

Generating and enumerating a specific class of objects are fundamental problems in discrete mathematics and related fields. In this communication we discuss a new approach to the problem of generation that allows to avoid testing on isomorphisms without producing duplicates. We illustrate our approach by presenting several algorithms to generate exhaustively some classes of non–isomorphic combinatorial objects, such as dissectible polyhedra and polygons, and triangular animals.

The usual approach to generate a class of objects is *incremental*: one starts from an initial object and adds one generating block (unit) at a time. Depending on the objects to be generated, the generating unit might be a vertex, an edge, a triangle, and so on. In the generating process one encounters, in general, two types of procedures. One procedure aims at obtaining objects with $m+1$ generating blocks from an object with m generating blocks, and the other – at obtaining from an object with m generating blocks another object with the same number of generating blocks by exchanging two generating blocks at a time, or by replacing a fixed group of generating blocks by another fixed group. The first procedure occurs in various combinatorial problems, when all objects of some class must be enumerated up to a given number of generating blocks. The second procedure might be a sub–procedure of the first procedure. One encounters the second procedure in its 'pure' form, for example, in the triangulation problem, when one needs to transform a triangulation of n points into another triangulation. In a plane such a transformation often is *flipping an edge*, and in a space - *replacing two adjacent tetrahedra by the other three*, and vice versa. In both procedures only one operation is performed at each step: adding a generating block, or exchanging generating blocks.

One of the most difficult tasks, due to its high computational expenses, in the process of enumerating/generating combinatorial objects is to avoid duplicates, especially if one deals with the problem of generating unlabelled objects. In this case the problem of eliminating isomorphic objects, or controlling isomorphisms, becomes crucial. For this reason many algorithms are restricted to generation of so-called rooted objects, when one 'detail' of the object is fixed (for example, a 'node'/vertex, or a 'side'/edge) [2]. Choosing a root destroys most of the symmetries, which makes enumeration/generation easier. However, the number of rooted objects is larger then that of non-rooted ones, and some non-isomorphic rooted objects are still isomorphic in the usual setting. Therefore, if we want to derive from 'rooted' objects non–isomorphic non–rooted ones, an additional check on isomorphisms is needed. Indeed, if an object has no symmetries, we can

*The research of the author is partially supported by the NWO (STW) (Dutch Organisation for Scientific Research), project No. TWI4816

presume that it does not matter what edge or vertex we take as a 'root', however if an object has symmetries, then some roots will be equivalent and therefore isomorphic non–rooted objects will be produced in the process of generating. On the other hand, we cannot completely rule out the possibility of generating isomorphic objects at some step of the generating process even if the initial object has no symmetries.

In our method we use a heuristic idea to fix several 'roots' at each step of the generating process in order to avoid repetitions of the objects and the consequent testing on isomorphisms. This idea leads to the logical conclusion: at each step of generation to add not only one generating block at time, but, depending on the situation, a collection of these generating blocks.

We apply our method to generate explicitly and exhaustively several types of objects: non–isomorphic simplicial *dissectible* polyhedra, their two-dimensional counterparts, and so–called *triangular animals*. The generating block is a vertex (together with a corresponding star). Therefore, in the case of dissectible polyhedra we use tetrahedra as generating blocks, and in the case of 'dissectible' polygons - triangles. We refer to generating blocks in both cases as *generating vertices*. In our approach at each step of the generating process only a fixed number of generating vertices are simultaneously added with the advantage that no testing on isomorphisms is required. Our method is not incremental, but it allows for parallelisation. In such a setting the problem is similar to the problem of simultaneously edge flipping in triangulations, recently arisen in computational geometry [6]. In [1] our first algorithm was introduced. In this communication we shortly describe a theoretical background of constructive enumeration of dissectible polyhedra without testing on isomorphisms, and then present several new algorithms to generate 'dissectible' polygons and triangular animals.

2 Main definitions and concepts

A dissectible polyhedron is inductively defined as follows [3]:

Definition 1 *1. A triangle and a tetrahedron are both dissectible polyhedra.*

2. A dissectible polyhedron with $(n+1)$ tetrahedra is obtainable from a dissectible polyhedron P with n tetrahedra by adding a new tetrahedron having precisely an exterior triangle in common with P.

The concept of dissectible polyhedron is a natural generalisation of the concept of dissections of a polygon which dates back to Euler. Euler formulated this problem as enumeration of the ways of triangulating a convex polygon by means of nonintersecting diagonals [5]. This problem is equivalent to the problem of dissecting a disk into triangular regions, since the boundary of a polygon is homeomorphic to the boundary of a disc. There are many works dedicated to the problem of dissecting a polygon or a disc. In the later case not only triangulated regions are considered [4]. The papers are mostly dedicated to the problem of enumerating dissections combinatorially. The interested reader is referred to the bibliography in [3], where also a method for combinatorial enumeration of dissectible polyhedra is given.

Our method deals with constructive enumeration of dissectible polyhedra, or, in other words, with their explicit generation. By means of the method we generate exhaustively dissectible polyhedra and polygons without duplications and testing on isomorphism. Both objects, a dissectible polyhedron and a polygon, represent triangulations. For simplicity, we refer to them as *dissectible triangulations*. We generate all possible groups of automorphisms for a given dissectible triangulation (called a *preceding triangulation*) and based on this generation we reconstruct all non-isomorphic triangulations that are 'derived' from the given one. We call those triangulations *successive triangulations*. The processes of generating groups of automorphisms and reconstructing triangulations are superposed, so the number of operations in the algorithm is reduced. In

order to avoid producing duplicates we add simultaneously a number of new generating vertices. The triangulations that are generated from different preceding ones, are then non–isomorphic by construction.

Essentially, our procedure consists of the following steps:

1. Take an initial (preceding) triangulation; define its groups of automorphisms.
2. Insert new vertices in such a way, that each group of automorphisms of the initial triangulation yields only one new (successive) triangulation (a 'derived' representative of the given automorphism group).
3. Repeat the above two steps for each new obtained triangulation.

Note. Only at the initial step a full group of automorphisms is generated. At each following step a restricted check on automorphisms suffices.

3 Results

On the base of our method several algorithms have been developed, each with a different number of generating vertices. In each subsequent algorithm we were able to reduce considerably this number. We denote the number of generating vertices by GV_i, where i corresponds to the number of the related algorithm.

In the first algorithm, developed for dissectible polyhedra, GV_1 is not less than k; k is the number of vertices of degree three in a triangulation with n vertices. Suppose, that GV_1 is equal to l, then the number l satisfies the following condition:

$$k \leqslant l \leqslant 2(n-2), \tag{1}$$

where k is a number of vertices of degree 3 of the initial/preceding triangulation, and $2(n-2)$ is the number of all faces (triangles) of a triangulation with n vertices. Indeed, we cannot add more than $2(n-2)$ vertices to a triangulation with n vertices. We add new vertices in a special way: to each star of a vertex of degree 3 one new vertex is always added. We define the vertices of degree three to which stars we add a new vertex as *active* vertices, and corresponding stars - as *active* stars. A new vertex is always added to each active star (to one of three possible faces) and the remaining $l-k$ points are added to some other faces of the same triangulation with n vertices. The latter faces do not need to belong to the stars of vertices of degree three. The remaining $l-k$ vertices are added not simultaneously, but one by one. We call our method *k-incremental*, since at the first step we always add k vertices simultaneously, but the further steps are incremental, because at each next step we add only one vertex from the remaining $l-k$ vertices that may be added. Therefore GV can be presented as $GVS + GVR$, where GVS is fixed and determined by the number of active vertices, and GVR varies. This algorithm allows a direct analogue for dissectible polygons. In this case k is the number of vertices of degree 2, GV_2 does not exceed n, where n is the number of boundary edges (equal to the number of vertices in the preceding triangulation). The following theorem is proved:

Theorem 2 *All generated dissectible triangulations are non-isomorphic and their generation is exhaustive.*

We have developed two more algorithms for dissectible polygons. The GVS has been significantly reduced. In the best algorithm is equal to 2 or 3, depending on the situation. We modify then this algorithm to generate *triangular animals*. This problem belongs to so-called *cell-growth problem* [9]. An animal is constructed from a collection of equivalent cells. A cell can be an equilateral triangle, a square or a hexahedron. The cells must be non–overlapping. Recently, the cell-growth problem has attracted attention of specialists in various fields (see, for example, [7]). Literature

dedicated to generation of animals is scarce. One of the recent works is [8]. In this work a constructive enumeration of triangular animals up to 13 cells (triangles) is given, however, the algorithm requires testing on isomorphisms . We have developed algorithms to generate simply-connected and multiply-connected triangular animals without internal vertices. Simply–connected triangular animals have been easily generated up to 18 cells (20 vertices). The results of generation are given in Tab. 1.

Number of cells	13	14	15	16	17	18
Number of animals	7541	20525	55633	152181	416188	1143526

Table 1: Counts of simply–connected triangular animals.

4 Conclusion

We presented a new approach to the problem of constructive enumeration of some classes of the objects, that excludes testing on isomorphisms. By simultaneously adding several generating blocks and by treating this operation as a primitive operation, one can expect a considerable reduction of computational cost. Another advantage is that our approach divides the computation into mutually disjoint sub-computations. In another words, our computation process is a forest structure. Sub-computation processes (trees) are completely independent, and therefore can be generated on separate processors. The open problem is to generalise the method to more complex objects. The method might also be useful in simulation of growth of other structures, such as molecular structures, corals, fractals, where a similar growth pattern is repeated at each subsequent growth step.

References

[1] Alboul, L., Netchaev, A.: Isomorphic–free generation of some classes triangulations without repetitions. In Proc. of EWCG 2002 (European Workshop in Computational Geometry), April 10-12, 2002, Warsaw, pp. 116-117

[2] Avis, D.: Generating rooted triangulations without repetitions. Algorithmica, 16 (1996), 618–632.

[3] Beineke, L.W., Pippert, R.E.: Enumerating dissectible polyhedra by their automorphism groups. Can. J. Math., 26(1):50–67, 1974.

[4] Brown, W.G.: Enumeration of quadrilangular dissections of the disc. Can. J. Math., 17 (1965), 302–317.

[5] Euler, L.: Novi commentarii academiae scientiarium imperialis petropolitanae 7 (1758-1759), 13-14.

[6] Galtier, J., Hurtado, F., Noy, M., Perennes, S., Urrutia, J.: Parallel edge flipping. See: http://www-ma2.upc.es/ hurtado/flipcorner.html

[7] Ivanov, A.O, Tuzhilin, A.A.: Branched geodesics. Geometrical theory of local minimal networks. (in Russian) Russian Research in Mathematics and Science, Vol. 5. The Elwin Mellen Press 1999. ISBN 0-7734-3178-0.

[8] Konstantinova, E.: Constructive enumeration of triangular of triangular animals. See: http://com2mac.postech.ac.kr/papers/2000/00-22.ps

[9] Palmer, E.M.: Variations of the cell-growth problem. In: Graph theory and applications. Proc. Conf. western Michigan University, May 10-13 (1972), pp. 215-224, Berlin 1972.

Kinetic Convex Hull Maintenance Using Nested Convex Hulls

Mohammad Reza Razzazi[1], Ali Sajedi[2]

[1]Software Research and Development Laboratory, Computer Engineering Department, AmirKabir University of Technology, Tehran, Iran

razzazi@ce.aut.ac.ir

[2] Software Research and Development Laboratory, Computer Engineering Department, AmirKabir University of Technology, Tehran, Iran

alisajedi@yahoo.com

ABSTRACT

In this paper we present an effective kinetic data structure and algorithm for efficient maintenance of convex hull of moving points in 2d space. Given *n* points continuously moving in the plane we give an efficient algorithm for maintaining their convex hull. Our algorithm partitions the original points into several groups, each group's points forming a convex polygon and the polygons are nested.

1- INTRODUCTION

The problem of convex hull has been exhaustively studied in computational geometry [1, 2, 3, 6], but almost in the context of static objects with operations like insertion and deletion. Our emphasis is on maintenance of convex hull under continuous motions of the given objects. Our algorithm takes advantage of concurrency and neighbourhood in motions to achieve a minimal number of combinatorial events. From this point of view our data structure is similar to the dynamic computational geometry framework introduced by Atallah [7], which studies the number of combinatorially distinct configurations of convex hull resulting from continuous motion of objects. Our data structure does not need to know the full motion of the objects in the beginning.

Bash, Guibas, and Hershberger in [8] introduced a useful technique for maintaining convex hull and closest pair of moving points in the plane called *kinetic*. Kinetic solutions are based on occurrence of events. Each event corresponds to changes in combination of a constant number of points such as reversing the sign of angle ABC, or crossing the point A with line segment BC. They called these changes 'certificates'. Events are collected and scheduled in a global event queue. In kinetic solutions we try to minimize the number of events to reduce process time and space.

A good kinetic algorithm is local, in other words, each point is involved in only polylogarithmically many certificates, and occurrence of one event does not affect so many points and events. For more information about kinetic solutions and parameters refer to [9].

In [4], [5] the convex hull algorithm is based on upper envelopes of duals of points in 2d space and calculates a good number of events. Their work is on line segments and envelopes, but our algorithm acts directly on points and convex hull of them, hence our algorithm uses a sensible and direct approach.

Our algorithm only schedules events for adjacent points in the data structure, and hence does not involves too many events.

Each object is assumed to be a point. Thus at any time we want to have the convex hull of *n* points continuously moving in a restricted area such as a rectangle. We assume that each point has a flight plan that defines the moving direction and speed of that point. This direction can only change because of a collision between the point and borders of the region. Also we assume that points can cross each other without any collision.

2- PRELIMINARIES

It is obvious that during the time between occurrences of two consequent events, the points present in convex hull does not change, in other word the convex hull is formed of the same points (with changing places) resulting the moving convex hull. We do not need to calculate the convex hull all the time. We initialize the data structure at the beginning and then only at the scheduled times apply the events, possibly changing the status of the convex hull.

In the following, we first, present a simpler version of our kinetic data structure and then to improve its locality some modifications will be applied. However, the main algorithm is the same and can use each version of the data structure.

3- KINETIC DATA STRUCTURE

Our kinetic data structure is a set of nested convex hulls (NCHs) containing all points of the problem (maybe there will be only one or two points in the most inner convex hull, that represents respectively a point or a line segment. We assume that it also represents a convex hull). The convex hulls are kept in a simple data structure such as array of linked lists (array of convex hulls) or linked list of linked lists (linked list of convex hulls). What is important is the sequence of points in each convex hull. We study each convex hull in clockwise order and **next** and **prior** pointers for each point of it.

It is obvious that having NCHs, the convex hull is always available (The convex hull of all points is always the outer

convex hull). By occurrence of any event we will update the NCHs, possibly changing the place of some points in two adjacent convex hulls or just changing the child(s) of a point, which would be defined later.

The manner of creating NCHs from initial points is as follows:

```
Algorithm createNCHs(pointsArray)      // Returns NCHs
        Input: pointsArray[1..n]
                    // Array of coordinates of input points
        Output: nested convex hull of (fixed) points
{
    S: NCHs;
    flag: array [1..n] of Boolean;
    for all points, i, in pointsArray do
        flag[i] ← false;
    while (there is any unflagged point in pointsArrsy){
        obtain the convex hull of all unflagged points of
            pointsArray, naming CH;
        add CH to S;
        set flags of all points of CH to true;
        }
    return S;
}
```

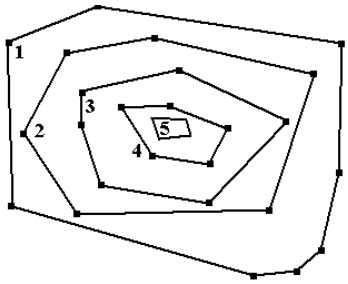


Figure 1- Nested Convex Hulls used in our kinetic data structure

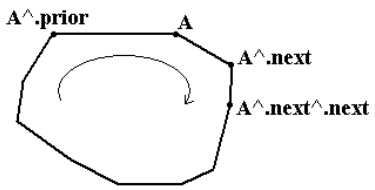


Figure 2- order of points in each convex hull is clockwise and each point has two pointers to next and prior points.

Note that only once we need to create the NCHs from initial points; the other times we change the NCHs when an event occurs. These changes are all local and we do not have a global event.

3-1- List of events

For each event we should work on the points of NCHs that cooperate in that event.

For each point there are (possibly) internal and external convex hulls. A part of NCHs is shown in Figure 3 (convex hulls A, B, C...). Lines connecting B_3 to C_2 and B_3 to C_6 indicates the FirstChild and LastChild of B_3 that will be defined for fully localizing the algorithm.

These definitions for a point, X, as follows:

- firstChild: Consider a point, P, on immediate inner layer of X (IL(X)) which is not visible from X. Moving clockwise from p on IL(X), the first point visible from X is called firstChild(X).

- lastChild: Same as firstChild, the last point visible from X is called lastChild(X).

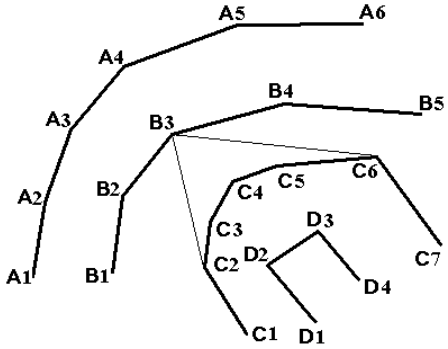


Figure 3- a part of NCHs named alphabetically

For example for point B_3 the points as firstChild and lastChild are shown in Figure 3 (C_2 and C_6).

Important events that may change the outer convex hull include: moving a point, p, from convex hull *i* to convex hull *i+1* or *i-1* (*goOut(p) and goIn(p)*).

List of possible events for a point such as B_3 is as follows:

1- Reaching the border of the region and reflecting the direction of motion. We call this event changeDir(B_3)

2- Moving B_3 toward the inner convex hull (C). We call this event *goIn(B_3)*.

3- Moving B_3 toward the outer convex hull (A). We call this event *goOut(B_3)*.

4- Exiting C_2 from visibility region of B_3 (C_3 intersects line segment B_3C_2). We call this event notFirstChild(B_3).

5- Entering C1 into visibility region of B_3. We call this event beFirstChild(B_3).

6- Exiting C_6 from visibility region of B_3 (C5 intersects line segment B_3C_6). We call this event notLastChild(B_3).

7- Moving C_7 into visibility region of B_3. We call this event beLastChild(B_3).

The last four events ensures having correct firstChild and lastChild for each point at event times.

At each scheduled event we do necessary modifications to NCHs. These changes are all local. For example in case of Figure 3, moving B_3 toward convex hull C leaves B_2 and B_4 in convex hull B as neighbours (deleting B_3 form convex hull B), also inserts B_3 as a point of convex hull C between C_2 and C_6. This results entering C_3, C_4 and C_5 recursively in inner convex hulls. This (entrance to inner convex hulls

recursively) continues until the node that should enter inside has only its two Children as visible points of inner convex hull; in this case deleting it from outer convex hull and inserting it in the inner convex hull is enough. This operation is called *goIn*. In Figures 4, 5 examples of simple goOut and simple goIn are shown. Simple events only change the two adjacent convex hulls, and do not change the other internal convex hulls, whilst complex events change several nested convex hulls recursively.
All these operations are applied at each event and NCHs are updated accordingly.

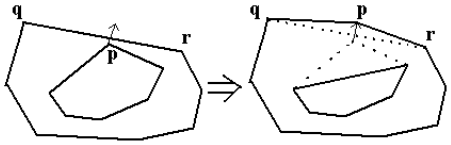


Figure 4- Simple *goOut(p)* event and changes made in the NCHs

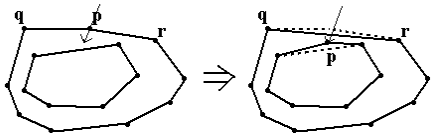


Figure 5- Simple *goIn(p)* event and changes made in the NCHs

In Table 1 we show the changes needed to do with occurrence of each event:

Event name	Certificate	Changes after occurrence
changeDir (B_3)	Crossing B_3 with the border of the region	A reflected direction of B_3 with respect to the border will be applied
goIn(B_3)	Crossing B_3 with line segment B_2B_4	B_3 will be inserted in the inner convex hull that may result in goIn(X_i) for some X and i (X∈ layers C, D, …) recursively
goOut (B_3^. firstChild)	Crossing B_3 ^.firstChild (e.g. C_2) with the line segment B_3B_3^.prior	B_3^.firstChild (e.g. C_2) will be deleted from inner convex hull and will be inserted between B_3 and B_3^.prior (e.g. B_2) that may result in goOut(X_i) for some i and X (X∈ layers C, D, …) recursively
notFirstChild(B_3)	Crossing the point B_3^.firstChild ^.next (e.g. C_3) with line segment B_3B_3^.firstChild (e.g. B_3C_2)	B_3^.firstChild^.next (e.g. C_3) will become the new B_3^.firstChild
beFirstChild(B_3)	Crossing the point B_3^.firstChild ^.prior (e.g. C_1) with extension of line segment B_3B_3^.firstChild (e.g. B_3C_2)	B_3^.firstChild^.prior (e.g. C_1) will become the new B_3^.firstChild
notLastChild(B_3)	Crossing the point B_3^.lastChild ^.prior (e.g. C_5) with line segment B_3B_3^.lastChild (e.g. B_3C_6)	B_3^.lastChild^.prior (e.g. C_5) will become the new B_3^.lastChild
beLastChild(B_3)	Crossing point B_3^.lastChild ^.next (e.g. C_7) with extension of line segment B_3B_3^.lastChild (e.g. B_3C_6)	B_3^.lastChild^.next (e.g. C_7)will become the new B_3^.lastChild

Table 1- List of possible events

In the next section we present the algorithm working with the data structure to maintain the convex hull.

4- KINETIC CONVEX HULL MAINTENANCE ALGORITHM

The high-level pseudo code for this algorithm is given below. The code simulates NCHs maintenance and animates moving NCHs. Note that all links are implemented by pointers; each point has a pointer to next and previous points, and pointers to its firstChild and lastChild. With occurrence of each event these pointers are modified according to type of event.

```
Algorithm Kinetic_Convex_Hull:
        Input:    pointsArray[1..n]
                      // Array of coordinates of input points
                  simTime
                      // The simulation time
        Output:  moving convex hull of points
Var
    S: NCHs;
    E: linked list of Events sorted by event time;
    t, occurrenceTime: Time
{
    S ← createNCHs(pointsArray);
    S ← linkChilds(S);
        // : for all points finds firstChild and lastChild points
        // (if any)
    E ← scheduleAllEvents(S);
        // For almost all points there will be 7 events
        // according to definition in section 3-1.
        // Exceptions are the points on innermost convex hull
    t ← getTime();
```

```
    simTime ← simTime + t;
    occurrenceTime ← E^.occurrenceTime;
        // E^.occurrenceTime is the time of first event
    while(occurrenceTime < simTime){
        draw S till time occurrenceTime;
          // S has the same configuration during this time
        applyFirstEvent(E, S);
         // Applies the first event of list E. Applying this
         //  event may change both E and S. Because of
         // this, these two these parameters are called by
         // reference and change their values in the
         // function. The places of points of NCHs are updated
         // and their related events in E are rescheduled.
        E ← E^.nextEvent;  // pointing E to next Event.
        occurrenceTime ← E ^.occurrenceTime;
    }
}
```

5- CONCLUSIONS

Using the framework defined in [8] we proposed an efficient algorithm using a direct approach. Because of the nature of convex hull, it is difficult to localize the problem. By using nested convex hulls we showed that each point's motion may only change the status of some neighboring points, and as a result were able to eliminate many events and achieve efficiency.

6- REFERENCES

[1] 1. J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. BIT, 32:249-267, 1992.

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

[3] 3. F. P. Perparata and M. I. Shamos. Computational Geom-etry: An Introduction. Springer-Verlag, New York, NY, 1985.

[4] 4. P. K. Agarwal, O. Schwarzkopf, and M.Sharir. The overlay of lower envelopes and its applications. Discrete Comput. Geom., 15:1-13, 1996

[5] 5. J. Hershberger. Finding the upper envelope of n line seg-ments in *O(nlogn)* time. *Inform* Process. Left., 33:169-174, 1989.

[6] 6. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry, Algorithms and applications. Springer-Verlag, Berlin, 2000.

[7] 7. M. J. Atallah. Some dynamic computational geometry problems. Comput. Math. Appl., 11:1171-1181, 1985.

[8] 8. J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. SoDA, 1997.

[9] 9. J. Basch L. J. Guibas, C. D. Silverstein and L. Zhang. A Practical Evaluation of Kinetic Data Structures. 13th Symposium on Computational Geometry, 1997.

Optimal tolerancing in mechanical design using polyhedral computation tools

Komei Fukuda*
School of Computer Science
McGill University
Montreal, Canada

Jean-Philippe Petit
Laboratoire de Mécanique Appliquée
Ecole Supérieure d'Ingénieurs d'Annecy
Annecy, France

January 1, 2003

1 Introduction

Mechanical engineering is a field in which mechanisms (assemblies of parts) are designed and manufactured. This mechanism is born from a customer need and this last specifies technical functions which the final product will have to fulfil. A specification textbook is written. The concern of the designer is then to translate these functions through technological choices. A universal language is essential to define the product characteristics and must be common to each sector of its development (design, manufacture and control). This language is called dimensioning and tolerancing. Each part is represented by its nominal geometry (which is represented in CAD software) based on perfect dimensions. Moreover, these parts are manufactured, so they have defects. It is thus necessary to define acceptable limits in term of form, dimensions and position of functional features. Tolerancing is an important operation because of this one will depend the correct operation of the mechanism but also its cost (the manufacturing cost increases with the precision of tolerances values); one can then be astonished by the absence of tolerancing assistance modules in CAD software. A research team of LMécA (Laboratoire de Mécanique Appliquée) works on a model intended to be integrated in CAD system. This model is called model of the clearance domains and deviation domains [1]. These domains are polytopes built in the six dimensional Euclidean space $\mathbb{R}^6$. For tolerancing analysis, various geometrical operations (Minkowski sums, intersections...) on these polytopes are used (according to the mechanism configuration : single loop, parallel loops or open loop).

In order to perform geometrical operations on polytopes in higher dimensions, we have tested the polyhedral computation library CDDLIB [2]. CDDLIB provides two fundamental operations on convex polytopes, the vertex enumeration and the facet enumeration. More precisely, the vertex enumeration is to obtain the minimal V-representation of an H-polyhedron, and the facet enumeration is the converse. The library can be used with both floating-point arithmetic and infinite precision rational (GMP) arithmetic. Even for simple examples, the floating point computation was seen unstable. We could successfully use the GMP version of CDDLIB for several models of the clearance domains and deviation domains.

2 Clearance Domain

The first function to fulfill for a mechanism is the assemblebility of its constituting parts. These parts are connected the ones to the others by means of joints. It is necessary to consider these joints in the model by defining clearance domain associated to each joint. A joint is constituted

*The research is partially supported by an NSERC grant, Canada.

of two distinct parts and a reference frame is attached to each part. Clearance domain of the joint is a 6D polytopes (3 translations and 3 rotations) whose vertices correspond to maximum displacements of a reference frame compared to the other.

Example (Clearance domain of a cylindrical joint).

For this joint, translation and rotation along the axis x are infinite ($Tx = Rx = \infty$). There are also small displacements due to clearance ($J = D - d$) of the part (1) compared to (0): Ty, Tz, Ry and Rz

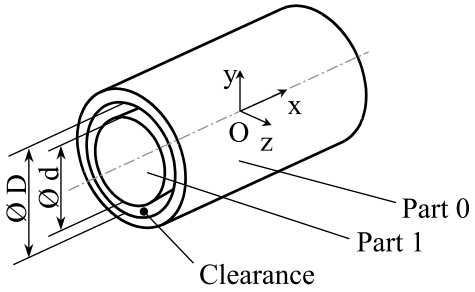


Figure 1: Cylindrical joint

By considering small displacements torsors [3] of the joint, it is possible to write a linear set of inequalities which leads on contact conditions. Those inequalities define the clearance domain of the joint A between the parts (1) and (0). The domain noted $\{J_{0A1}\}$ is unbounded in Tx and Rx directions, a 3D representation is shown in Figure 2.

This representation is a 3D cut of a 6D polytope with $Tx = Rx = Ty = 0$.

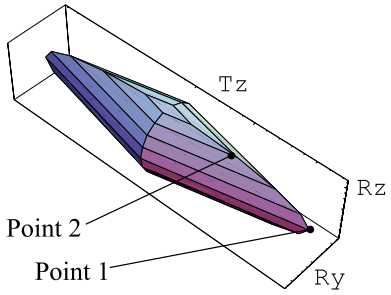


Figure 2: 3D representation of $\{J_{0A1}\}$

Point 1 means: when there is no rotation of part (1) compared to (0) in the joint (nominal position, see fig.1), maximal translation of (1) along $\vec{z}$ axis is equal to $J/2$. For the point 2, if there is a small rotation around $\vec{z}$ between the two parts, the translation will be less than $J/2$. All the joint configurations are considered through the 6D domain.

3 Deviation Domain

Tolerancing and so specifications translate designer requirements to define acceptable defect limits of functional features (surface, axis...) with standard language (see Figure 3).

The specification proposed in Figure 3 means that toleranced surface must be positioned (symbol) at a distant a with a tolerance value of t compared to the reference A. The control of such a requirement consists in checking if the manufactured surface is gap between two virtual plans (dotted zone in Figure 3) which are parallel to the reference A and distant of a t value one from

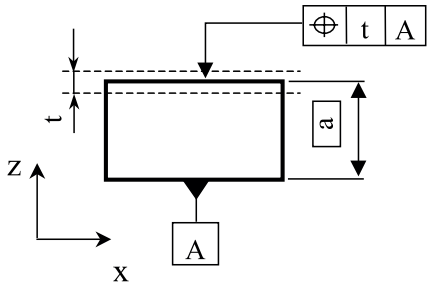


Figure 3: Specification example

the other. In the 6D configuration space (3 translations and 3 rotations), displacements of characteristic points Pi of the toleranced surface are expressed through a set of linear inequalities: $-t/2 \leq \overrightarrow{\delta Pi}.\overrightarrow{z} \leq t/2$. The 6-polytope built with this set of inequalities is the deviation domain $\{E\}$ (see Figure 4) associated to the position specification. It represents the maximum defects (displacements and angular position) of the toleranced surface.

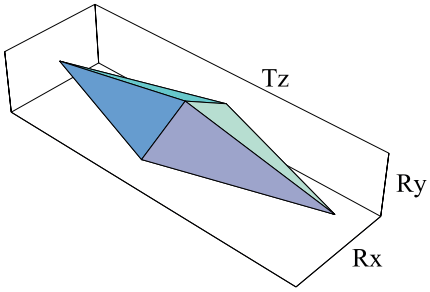


Figure 4: Deviation domain $\{E\}$ of the position specification

4 Application to a Single Loop Mechanism

The mechanism is composed of three parts (see Figure 5):

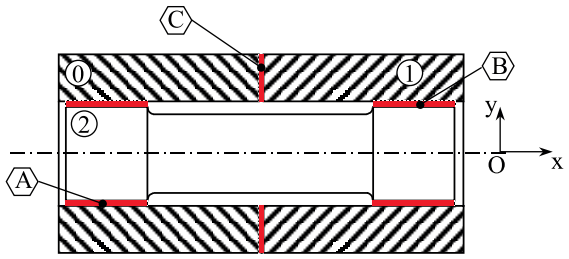


Figure 5: Assembly drawing

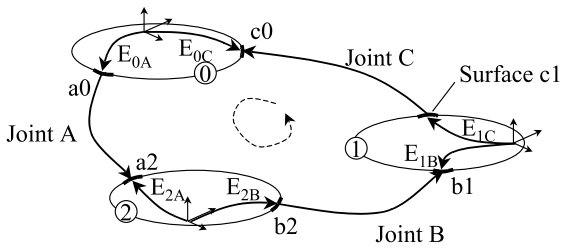


Figure 6: Mechanism diagram

The three joints (A, B and C) can be translated by three clearance domains built in the same point O. Surfaces of each part constituting joint are functional so supposed to be toleranced (engineering drawing is not showed here) and the deviation domains are built in point O. Mechanism theory [4] and configuration of mechanism diagram allow to write the following equation:

$$E_{0C} + J_{0C1} - E_{1C} + E_{1B} + J_{1B2} - E_{2B} + E_{2A} + J_{2A0} - E_{0A} = 0 \tag{4.1}$$

Chosen tolerancing must verify assemblebility and interchangeability of any part belonging to a batch. Fixing $\{J\} = \{J_{0C1}\} + \{J_{1B2}\} + \{J_{2A0}\}$ and $\{E\} = \{E_{0C}\} - \{E_{1C}\} + \{E_{1B}\} - \{E_{2B}\} + \{E_{2A}\} - \{E_{0A}\}$, considering equation (4.1), the two conditions above are satisfied if $\{E\} \subseteq \{J\}$. In other words, more the functional features defects are important, more clearance into mechanism joint will be necessary to correct those defects. From this condition, the procedure of tolerancing analysis is

- Building each clearance domains (6-polytopes) associated to each joint,
- Making Minkowski sum of these domains $\rightarrow \{J\}$,
- Building deviation domains associated to each specification,
- Making Minkowski sum of these domains $\rightarrow \{E\}$,
- Checking the inclusion of $\{E\}$ into $\{J\}$.

Tolerancing is optimal when $\{J\} = \{E\}$.

5 Conclusion

In this model, polytopes help the designer to validate his tolerancing choices (tolerancing analysis): checking mechanism assemblebility, respect of functional requirements. 3D cuts of graphic representations of domains can also inform the designer on his qualitative and quantitative tolerancing choices (tolerancing synthesis).

While existing codes for polyhedral computation turned out to be useful for analyzing some models of the clearance domains and deviation domains, we need further developments of polyhedral computation codes. In particular, there is no efficient codes to compute the Minkowski addition [5] of convex polytopes. In fact, this motivates one of the authors to design a new efficient algorithm for this problem which is highly parallelizable and easy to implement, see [6]. Implementing this algorithm and applying to our models, in particular for tolerancing analysis, is one of our future research projects.

References

[1] Giordano Duret. Clearance space and deviation space - application to 3D chain of dimensions and positions. *3rd CIRP*, Cachan, 1993.

[2] Komei Fukuda. *cdd, cddplus and cddlib homepage*. McGill University, Montreal, Canada, 2002. http://www.cs.mcgill.ca/~fukuda/software/cdd_home/cdd.html.

[3] Bourdet Mathieu Lartigue Ballu. The concept of small displacement torsor in metrology. Advanced mathematical tool in metrology, *Series on advances in mathematics for applied sciences*, World scientific, Vol 40, 1996.

[4] Le Borzec and Lotterie. Théorie des mécanismes. *Dunod*, 1974.

[5] Peter Gritzmann and Bernt Sturmfels. Minkowski addition of polytopes: computational complexity and applications to Gröbner bases. *SIAM J. Dics. Math.*, 6:246–269, 1993.

[6] Komei Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. Preprint, School of Computer Science, McGill University, Montreal, 2002.

Approximating the Visible Region of a Point on a Terrain *

Boaz Ben-Moshe Paz Carmi Matthew J. Katz

Department of Computer Science
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
{benmoshe,carmip,matya}@cs.bgu.ac.il

Abstract

Given a terrain T and a point p on it, we wish to compute the region that is visible from p. We present a generic radar-like algorithm for computing an approximation of this region. The algorithm *extrapolates* the visible region between two consecutive rays (from p) whenever the rays are *close enough*; that is, whenever the difference between the sets of visible segments along the rays is below some threshold. Thus the density of the sampling by rays is sensitive to the shape of the visible region. We suggest a specific way to measure the resemblance (difference) and to extrapolate the visible region between two consecutive rays. We report on preliminary experimental results.

1 Introduction

Let T be a triangulation representing a terrain (i.e., there is a height (z-coordinate) associated with each triangle vertex). We are interested in the following well known problem. Given a point p on (or above) T, compute the region R_p of T that is visible from p. A point q on T is visible from p if and only if the line segment $\overline{pq}$ lies above T (in the weak sense). Thus R_p consists of all points on T that are visible from p. The problem of computing the visible region of a point arises as a subproblem in numerous applications (see, e.g., [7, 9, 11]), and, as such, has been studied extensively [2, 3, 4, 5, 7]. For example, the coverage area of an antenna for which line of sight is required may be determined by clipping the region that is visible from the tip of the antenna with an appropriate disk centered at the antenna.

Since the combinatorial complexity of R_p might be $\Omega(n^2)$ [3, 6], where n is the number of triangles in T, it is desirable to also have fast approximation algorithms, i.e., algorithms that compute an approximation of R_p. Moreover, a good approximation of the visible region is often sufficient, especially when the triangulation itself is only a rough approximation of the underlying terrain. Note that in this paper we are assuming that the terrain representation (i.e., the triangulation T) is fixed and cannot be modified. Simplifying the triangulation can of course lead to faster running times of any algorithm for computing the visible region. This approach was studied in a previous paper [1]. See, e.g., [8] for more information on terrain simplification.

We present a generic radar-like algorithm for computing an approximation of R_p. The algorithm computes the visible segments along two rays ρ_1, ρ_2 emanating from p, where the angle between the rays is not too big. It then has to decide whether the two sets of visible segments (one per ray) are *close enough* so that it can *extrapolate* the visible region of p within the wedge defined by ρ_1 and ρ_2, or whether an intermediate ray is needed. In the latter case the algorithm will now consider the smaller wedge defined by ρ_1 and the intermediate ray. Thus a nice property of the algorithm is that the density of the sample rays varies and depends on the shape of R_p.

*Research by Ben-Moshe and Katz is partially supported by grant no. 2000160 from the U.S.-Israel Binational Science Foundation, and by the MAGNET program of the Israel Ministry of Industry and Trade (LSRT consortium). Research by Carmi is partially supported by a Kreitman Foundation doctoral fellowship.

In order to use this generic algorithm one must provide (i) a measure of resemblance for two sets of visible segments, where each set consists of the visible segments along some ray from p, and (ii) an algorithm to extrapolate the visible region between two rays whose corresponding sets were found similar enough. In Section 2 we describe in more detail the generic algorithm and provide the missing ingredients.

In Section 3 we suggest a natural way to measure the error associated with an approximation of R_p. Using this error measure, we compare between our algorithm and the corresponding fixed-angle version with the same number of sample rays. According to these experiments our algorithm is much better in situations of "under sampling," where the number of sample rays is small. We are currently comparing our algorithm (and several variants of it) with other (known) algorithms for approximating the visible region. A report on these experiments will be included in the full version of this paper.

2 The Algorithm

In this section we first present our radar-like generic algorithm. Next we describe the measure of resemblance and the extrapolation algorithm that we devised, and that are needed in order to transform the generic algorithm into an actual algorithm.

The generic algorithm is presented in the frame below. The basic operation that is used is the cross-section operation, denoted $\mathit{cross\text{-}section}(T, p, \theta)$, which computes the visible segments along the ray emanating from p and forming an angle θ with the positive x-axis. Roughly speaking, the generic algorithm sweeps the terrain T counter clockwise with a ray ρ emanating from p, performing the cross-section operation whenever the pattern of visible segments on ρ is about to change significantly with respect to the pattern that was found by the previous call to cross-section. The algorithm then extrapolates, for each pair of consecutive patterns, the visible region of p within the wedge defined by the corresponding locations of ρ.

Given a triangulation T representing a terrain (i.e., with heights associated with the triangle vertices) and a view point p on or above T:
$\theta \leftarrow 0$.
$\alpha \leftarrow$ some constant angle, say, $\pi/45$.
$S_1 \leftarrow \mathit{cross\text{-}section}(T, p, \theta)$.
$S_2 \leftarrow \mathit{cross\text{-}section}(T, p, \theta + \alpha)$.
while $(\theta < 360)$
    if (S_1 is **close enough** to S_2)
        extrapolate(S_1, S_2);
        $\theta \leftarrow S_2.\mathit{angle}$;
        $S_1 \leftarrow S_2$;
        $S_2 \leftarrow \mathit{cross\text{-}section}(T, p, \mathit{min}(\theta + \alpha, 360))$;
    else
        $\mu \leftarrow (S_1.\mathit{angle} + S_2.\mathit{angle})/2$;
        $S_2 \leftarrow \mathit{cross\text{-}section}(T, p, \mu)$;

In order to obtain an actual algorithm we must provide precise definitions of ***close enough*** and ***extrapolate***.

Close enough: A threshold function that checks whether two patterns S_1, S_2 are similar, where each of the patterns corresponds to the set of visible segments on some ray from p. There are of course many ways to define ***close enough***. We chose the following definition. In practice, the rotating ray is actually a rotating segment of an appropriate length. Let l denote this length. We refer to l as the *range of sight*. Now rotate the ray containing S_2 clockwise until it coincides with the ray containing S_1. See Figure 1 (a). Next compute the length of the **XOR** of S_1 and S_2, that is, the total length covered by only one of the sets S_1, S_2. This length is then divided by l. Denote by v the value that was computed, and let δ be the angle between S_1 and S_2. If $\delta \cdot v \leq C$, where C is some constant, then return TRUE else return FALSE. The role of δ in the above formula is to force ***close enough*** to return TRUE when the angle between the rays is small, even if the patterns that are being compared

differ significantly.

Extrapolate: Given two patterns S_1, S_2 which are ***close enough***, we need to compute an approximation of the portion of the visible region of p that is contained in the corresponding wedge. We do this as follows. Consider Figure 1 (b). For each 'event point' (i.e., start or end point of a visible segment) on one of the two horizontal rays, draw a vertical segment that connects it with the corresponding point on the other ray. For each rectangle that is obtained color it as follows, where grey means visible and black means invisible. If the horizontal edges of a rectangle are either both visible from p or both invisible from p, then, if both are visible, color it grey, else color it black. If, however, one of the horizontal edges is visible and the other is invisible, divide the rectangle into four pieces by drawing the two diagonals. The color of the upper and lower pieces is determined by the color of the upper and lower edges, respectively, and the color of the left and right pieces is determined by the color of the rectangles on the left and on the right, respectively.

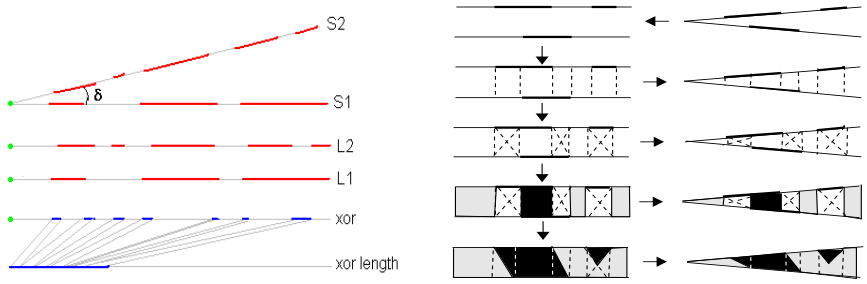


Figure 1: Grey marks visible and black marks invisible. (a) The ***close enough*** threshold function: δ times the relative length of the **XOR** of S_1 and S_2. (b) The***extrapolate*** function.

3 Experimental Results

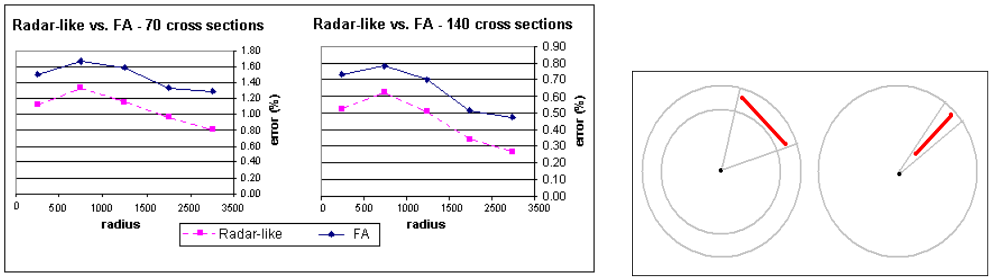
In our experiments we use the following natural error measure. Let R'_p be an approximation of R_p obtained by some approximation algorithm, where R_p is the region visible from p. Then the error associated with R'_p is the area of the **XOR** of R'_p and R_p, divided by the area of the disk of radius l, where l is the range of sight that is in use. See Figure 2.

Figure 2: Left: the exact region R_p; Middle: the approximate region R'_p computed by our algorithm; Right: $\mathbf{XOR}(R'_p, R_p)$.

We compared between our radar-like algorithm and the corresponding fixed-angle version. That is, we ran our algorithm, with several values of α, on a collection of terrains, view points, and ranges of sight. For each application of our algorithm, we also ran the fixed-angle version with angle $360/n$, where n is the number of sample rays (i.e., cross-section operations) that were used in this application. We then computed the errors for the two regions that were obtained. Figure 3(a) shows some typical results. From these results it is clear that our

algorithm is better is situations of "under sampling," where the number of sample rays is small. Both algorithms and the error computation were implemented in Java. (The error computation is actually a grid-based approximation of the error measure defined above.)

We are currently comparing our algorithm (and several variants of it) with other (known) algorithms for approximating the visible region, including the z-buffer algorithm [10] and an algorithm (see Figure 3(b)), that is in some sense orthogonal to our algorithm, that uses circles of increasing radii instead of cross-sections [5]. A report on these experiments will be included in the full version of this paper.



(a) (b)

Figure 3: (a) Our algorithm is more accurate than the corresponding fixed-angle version. (b) Circles of increasing radii approach vs. Radar-like approach.

Acknowledgment. The authors wish to thank Ofir Ganani and Maor Mishkin who helped implementing the radar-like algorithm.

References

[1] B. BenMoshe, M.J. Katz, J.S.B. Mitchell and Y. Nir. Visibility preserving terrain simplification. *Proc. 18th ACM Sympos. Comput. Geom.* 303–311, 2002.

[2] D. Cohen-Or and A. Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum* 14(3):171–179, 1995.

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

[4] L. De Floriani and P. Magillo. Visibility algorithms on triangulated digital terrain model. *International Journal of GIS* 8(1):13–41, 1994.

[5] L. De Floriani and P. Magillo. Representing the visibility structure of a polyhedral terrain through a horizon map. *International Journal of GIS* 10:541–562, 1996.

[6] F. Devai. Quadratic bounds for hidden line elimination. *Proc. 2nd ACM Sympos. Comput. Geom.* 269–275, 1986.

[7] R. Franklin, C.K. Ray and S. Mehta. Geometric algorithms for siting of air defense missile batteries. *Technical Report Contract No. DAAL03-86-D-0001*, 1994.

[8] P. S. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[9] M.F. Goodchild and J. Lee. Coverage problems and visibility regions on topographic surfaces. *Annals of Operation Research* 18:175–186, 1989.

[10] N. Greene, M. Kass and G. Miller. Hierarchical z-buffer visibility. *Computer Graphics Proc. Annu. Conference Series* 273–278, 1993.

[11] A.J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Trans. Visualizat. Compt. Graph* 4(1):82–93, 1998.

Pheromone-guided Dispersion for Swarms of Robots

Tien-Ruey Hsiang* Marcelo Sztainberg*

Abstract

We propose and analyze an algorithm for dispersing a swarm of robots in an unknown environment R. We use simple robots with the ability to leave a pheromone behind. Robots can get local information with their sensors but cannot communicate with other robots. The primary objective is to minimize the *makespan*, that is, the time to *fill* the entire region. We achieve a competitive ratio of $O(\log(k+1))$ where k is the number of doorways.

1 Introduction

Multiagent robotics has been an active field in the recent years and there has been many works on distributed control and coordination of a set of autonomous robots. Principe et al. [6, 16] and Suzuki et al. [5, 19, 20] have studied pattern formation in distributed autonomous robotics under various models of robots with minimal capabilities. The related flocking problem, which requires that a set of robots follow a leader while maintaining a formation, has been studied in several recent papers; see, e.g., [1, 2, 10] Wagner et al. [3, 21, 22, 23] developed multi-robot algorithms, inspired by ant behaviors, for searching and covering. Payton et al. [15, 14] propose the notion of "pheromone robotics" for world-embedded computation.

A natural problem that arises in the study of "swarm robotics" is how to obtain a quick dispersal and filling of the environment while maintaining the connectivity of the robot swarm. That is, devise algorithms to reposition robot swarms in an uknown domain such that every point of the domain is seen by some robots and for any given pair of robots we can establish a visibility chain in which consecutive robots can see each other.

Motivations for this behavior can be found on a diverse spectrum of applications for different domains: space exploration, medicine, military, and industry, just to name a few. Common applications include exploration and map extraction of an unknown domain, mine sweeping, and guarding.

A big proportion of previously developed dispersion algorithms rely on greedy strategies such as *go-for-free space* [13], where robots move to fill unoccupied space; *artificial physics* [18] strategies, where neighboring robots exert "forces" on each other: repulsion forces if the robots are closer than the target separation, and attraction forces if the neighboring robots are farther away (and the swarm is in danger of becoming disconnected); and *potential fields* [11, 17]. Almost all of them require communication, albeit local, within pairs of robots

Our goal is to develop dispersion algorithm on discrete environments. A discrete environment is composed of *squares* or *pixels* that form a connected subset of the integer grid. There is at most one robot per pixel and robots move horizontally or vertically at unit speed. Robots enter domain R through $k \geq 1$ *door pixels*, each of which acts as an infinite source of robots.

Our robots can be implemented with $O(1)$-size memory and $O(1)$-size sensor range. There is no direct communication between robots, instead, they leave pheromones as traces for others to follow. The robots are decentralized and move only according to local information.

*Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794-3600, email: {trhsiang, mos}@ams.sunysb.edu

2 Pheromone-guided Dispersion

As the robots move through the domain, they leave a pheromone trail behind. At each step a robot surveys its local neighborhood (Figure 1) and decides its next move according to the positioning of the pheromones, other robots, and obstacles. This process does not require direct communication between robots. For the case of multiple doors, the pheromones are differentiated by *teams* corresponding to each particular doorway.

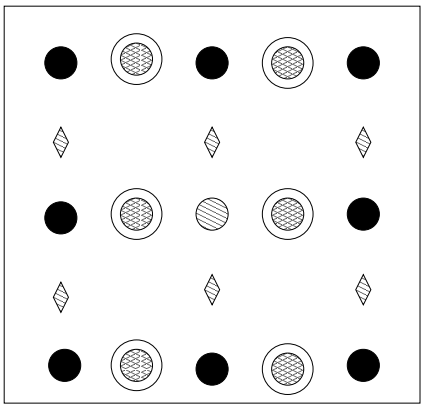


Figure 1: Local information available to a robot (at the center in gray) at a particular step of the simulation. Diamonds represent pheromones, double circled points represent stopped robots, and black points are currently moving robots

The main goal for a robot is to find a spot in the domain where it can *stop*, preferrably at the shortest possible amount of traveled distance. This stopping spot can only be found at the left or the right of the direction the robot is moving, so not to interrupt the flow of its line. To keep the robots advancing through the domain and to avoid blocking paths, a spot is available for stopping only if its surrounding spots do not hold a stopped robot. (Figure 2)

If a robot does not have spot available to stop it should try to keep moving following these set of priority rules:

- Follow the path is was moving on. Otherwise...
- Try to make a turn that does not conflict with other teams of robots. Otherwise...
- Wait for a constant number of iterations. If there is no possibility of continuing it should stop.

The pheromones that robot utilize to make trails contain only the *team-tag* information. Each pheromone has a constant size lifetime after which it *dissapears*. Since each moving robot has another robot following behind at a close distance pheromones get replaced by newer ones as the trail keeps moving.

Summary of Results With a simpler heuristic and a lack of communication our algorithm obtained similar results as those expressed by Hsiang et al [12]. We prove that our algorithms have optimal competitive ratio of $O(\log(k+1))$ where k is the number of doorways. By having the robots searching for the closest stopping place we ensure a more efficient use of individual resources (power supply), while achieving a complete stop for the run with no need of general communication.

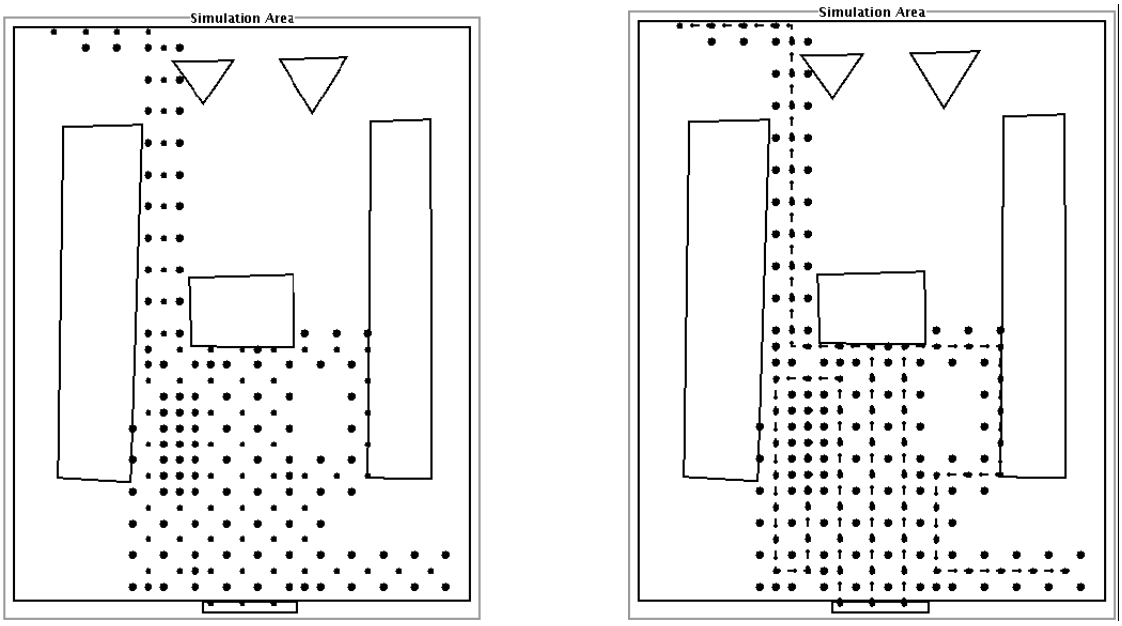


Figure 2: The figure on the left illustrates an instance of the simulation. Robots are entering the domain through the door located at the bottom center of the figure. The figure on the right expands the original at the left with the display of the pheromones that guide the robots.

3 Future Work

- We modified our heuristic, requiring for a robot to stop only when it does not have any other place to move to.
- We are implementing a new strategy that involves *random branching*, in which teams coming from the same doorway, split and branch randomly along the march.
- Similar heuristics are being implemented for continuous environments. For those environments the step size, and the steering angle of a robot are subject to measurement errors.

References

[1] T. Balch and R. C. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[2] O. B. Bayazit, J.-M. Lien, and N. M. Amato. Better flocking behaviors using rule-based roadmaps. In *Proc. Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002.

[3] A. M. Bruckstein, C. L. Mallows, and I. A. Wagner. Probabilistic pursuits on the grid. *American Mathematical Monthly*, 104(4):323–343, 1997.

[4] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *Journal of the ACM*, 45(2):215–245, Mar. 1998.

[5] A. Dumitrescu, I. Suzuki, and M. Yamashita. High speed formations of reconfigurable modular robotic systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, pages 123–128, 2002.

[6] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *Proc. IEEE Intelligent Vehicle Symposium (IV 2000)*, pages 480–485, 2000.

[7] D. Gage. Many-robot systems. http://www.spawar.navy.mil/robots/research/manyrobo/manyrobo.html.

[8] D. Gage. Sensor abstractions to support many-robot systems. In *Proceedings of SPIE Mobile Robots VII, Boston MA, 18-20 November 1992, Volume 1831*, pages 235–246, 1992.

[9] D. Gage. An evolutionary strategy for achieving autonomous navigation. In *SPIE Proc. 3525: Mobile Robots XIII, Boston MA, November*, 1998.

[10] V. Gervasi and G. Prencipe. Flocking by a set of autonomous mobile robots. Technical Report TR-01-24, University of Pisa, Dipartimento di Informatica, Oct. 2001.

[11] A. Howard, M. J. Mataric, and G. S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed scalable solution to the area coverage problem. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Proc. 6th International Symposium on Distributed Autonomous Robotic Systems*, page to appear, Fukuoka, Japan, 2002. Springer-Verlag.

[12] T.-R. Hsiang, E. M. Arkin, M. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. *Fifth International Workshop on Algorithmic Foundations of Robotics*, December 15-17 2002.

[13] N. Jovanovic, T.-R. Hsiang, and M. Sztainberg. Experimental comparison of dispersion protocols for robot swarms. Technical report, Stony Brook University, 2002.

[14] D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone robotics. *Autonomous Robots*, pages 319–324, 2001.

[15] D. Payton, R. Estkowski, and M. Howard. Progress in pheromone robotics. *Proc. 7th International Conference on Intelligent Autonomous Systems*, 2002.

[16] G. Prencipe. *Distributed Coordination of a Set of Autonomous Mobile Robots.* PhD thesis, Dipartimento di Informatica, Università degli Studi di Pisa, 2002.

[17] J. H. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems*, 27(3):171–194, 1999.

[18] W. Spears and D. Gordon. Using artificial physics to control agents. In *In IEEE International Conference on Information, Intelligence, and Systems.*, 1999.

[19] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotic Systems 13, 3*, pages 127–139, 1996.

[20] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[21] I. Wagner and A. Bruckstein. Cooperative cleaners: a study in ant robotics. Technical Report CIS9512, Technion, 1995.

[22] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15(5):918–933, 1999.

[23] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Efficiently searching a graph by a smell-oriented vertex process. *Annals of Mathematics and Artificial Intelligence*, 24(1-4):211–223, 1998.

[24] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Mac vs. pc - determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. Technical Report CIS9814, Technion, 1998.

Orthogonal Segment Stabbing*

Matthew J. Katz[1] Joseph S. B. Mitchell[2] Yuval Nir[1]

[1]Department of Computer Science
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
{matya,yoval}@cs.bgu.ac.il
[2]Department of Applied Mathematics and Statistics
State University of New York, Stony Brook, NY 11794-3600
jsbm@ams.sunysb.edu

1 Introduction

We consider various stabbing problems for orthogonal line segments in the plane. Let H, V be sets of horizontal and vertical segments. In the first problem that we consider, one has to find a subset S of $H \cup V$ of minimal size, such that for each segment $s \in H \cup V$, either $s \in S$ or s is stabbed by some segment s' in S. We refer to this problem as the Orthogonal Segment Dominating Set problem (OSDS). This problem is closely related to a guarding problem posed by Frank Hoffmann. In Hoffmann's problem, the sets H and V represent a connected network of horizontal and vertical streets, and one has to find a patrol path of minimal length, such that every street point (i.e., a point on an input segment) is seen from some point on the path. In another problem that we consider, one has to find a minimal subset of *vertical* segments that stab all horizontal segments (assuming such a set exists). We refer to this problem as the Orthogonal Segment Vertex Cover problem (OSVC).

In Section 2 we show that OSDS and OSVC are NP-complete, by reducing them to vertex dominating set for planar bipartite graphs and to vertex cover for planar graphs, respectively. Both reductions are based on a representation theorem (Theorem 2.1) due to Ben-Arroyo Hartman et al. [1] and to de Fraysseix et al. [4], and on the corresponding algorithmic result of de Fraysseix et al. [5]. This theorem states that any planar bipartite graph $G = (A, B; E)$ can be represented by a collection of $|A|$ horizontal segments and $|B|$ vertical segments, such that a horizontal segment and a vertical segment corresponding to vertices a and b stab each other if and only if $(a, b) \in E$.

We obtain two variants of OSVC by extending the segments in V to downwards-directed rays, or by extending the segments in H to rightwards-directed rays. In other words, let H be a set of horizontal segments and let R be a set of downwards-directed rays. In the *stabbing segments with rays problem* one has to find a minimal subset of R that stabs all segments in H, and in the *stabbing rays with segments problem* one has to find a minimal subset of H that stabs all rays in R. We present polynomial-time solutions to both these problems that are based on dynamic programming. In this extended abstract we only sketch the solution to the former problem (Section 3), and completely omit the solution to the latter problem, which is significantly more complicated.

Both these problems have an equivalent formulation involving a set T of (possibly intersecting) axis-parallel rectangles and a set P of points. Given such sets, the corresponding piercing problem (i.e., compute a minimal subset P' of P, such that each rectangle is pierced by a point in P') and

*Research by M.K. and J.M. is partially supported by grant no. 2000160 from the U.S.-Israel Binational Science Foundation. Research by M.K. and Y.N. is also partially supported by the MAGNET program of the Israel Ministry of Industry and Trade (LSRT consortium). Research by J.M. is also partially supported by HRL Laboratories, NASA Ames Research, National Science Foundation, Northrop-Grumman, Sandia National Labs, Sun Microsystems.

the corresponding covering problem (i.e., compute a minimal subset T' of T, such that each point is covered by a rectangle in T') are known to be NP-complete (see [3]).

We restrict these problems by adding the assumptions that all rectangles in T are "hanging" from a mutual line. With this assumption, the piercing problem becomes the stabbing segments with rays problem, and the covering problem becomes the stabbing rays with segments problem. (For each point $p \in P$ draw a downwards-directed ray emanating from p, and for each rectangle $t \in T$ keep its bottom edge only).

2 Stabbing Segments with Segments

In this section we prove that OSVC and OSDS are NP-complete. Our proofs are based on the following representation theorem due to Ben-Arroyo Hartman et al. [1] and to de Fraysseix et al. [4], and on the corresponding algorithmic result of de Fraysseix et al. [5].

Theorem 2.1 *[1, 4] Any planar bipartite graph $G = (A, B; E)$ can be represented by a set $I(G)$ of $|A|$ disjoint horizontal segments and $|B|$ disjoint vertical segments, such that two segments a and b stab each other if and only if $(a, b) \in E$. The set $I(G)$ is said to be a* grid representation *of G.*

Theorem 2.2 *[5] Let $G = (A, B; E)$ be a planar bipartite graph. One can compute a grid representation $I(G)$ of G in $O(|A| \cup |B|)$ time.*

2.1 OSVC is NP-complete

Let H be a set of horizontal segments and let V be a set of vertical segments. The Orthogonal Segment Vertex Cover problem (OSVC) asks for a minimal subset of V that stabs all segments in H.

Theorem 2.3 *OSVC is NP-Complete.*

Proof: We prove that OSVC is NP-complete by reducing it to Minimum Vertex Cover for planar graphs, which is known to be NP-complete [7]. Let $G = (V, E)$ be a planar graph. By placing a new vertex b_e in the middle of each edge e of E, we obtain a planar bipartite graph $G' = (A, B; F)$, where $A = V$, B corresponds to E, and there is an arc between a and b if and only if a is adjacent to the edge of G corresponding to b. It follows from the construction that G' is planar and bipartite.

Next we compute a grid representation $I(G') = V \cup H$ of G', using the linear-time algorithm of [5]. (The existence of such a representation follows from Theorem 2.1 above.) This completes the reduction, since a minimum vertex cover for G becomes a minimum subset of A that dominates all vertices in B, which in turn becomes a solution to OSVC (i.e., a minimum subset of V that stabs all segments in H). □

2.2 OSDS is NP-complete

Let H be a set of horizontal segments and let V be a set of vertical segments. The Orthogonal Segment Dominating Set problem (OSDS) asks for a minimal subset of $S = H \cup V$, such that for each segment $s \in H \cup V$, either $s \in S$ or s is stabbed by some segment s' in S.

Theorem 2.4 *OSDS is NP-Complete.*

Proof: We prove that OSDS is NP-complete by reducing it to Minimum Dominating Set for planar bipartite graphs. Let $G = (A, B; E)$ be a planar bipartite graph and let $I(G) = V \cup H$ be a grid representation of G. (By Theorem 2.1 and Theorem 2.2, $I(G)$ exists and can be computed in linear time.) Assuming that Minimum Dominating Set for planar bipartite graphs is NP-complete

we are done, since a minimum dominating set for G becomes a solution to $OSDS$. However, our search for a proof of the NP-completeness of Minimum Dominating Set for planar bipartite graphs was unsuccessful, so we include below a simple proof of this claim. This proof appears implicitly in a paper by Kariv and Hakimi [2]; it is based on the NP-completeness of Minimum Vertex Cover for planar graphs [7].

We cut each edge (u, v) of the planar graph G and insert a square a,x,b,y as shown in Figure 1. Clearly, the graph G' that is obtained remains planar. It is also bipartite. Moreover, a minimum vertex cover for G can easily be transformed to a minimum dominating set for G', by adding, for each edge (u, v) of G, either the vertex a or the vertex b. If u is not in the minimum vertex cover for G, we add a, if v is not in the minimum vertex cover, we add b, and otherwise we pick one of the two arbitrarily. □

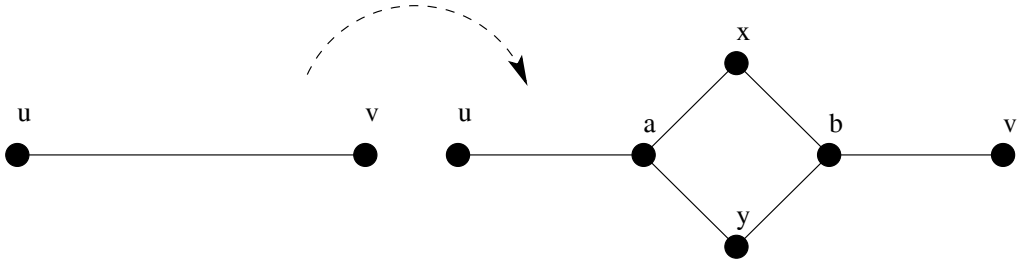


Figure 1: Construction for proof of Theorem 2.4.

3 Stabbing Segments with Rays

Let S be a set of horizontal segments and let R be a set of downwards-directed rays. For a ray $r \in R$, let $x(r)$ and $y(r)$ denote the x and y values, respectively, of r's origin. For a segment $s \in S$ let $h(s)$ denote its height, and let $l(s)$ and $r(s)$ denote the x values of its left and right endpoints, respectively. A ray is *maximal* if it is above the highest segment.

In the Stabbing Segments with Rays problem (SSR) one has to find a minimal subset $R' \subseteq R$, such that, for each segment $s \in S$, there exists a ray $r \in R'$ that stabs it.

Lemma 3.1 *Any solution consists of a maximal ray and solutions for two disjoint subproblems.*

Proof: Let R^* be a solution for $< S, R >$ and let $r \in R^*$ be a ray that stabs a segment of maximum height in S. (r is by definition a maximal ray.) Any segment that is not stabbed by r is either completely to the left or completely to the right of r, and any ray in $R \setminus \{r\}$ is either to the left or to the right of r. Therefore if R_l, R_r are solutions for the resulting left and right subproblems, respectively, then $\{r\} \cup R_l \cup R_r$ is a solution for $< S, R >$. □

We present a top-down recursive algorithm for computing a solution for SSR. By employing dynamic programming a bottom-up non-recursive algorithm can be obtained, that is more efficient in terms of space.

A subproblem $< Left, Height, Right >$ contains all segments and rays of the problem $< S, R >$ that are contained in the interior of the corresponding (infinite) vertical slab, and whose height is less or equal to $Height$. According to some simple observations that are omitted from this version, we may assume that the problem $< S, R >$ is given by $< 0, |S| + |R|, 2|S| + |R| + 1 >$, i.e., all endpoints are unique integers in the range $[1, 2|S| + |R|]$ for the x-values, and in the range $[1, |S| + |R|]$ for the y-values.

The algorithm examines the highest ray r within the subproblem $< Left, Height, Right >$. If r is not maximal (within the subproblem), then the subproblem has no solution. Else, the

algorithm compares between the sizes of the subsets of R obtained by either including r or not including r (see Figure 2). If r is included, then the solution obtained is the union of $\{r\}$ and the solutions for the subproblems $< Left, Height - 1, x(r) >$ and $< x(r), Height - 1, Right >$ (that do not contain segments that are stabbed by r). And if r is not included, then the solution for the subproblem is $< Left, Height - 1, Right >$. We omit the analysis of the algorithm from this version, and only mention that its time complexity is $O(|S|^3)$.

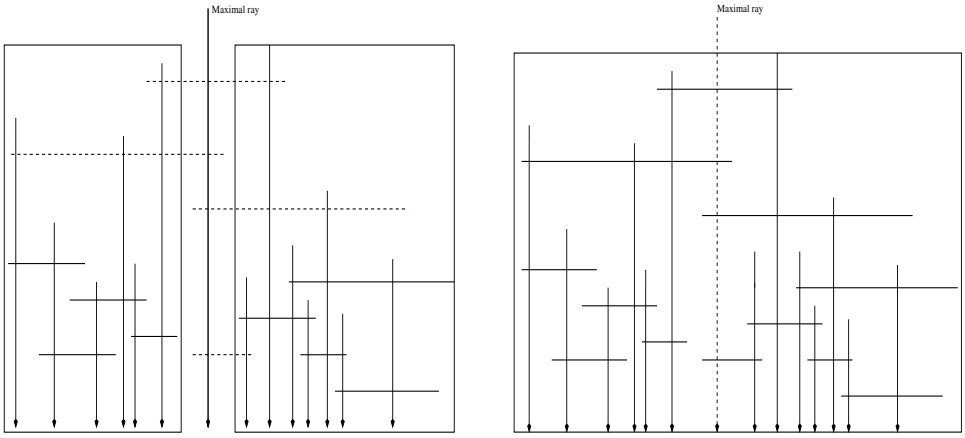


Figure 2: Subproblems resulting from (a) choosing the maximal ray r and (b) not choosing r.

Acknowledgement

We wish to thank Arie Tamir for helpful discussions.

References

[1] I. Ben-Arroyo Hartman, I. Newman and R. Ziv, *On grid intersection graphs*, Discrete Mathematics 87, 1991, 41–52.

[2] O. Kariv and L. Hakimi, *Algorithmic approach to network location problems*, SIAM J. Applied Mathematics 37, 1979, 513–538.

[3] R. J. Fowler, M. S. Paterson and, S. L. Tanimoto, *Optimal packing and covering in the plane are NP-complete*, Information Processing Letters, 12(3), 1981, 133–137.

[4] H. de Fraysseix, P. O. de Mendez, and J. Pach *Representation of planar graphs by segments* Intuitive Geometry, Coll. Math. Soc. J. Bolyai 63, 1991, 109–117.

[5] H. de Fraysseix, P. O. de Mendez, and J. Pach *A left-first search algorithm for planar graphs* Discrete Computational Geometry, 13, 1995, 459–468.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and co., New York.

[7] M. R. Garey and D. S. Johnson, *Some simplified NP-complete graph problems*, Theoretical Comp. Sci., 1, 1976, 237–267.

[8] R. Hassin and N. Megiddo, *Approximation algorithms for hitting objects by straight lines*, Discrete Applied Mathematics, 30, 1991, 29–42.

Alternating Paths along Orthogonal Segments

Csaba D. Tóth[1]
toth@cs.ucsb.edu

Abstract
It was shown recently that in the segment endpoint visibility graph Vis(S) of any set S of n disjoint line segments in the plane, there is an alternating path of length $\Theta(\log n)$, and this bound best possible apart from a constant factor. This talk focuses on the variant of the problem where S is a set of n disjoint *axis-parallel* line segments, and shows that the length of a longest alternating path in the worst case is $\Omega(\sqrt{n/2})$ and $O(n/2+2)$.

1 Introduction

Given a set S of disjoint line segments in the plane, an *alternating paths* is a simple polygonal path $p = (v_1\, v_2, \ldots, v_k)$ such that $v_{2i-1}v_{2i} \in S$, $i = 1, \ldots, \lfloor k/2 \rfloor$ and $v_{2i}v_{2i+1}$ does not cross any segment of S for $i = 1, \ldots, \lfloor (k-1)/2 \rfloor$.

It is known that there are sets of disjoint segments that do not admit an alternating Hamiltonian path. Hoffmann and Töth [3] proved recently, answering a question of Bose [1, 6], that for any set S of n disjoint line segments in the plane, there is an alternating path running through at least $\lceil \log_2(n+2) \rceil - 1$ segments of S, and this bound is best possible apart from a constant factor.

The $O(\log n)$ upper bound construction [6, 3] is a set S of line segments arranged so that every segment $s \in S$ has two endpoints on the convex hull $\mathrm{conv}(\bigcup S)$, and therefore any alternating path containing segments from both sides of s should go through s as well. In that construction n segments have $\Omega(n)$ different orientations. If all segments are axis-parallel, we can prove a better lower bound:

Theorem 1 *For any set S of n disjoint axis-parallel segments in the plane, there is an alternating path running through $\sqrt{n/2}$ segments of S.*

Restricting the general upper bound construction to axis-parallel segments, we obtain an upper bound of $O(n/2+2)$ (see Fig. 1), which leaves room for further improvements from below or from above.

[1]Department of Computer Science, University of California at Santa Barbara, CA-93106.

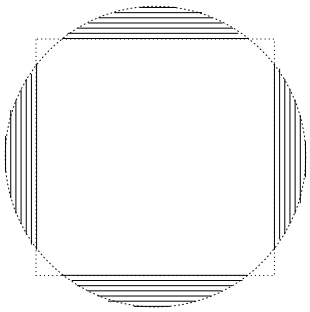


Figure 1: Axis-parallel segments clipped to a disk.

2 Axis-parallel segments

We may assume that there are at least $n/2$ horizontal segments in S. Let H, $H \subseteq S$, denote their set. For two segments $s \in H$ and $t \in H$, we say that s *supports* t if there is an x-monotone and y-monotone curve connecting a point of s to a point of t such that it does not cross any segment of S. (This includes the case where there is vertical visibility between s and t.) We say that $s \prec t$ iff there is a sequence $(s = s_0, s_1, s_2, \ldots, s_r = t)$ in H such that s_i supports s_{i+1}, $i = 0, 1, 2, \ldots, r-1$ (Fig. 2). The relation $\prec$ is a partial order in H. (A similar order was used in [5]).

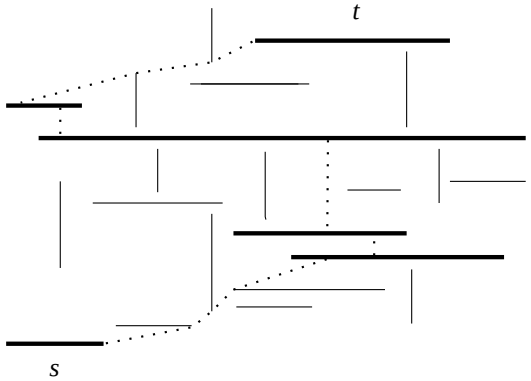


Figure 2: $s \prec t$.

By Dilworth's theorem [2], there is either (i) a chain or (ii) an anti-chain of size $\sqrt{n/2}$ with respect to $\prec$.

(We note that the size of the maximal chain and anti-chain can be $\sqrt{n/2}$ simultaneously.) In either case, we show that all segments in the chain or anti-chain can be linked together in a common alternating path.

In case (i), let $s_1, s_2, \ldots, s_r$ be a sequence of r, $r \geq \sqrt{n/2}$, segments of H such that each s_i supports s_{i+1}. Denote the left and right endpoint of s_i by a_i and b_i. Let $\gamma(i)$, be the x- and y-monotone curve connecting s_i and s_{i+1} such that the two endpoints of $\gamma(i)$ are $v_i \in s_i$ and $w_i \in s_{i+1}$ (fig. 2).

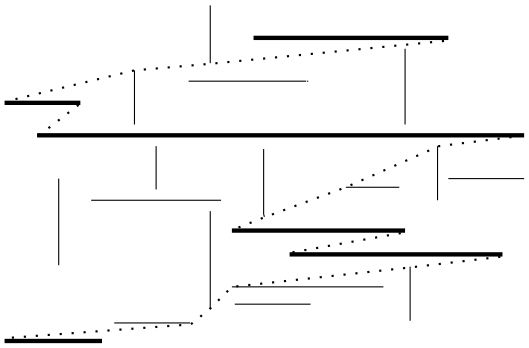


Figure 3: The initial paths $\pi(i)$.

For every i, place a rubber band along the path $(a_i v_i) \cup \gamma(i) \cup (w_i b_{i+1})$. Then let the rubber band contract while its endpoints stay pinned down at a_i and b_{i+1} with the constraint that it cannot cross any segment of S. The rubber band forms a polygonal path $\pi(i)$ through segment endpoints lying between s_i and s_{i+1} (Fig. 4). Notice that $\pi(i)$ remain x- and y-monotone.

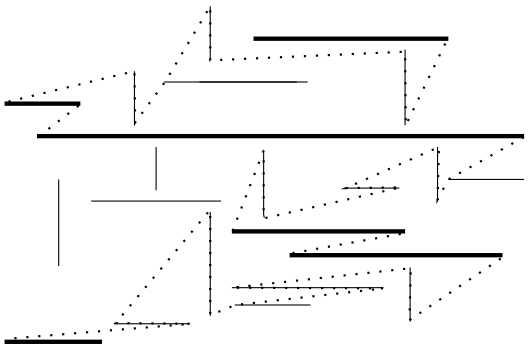


Figure 4: Paths $\pi(i)$ after one iteration.

Next, we expand recursively every $\pi(i)$ into an alternating path between s_i and s_{i+1}. We want to make sure that the concatenation of the resulting $r-1$ alternating paths is also a simple alternating path, that is, the $r-1$ alternating paths are pairwise disjoint.

Consider a path $\pi(i)$. If there is a segment $\hat{s}$ which has exactly one endpoint $v(\hat{s})$ with $\pi(i)$, then we modify $\pi(i)$ to go along $\hat{s}$ and visit the second endpoint of $\hat{s}$. We call this operation an *expansion* of $\pi(i)$. The expansion practically means that we pick the segment of $\pi(i)$ lying before or after the common vertex $v(\hat{s})$ and pull the rubber band to the second endpoint of $\hat{s}$ with the constraint that it cannot cross any other segment of S. We choose the *directions* of the expansion as follows: If $\hat{s}$ is horizontal and lies on the left (right) side of $\pi(i)$ then we expand the segment of $\pi(i)$ above (below) $v(\hat{s})$. If $\hat{s}$ is vertical and lies on the left (right) side of $\pi(i)$ then we expand the segment of $\pi(i)$ to the left (right) of $v(\hat{s})$ (see Fig. 4 and 5).

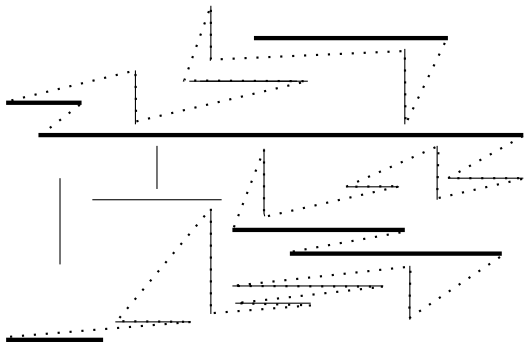


Figure 5: The resulting alternating path.

Inevitably, when we expand a segment of $\pi(i)$ to visit a second endpoint of $\hat{s}$, our path may hit other segment endpoints between s_i and s_{i+1}. The choice of directions of the expansion ensure that $\pi(i)$ never hits a segment endpoint that is already visited by $\pi(i)$ or a second endpoint of a segment whose one endpoint is in $\pi(i)$. We maintain two invariants:

1. Every piece of $\pi(i)$ which does not lie along a segment of S is x- and y-monotone;
2. If an $\hat{s}$ has one common point $v(\hat{s})$ with $\pi(i)$ and lies on the left (right) of $\pi(i)$, then $v(\hat{s})$ is its right or lower (left or upper) endpoint.

One can show that repeating the *expansion* operation on the path $\pi(i)$, we end up with $r-1$ pairwise disjoint alternating paths between the pairs (s_i, s_{i+1}).

In case (ii), note that any two segments in an anti-chain are separated by a vertical line, therefore the segments in the anti-chain have a linear left-to-right order. Consider the r, $r \geq \sqrt{n/2}$, segments of an anti-chain $A = \{s_1, s_2, \ldots, s_r\} \subset H$ labeled according to this order (Fig. 6). Denote the left and right endpoint of each s_i by a_i and b_i.

Observe that $s_i \not\prec s_{i+1}$ and $s_i \not\succ s_{i+1}$ implies that there is no x- and y-monotone curve between

s_i and s_{i+1} which avoids vertical segments but possibly crosses horizontal segments. Specifically, if the y-coordinate of s_i is smaller than that of s_{i+1}, then there is staircase (axis-parallel x- and y-monotone) curve between b_i and a_{i+1} whose every vertical segment is along a vertical segment of S (middle of Fig. 6). Informally, this staircase curve blocks any curve which would imply a relation $s_i \prec s_{i+1}$. (If the y-coordinate of s_i is larger than that of s_{i+1}, there is no specific condition.)

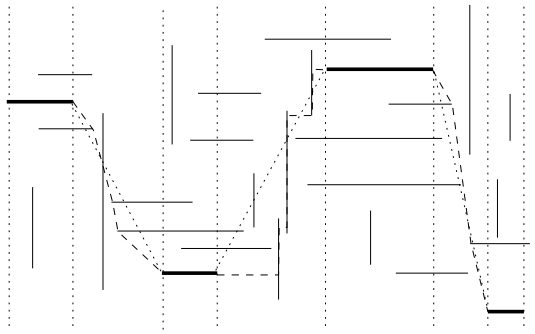


Figure 6: Linear oder in an anti-chain.

For every i, $i = 1, 2, \ldots t-1$, we connect b_i and a_{i+1} by a rubber band $\varrho(i)$: If b_i is above a_{i+1} then we find a monotone descending polygonal path that can cross vertical segments but must avoid horizontal segments. If b_i is below a_{i+1} then consider the polygonal staircase path from b_i to a_{i+1} as described above. In both cases, let us denote this polygonal path by $\varrho(i)$ (dashed in Fig. 6).

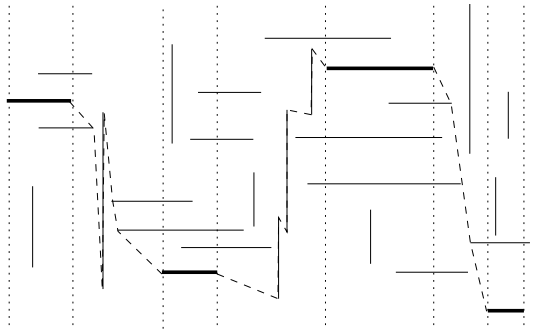


Figure 7: The initial curve $\varrho(i)$ with monotone descending visibility edges.

Recursively, at every intersection point of $\varrho(i)$ with a vertical segment vw, we force the rubber band $\varrho(i)$ to pass through v and w such that $\varrho(i)$ is ascending along vw. In this way, we obtain a curve $\pi(i)$ that does not cross any segment of S and whose pieces not lying along segments of S are x-monotone increasing and y-monotone descending.

Finally, we expand recursively every $\pi(i)$ into an alternating path between b_i and a_{i+1} using the similar expanding operations as in case (i). Consider a segment $\hat{s} \in S$ with one common point $v(\hat{s})$ with $\pi(i)$. If $\hat{s}$ is horizontal and lies on the left (right) side of $\pi(i)$ then we expand the segment of $\pi(i)$ below (above) $v(\hat{s})$. If $\hat{s}$ is vertical and lies on the left (right) side of $\pi(i)$ then we expand the segment of $\pi(i)$ the the left (right) of $v(\hat{s})$ (Fig. 8).

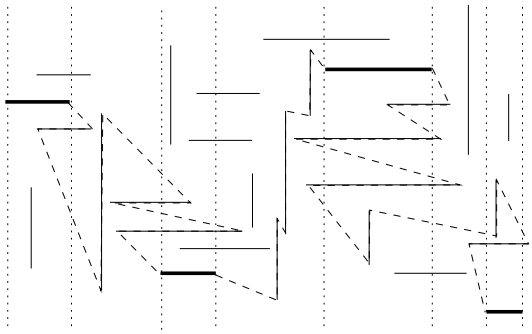


Figure 8: Winding the rubber band around obstacles.

We can maintain similar invariants as in case (i) (now, portions of $\pi(i)$ not lying along segments of S are monotone descending). One can show that the expansion of $\pi(i)$ does not touch any segment endpoint twice, every resulting path is simple. The invariants also ensure that all $r-1$ resulting alternating paths are pairwise disjoint. This completes the proof of Theorem 1.

References

[1] E.D. Demaine and J. O'Rourke, Open Problems from CCCG'99, in: *Proc. 11th Canadian Conf. on Comput. Geom. (Vancouver, 1999).*

[2] R. Dilworth, A decomposition theorem for partially ordered sets, *Ann. of Maths.* **51** (1950), 161–166.

[3] M. Hoffmann and Cs. D. Tóth, Alternating paths through disjoint line segments, submitted, presented at the *18th European Workshop on Computational Geometry (Warsaw, 2002).*

[4] M. Hoffmann and Cs. D. Tóth, Segment endpoint visibility graphs are Hamiltonian, *Comput. Geom. Theory Appl.*, in print.

[5] R. Tamassia, I.G. Tollis, A unified approach to visibility representations of planar graphs, *Discrete Comput. Geom.* **1** (1986), 321–341.

[6] J. Urrutia, Algunos problemas abiertos (in Spanish), in: *Actas de los IX Encuentros de Geometría Computacional (Girona, 2001).*

Shortest Paths in Polygonal Domains with Polygon-Meet Constraints

Ramtin Khosravi* Mohammad Ghodsi†

Department of Computer Engineering
Sharif University of Technology
P.O. Box: 11365-9415,
Tehran, Iran.
Tel: +98 (21) 600-5616

Abstract

In this paper, we study the problem of finding the shortest path in a polygonal domain in which the path should meet (touch or cross) a simple polygon in the domain. Our method uses the continuous Dijkstra paradigm and reflected wavefronts to solve the problem in worst-case optimal time $O(n \log n)$.

1 Introduction

Finding the shortest path between two points is a basic problem in computational geometry and has many applications in different areas such as motion planning and navigation. The problem is studied over various geometric domains such as simple polygons, polygonal domains, polyhedral surfaces, etc. Also several variations exists according to the metric for computing distances, and different constraints applied to the solution path. Examples of such restrictions are curvature constraints [2] or altitude constraints [1].

We study a special kind of constraints called polygon-meet constraints, in which the shortest path from source to destination should meet a given polygon in a polygonal domain. By meeting a polygon, we mean the path should have non-empty intersection with the closure (interior plus boundary) of a polygon, i.e. either touch the boundary of the polygon and reflect, or cross a part of the polygon.

One possible applications are resource-collection in which an object moving from a source point to a destination point has to collect some resources found in a certain region. Another application is visibility-related constraints in which the polygon to be met is the visibility polygon of a point or an object in the domain. Such a case may arise when direct visibility is needed between the moving object and the viewpoint, such as in communications, or guarding applications.

Our approach here is to use the *continuous Dijkstra* paradigm for finding the shortest polygon- meeting path between two points. As a result of using this paradigm, a subdivision of the domain can be built to answer single-source shortest polygon-meeting path queries. The idea of the method is to propagate a wavefront from the source until it touches the boundary of the target polygon. Upon touching the boundary, the original wavefront is marked as *met* and is propagated ignoring future contacts with the target polygon. Besides the original wavefront, its *reflected* version is also propagated in the opposite direction which is also marked *met*. By *met* wavefront we mean the set of points in the free space to which the length of the shortest polygon-meeting path is δ (Figure 1). The reflected part corresponds to those point whose shortest polygon-meeting path from source has been touched the boundary of the target polygon and has been reflected. If we use the method of Hershberger and Suri [3] for wavefront propagation, we obtain a time-bound of $O(n \log n)$ and $O(n \log n)$ space for our problem which is worst-case optimal. As the size of the constructed map is linear, each query can be answered in $O(\log n)$ time. To the best of our knowledge, no other result has been available on this problem.

*ramtin@mehr.sharif.edu
†ghodsi@sharif.edu

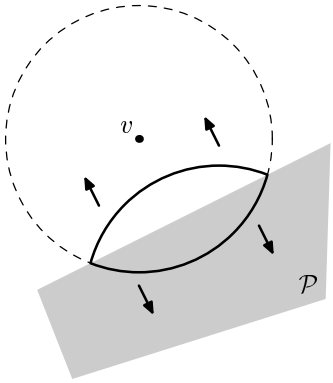


Figure 1: *met* wavefront propagation: the dashed circle is the wavelet generated by vertex v, the gray area belongs to the target polygon $\mathcal{P}$, the solid arcs are *met* wavelets. Arrows show the direction of wavefront expansion.

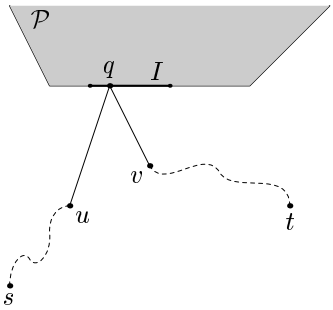


Figure 2: The linearity interval I (shown with heavy solid line) is a connected subset of the boundary of $\mathcal{P}$ such that the shortest path from s (resp. t) to its points passes u (resp. v) as the last vertex. $q \in I$ is the point with minimum total shortest path distance to s and t.

A similar problem is studied in [5] where the constraint is to meet the visibility region of a point. The problem is considered in several domains: simple polygons, polygonal domains, and polyhedral surfaces. The method is based on partitioning the boundary of the target polygon (the polygon to be met) into *linearity intervals* (L-intervals), such that the points of each interval has the property that their shortest paths to both source and destination has the same combinatorial structure. Because of this property, one can find a point $q(I)$ on an L-interval I that has the minimum total distance to source and destination in constant time. So, finding the shortest polygon-meeting path can be done by taking minimum over all L-intervals.

In the case of polygonal domains, the method mentioned yields to the time bound of $O(n \log n)$ which is similar to the bound obtained using the algorithm presented in the current paper. The advantage of the current method is in generating the shortest polygon-meeting path map which cannot be generated using the previous method.

2 Preliminaries

In this section we review some related terminology which is borrowed from [3], since our method relies on the shortest path algorithm given there. Suppose $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$ be the planar set of polygonal obstacles in the domain with distjoint closures, and s be the source point. Also suppose $\mathcal{P}$ be the target polygon to be met by the path. The total number of vertices in the obstacle polygons as well as the target polygon is assumed to be n. The plane minus the interior of all obstacle polygons is called the *free space*. Given a query point t, our goal is to find the shortest path between s and t which resides completely in the free space, having non-empty intersection with the closure of the target polygon $\mathcal{P}$.

The *shortest path map* of the source point s, denoted by $SPM(s)$, is a linear-size subdivision of the free space into regions (*cells*) such that the shortest path to all the points in one cells has the same

combinatorial structure, i.e., has the same sequence of obstacle vertices along the path. The last obstacle vertex along the path to the points in a cell is called the *root* of that cell. Each cell is star-shaped with respect to its root, which lies on the boundary of the cell. The boundaries of cells consist of portions of obstacle edges, extension segments (extensions of visibility graph edges incident on the root), and bisector curves. The bisector curves are, in general, hyperbolic arcs that are the locus of points that have shortest distance from two roots.

2.1 The Shortest Path Algorithm

Here we review the method of Hershberger and Suri for constructing shortest path map for a point in polygonal domain, which can be done in worst-case optimal time $O(n \log n)$, based on the continuous Dijkstra method. The method simuates the expansion of a wavefront form a point source in the presence of polygonal obstacles. The boundary of the wavefront is a set of cycles, each composed of a sequence of circular arcs. Each arc, called a *wavelet*, is generated by an obstacle vertex already covered by the wavefront; the vertex is called the *generator* of its wavelet. The meeting point between two adjacent wavelets sweeps along a bisector curve, which is either a straight line or a hyperbola. Simulating the wavefront requires processing events that change its topology. These events fall into two categories: wavefront-wavefront collisions and wavefront-obstacle collisions.

To speed up detecting and processing these events quickly, a special subdivision of size $O(n)$ is built on the vertices, temporarily ignoring the line segments between them. Each cell of this subdivision, called a *conforming subdivision*, has a constant number of straight line edges, contains at most one obstacle vertex, and satisfies the following crucial property: for any edge e of the subdivision, there are $O(1)$ cells within distance $2|e|$ of e. Then the obstacle line segments are inserted into the subdivision, but maintaining both the linear size of the subdivision and its conforming property—except now a non-obstacle edge e has the property that there are $O(1)$ cells within shortest path distance $2|e|$ of the edge. These cells form the units of the propagation algorithm: in each step, the wavefront is advanced through one cell. Since each cell has constant descriptive complexity, the propagation in a cell can be done efficiently. When propagating the wavefront across a boundary edge of a cell, instead of keeping track of the exact wavefront, two *approximate wavefronts* approaching the edge from opposite sides are maintained. Using approximate wavefronts, one can detect wavefront–wavefront collisions in a small neighborhood of their actual locations, marking those cells of the subdivision during the propagation phase. At the end of the propagation phase, the edges of the shortest path map are computed exactly for each cell from the collision information stored during propagation.

3 Computing Polygon-Meeting Paths

In this section we present our method for computing the shortest path with polygon-meet constraint. As mentioned earlier, the method is based on the wavefront propagation techniques in computing the shortest path map of the source point s. At the end of the execution, the algorithm generates the shortest polygon-meeting path map of the source point s, which can be used to answer queries in logarithmic time.

3.1 Geometric Properties

Suppose we have a line l, and a source point x in the plane (with no obstacles). Let H be the half-plane generated by l in which x lies. For a point $y \in H$, we define the shortest reflective distance from x to y, as the length of the shortest path that starts from x, meets some points of l, and continues to the point y. It is easy to see that the locus of points y that are of shortest reflective distance δ from x is the part of a circle of radius δ and center $\overline{x}$ that lies in H, where $\overline{x}$ is the reflection of x about l. If we consider a segment e instead of the line l, but with the same requirement for the shortest path, there will be two additional circular arcs generated from the two end-points of e.

Based on the above observations, we use the idea of reflecting a wavelet when colliding the boundary of $\mathcal{P}$, as well as letting it continue inside $\mathcal{P}$. By *original* wavefront we mean the wavefront resulting from the shortest path algorithm, ignoring the target polygon $\mathcal{P}$. Original wavelets are defined similarly. Upon collision of an original wavelet with the boundary of $\mathcal{P}$, a met wavelet of one of these kinds may be generated (Figure 3):

1. A wavelet that is essentially the part of the original wavelet that enters $\mathcal{P}$.

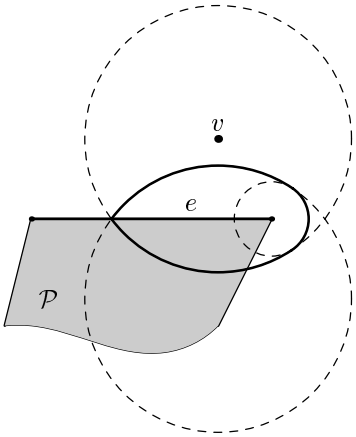


Figure 3: Three kinds of *met* wavelets are shown using solid arcs. The lower arc is part of the original wavefront generated by v, the upper arc is its reflected version, and the right arc (part of the small circle) is generated upon collision of the original wavefront by the right end-point of the edge e of $\mathcal{P}$.

2. A wavelet that is the reflection of the original wavelet about the edge of $\mathcal{P}$ with which the collision is occured.

3. A wavelet of zero radius. This happens when the original wavelet collides a vertex of $\mathcal{P}$.

We call these kind of wavelets (which are generated right after collision of original wavelets with the boundary of $\mathcal{P}$), *primary met wavelets*. In all of the above three forms, the primary wavelet is a circular arc: for the first kind, the center of the arc is the same as that of original wavelet, for the second kind, it is the center of the original wavelet reflected about the boundary edge, and for the third kind, it is the boundary vertex. Right after generation, primary met wavelets continue to propagate according to known rules of wavefront propagation in the Continuous Dijkstra paradigm. New wavelets may be added to the wavefront upon collision of met wavefronts with obstacle vertices, or may be deleted upon collision of met wavefront with each other. We call the newly generated wavelets, *secondary met wavelets*. Note that the collision of met wavelets with original wavelets are not considered during simulation.

Lemma 3.1 *The number of primary met wavelets is $O(n)$.*

Proof. As stated earlier, each primary met wavelet is genereated when an original wavelet first collides the boundary of $\mathcal{P}$. Since each cell of SPM is sweeped with one original wavefront, the portions of boundary that lie in one cell of SPM may be collided by only one original wavefront. In general, if we overlap SPM and the boundary of $\mathcal{P}$, the number of segments created on the boundary is $O(n^2)$, but as we will show, only $O(n)$ of these segments first collide with original wavelets. Others collide with wavelets that have collide these $O(n)$ segments before and therefore considered as met wavelets, not original ones.

To show that the number of segments on the boundary of $\mathcal{P}$ which generate primary met wavelets is linear, consider a cell f of the SPM with root r. If we connect r to vertices of $\mathcal{P}$ lying in f and continue until reaching the boundary of f, we obtain a number of slabs all having the root r in common. Each slab is intersected with a number of edges of $\mathcal{P}$ (possibly zero), but contains no vertex from it (Figure 4). It is easy to see that only one intersecting segment, the closest to r, can generate primary wavelets, since the original wavelet from r reaches all others after passing that segment and therefore considered as a met wavelet, not an original one. So the number of generator segments is bounded by total number of slabs in the SPM. Since each vertex of $\mathcal{P}$ adds at most one slab to the cell containing it, $\mathcal{P}$ has $O(n)$ vertices, and SPM has $O(n)$ cells, this bound is $O(n)$. Since each generator segment may generate at most two primary met wavelets and each vertex of $\mathcal{P}$ generates at most one such wavelet, the total number primary met wavelets is $O(n)$.

□

By *shortest $\mathcal{P}$–meeting path map* ($\mathcal{P}$-SPM for short), we mean the decomposition of the free space of the polygonal domain $\mathcal{P}$ to cells such that the shortest $\mathcal{P}$-meeting path to all points in one cell has the same combinatorial structure. Corresponding to two types of met wavelets (primary and secondary),

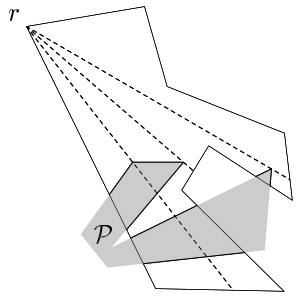


Figure 4: Proof of Lemma 3.1.

there are two kinds of cells in the $\mathcal{P}$-SPM, namely primary and secondary cells. Boundary of all primary cells have non-empty intersection with the boundary of $\mathcal{P}$. This intersection may be a segment, or a single vertex of $\mathcal{P}$.

Lemma 3.2 *The shortest polygon-meeting path map ($\mathcal{P}$-SPM) has linear size.*

Proof. From Lemma 3.1, we can easily conclude that the number of primary cells is linear. Since the secondary cells are generated as the result of propagation of circular arcs, they have the same properties as the cells of SPM, like the property stating that each obstacle vertex is the generator of at most one cell of the SPM. The same property holds for $\mathcal{P}$-SPM and we can conclude that the map has linear size. □

Finally, note that unlike the wavefront in the original shortest path algorithm, the met wavefront in our algorithm may be disconnected, i.e. consists of multiple closed chains of circular wavelets.

3.2 Computing $\mathcal{P}$-SPM

To compute $\mathcal{P}$-SPM, we use two extensions of the algorithm of [3] that the authors have mentioned: non-point sources and multiple sources with specified release times. The algorithm runs in two phases:

1. Computing the primary met wavelets.
2. Propagating the met wavefront.

In order to compute the primary met wavelets, we first use the original wavefront propagation algorithm to find the times at which the original wavefront meets the boundary of $\mathcal{P}$ generating primary met wavelets. To do this, we first consider $\mathcal{P}$ as an obstacle in the domain. This prevents original wavelets to enter the interior of $\mathcal{P}$. At the end of this phase, the following information should be obtained and stored:

1. The first time at which each wavelet collides the boundary $\mathcal{P}$. Note that if more than one wavelet collide a boundary edge, the times of all collisions should be stored.
2. The first time at which a wavelet collides a vertex of $\mathcal{P}$.

Note that considering $\mathcal{P}$ as an obstacle may cause some of the generated time labels be greater than the actual time if we let the wavefront enter $\mathcal{P}$ interior, but this does not make any problem, since in the second phase, when the simulation time reaches a time label, we check whether the generator of the event (either an edge or a vertex of $\mathcal{P}$) is already met. In this case we simpley ignore the event. The correctness of this decision follows from the fact that when we a collision occurs between an original wavelet and $\mathcal{P}$, we let the original wavelet propagete as a met wavelet. Therefore if the wavefront can reach a point of the boundary of $\mathcal{P}$ by entering its interior sooner than the time label, that point is no longer a generator.

In the second phase, we consider the problem as an extension of the standard shortest path algorithm in which there are multiple non-point sources with specified release time. Since the met wavefront may have several independent components, we should consider the propagation in the multiple-source model. As the times the primary met wavelets are generated may be different, these component have different release times. Considering the algorithm based on the non-point extension is due to the fact that the first

two kind of primary wavelets which are generated as a result of collision of original wavelets and $\mathcal{P}$ edges has non-zero radius when generated. This requires special attention for initialization of these wavelets.

In order to incorporate these characteristics into the original algorithm we should initialize the event priority queue of the shortest path algorithm with the information obtained from the first phase such that each primary met wavelet has one entry in the queue with an initial delay associated with it. When processing an event of this kind, the algorithm should compute the initial distance to neighboring conforming subdivision cells (which are constant in number). The rest of the processing is the same as what we have in ordinary shortest path algorithm. Note that the details of multiple-source version of the algorithm should be considered in this part to keep track of the wavefront colors and special case of overlapping well-covering regions of initial primary wavelets.

Using the above schema for the algorithm, we can conclude our main result:

Theorem 3.1 *For a polygonal domain of total complexity n, and a target polygon of size $O(n)$, the shortest polygon-meeting path map of the domain can be built in time $O(n \log n)$ and $O(n \log n)$ space.*

Proof. Since the target polygon has $O(n)$ edges, the first phase of our algorithm can be done in $O(n \log n)$ time and space using standard shortest path algorithm. The event queue can be also initialized in $O(n \log n)$ time. Since the total number of met wavelets is bounded by $O(n)$ and the size of the resulting map is also linear, the time required to construct the map is $O(n \log n)$ using extensions of the standard algorithm in [3]. □

4 Conclusions

We presented an algorithm for finding a special kind of shortest path map, in which the path is constrained to meet a target polygon, which may be a visibility polygon or a resource area in practical applications. The algorithm was based on a variation of the Continuous Dijkstra method for constructing shortest path maps. It is possible to study the problem in several extensions. One exension is to consider different metrics for distance calculations, such as link–distance.

Another extension is to consider the problem in other domains such as simple polygons or polyhedral surfaces. The case of simple polygon is particularly interesting since existence of linear time algorithms for finding shortest paths arises the question of solving the problem in linear time. Extending the method discussed in this paper to the case of polyhedral surfaces is also interesting since the subquadratic algorithm of Kapoor [4] for finding the shortest path in that domain is based on the Continuous Dijkstra paradigm.

If we consider the problem in the polygonal domain with the extension that the path should meet several polygons, it is called *TSP with neighborhoods* problem which is NP-hard and is studied in [6], but if we fix the order of meeting the target polygons, the problem seems to be less complex, yet we do not have any result for it.

References

[1] M. De Berg and M. Van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18:306–323, 1997.

[2] Jean-Daniel Boissonnat, André Cérézo, and Juliette Leblond. Shortest paths of bounded curvature in the plane. *Internat. J. Intell. Syst.*, 10:1–16, 1994.

[3] John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. Manuscript, Washington University, 1995.

[4] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proc. 32th Annu. ACM Sympos. Theory Comput.*, pages 770–779, 1999.

[5] Ramtin Khosravi and Mohammad Ghodsi. Shortest paths with single-point visibility constraints, Submitted for publication.

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

Tiling Polyominoes with Squares that Touch the Boundary

Andreas Spillner
Institut für Informatik, Friedrich-Schiller-Universität Jena, 07743 Jena
spillner@minet.uni-jena.de

Abstract

We study the problem of tiling a polyomino P with squares such that every tile S has a nonempty intersection with the boundary of P. We are especially interested in tilings with a minimum number of squares and in efficient algorithms to find such tilings.

1 Introduction

Our setting in the present paper will be the plane $\mathbb{R}^2$. For a subset $M \subseteq \mathbb{R}^2$ we denote by $int(M)$ the interior of M, by $bd(M)$ the boundary of M and by $conv(M)$ the convex hull of M. We will consider subsets of $\mathbb{R}^2$ which are usually called polyominoes [3].

Definition 1.1 The set $\mathfrak{C}$ of cells in the plane is defined as

$$\mathfrak{C} = \{conv(\{(k,l),(k+1,l),(k,l+1),(k+1,l+1)\}) : k,l \in \mathbb{Z}\}$$

Definition 1.2 $P \subseteq \mathbb{R}^2$ is called polyomino if and only if there exists a finite nonempty set $\mathfrak{B} \subseteq \mathfrak{C}$ such that $P = \cup_{C \in \mathfrak{B}} C$ and $int(P)$ is connected.

Definition 1.3 A rectangle $R \subseteq \mathbb{R}^2$ is called a cell rectangle if and only if R is a polyomino. A cell rectangle which is a square is called a cell square.

Definition 1.4 Let P be a polyomino. A set $\mathfrak{T}$ of cell squares is called a boundary touching tiling or BTT of P if and only if

1. $P = \cup_{S \in \mathfrak{T}} S$
2. $\forall S_1, S_2 \in \mathfrak{T}\ (S_1 \neq S_2 \ \Rightarrow\ int(S_1) \cap int(S_2) = \emptyset)$
3. $\forall S \in \mathfrak{T}\ (S \cap bd(P) \neq \emptyset)$

The tiling of bounded subsets $M \subseteq \mathbb{R}^2$ with squares that touch the boundary of M was to our knowledge first studied in [4]. However the definition of a BTT given there allows tilings with squares of arbitrary side length and tilings with an infinite number of squares. Then it is shown, that every open set $M \subseteq \mathbb{R}^2$ such that M is a homeomorph of an open disk or a homeomorph of an open disk with one hole admits a (possibly infinite) tiling with squares that touch the boundary of M. On the other hand there exists an open set $M \subseteq \mathbb{R}^2$ such that M is a homeomorph of an open disk with two holes which can not be tiled with squares that touch the boundary of M.

We will restrict ourselves to simple polyominoes, which are homeomorphs of a closed disk, and we use the definition of a BTT given above. With the same arguments as used in [4] one can show, that every simple polyomino admits a BTT. Now by defintion a BTT is finite and we can ask for BTT-s with a minimum number of tiles for a given simple polyomino.

Definition 1.5 Let P be a simple polyomino. $\mu(P) = min(\{|\mathfrak{T}| : \mathfrak{T} \text{ is a BTT of } P\})$

In the literature one finds results concerning the tiling of cell rectangles with a finite number of squares under various conditions. So for example: the number of squares has to be as small as possible [2], the squares of the tiling have to be pairwise incongruent [1]. The latter tilings are called perfect.

2 Some properties of boundary touching tilings

First we note, that for a given polyomino there need not be a unique BTT. Consider for example the rectangle with side lengths 2 and 8 displayed in figure 1. The integers inside the squares denote their side lengths.

Figure 1: Different BTT-s for a given polyomino.

Definition 2.1 Let P be a simple polyomino. A straight line segment C with endpoints x and y is called a cut through P if and only if $C \cap bd(P) = \{x, y\}$ and C dissects P into two simple polyominoes P_1 and P_2, that is $P_1 \cup P_2 = P$ and $P_1 \cap P_2 = C$.

Lemma 2.1 Let P be a simple polyomino and $\mathfrak{T}$ a BTT of P with $|\mathfrak{T}| > 1$. Then there exists a cut C through P such that $C \subseteq \cup_{S \in \mathfrak{T}} bd(S)$.

This lemma can be proved by contradiction. We suppose there is no such cut C. Then we can construct a simple closed curve H such that $H \subseteq int(P)$ and $H \subseteq \cup_{S \in \mathfrak{T}} bd(S)$. But this contradicts the fact that every $S \in \mathfrak{T}$ touches the boundary of P.

Corollary 2.1 Let R be a cell rectangle and $\mathfrak{T}$ a BTT of R with $|\mathfrak{T}| > 1$. Then there exist at least two distinct squares $S_1, S_2 \in \mathfrak{T}$ such that the side length of S_1 equals the side length of S_2 and is minimal among the side lengths of the squares in $\mathfrak{T}$.

Corollary 2.2 For a cell rectangle a BTT with more than one tile can never be a perfect tiling.

In the next section we want to use the cut property of a BTT to give an algorithm, which for some subclass of polyominoes admits a rather efficient computation of BTT-s with the mimimum number of tiles. Such BTT-s we will call minimum BTT-s for short.

3 Boundary touching tilings with the minimum number of tiles

Our idea for an algorithm to compute a minimum BTT for a given simple polyomino P is rather straight forward. We try every possible cut through P. From a cut C we get the two subpolyominoes $P_1(C)$ and $P_2(C)$. We compute a minimum BTT $\mathfrak{T}_1(C)$ for $P_1(C)$ and a minimum BTT $\mathfrak{T}_2(C)$ for $P_2(C)$. Then $\mathfrak{T}(C) = \mathfrak{T}_1(C) \cup \mathfrak{T}_2(C)$ is a BTT for P. To obtain a minimum BTT for P we only have to search for a cut C through P such that $|\mathfrak{T}(C)|$ is minimum.

However if the minimum BTT-s for the subpolyominoes are computed by a recursive call to the same algorithm we will end up with a very slow algorithm. Unfortunately in general we were not able to overcome this difficulty. Only for a very restricted subclass of simple polyominoes we could do better.

Definition 3.1 A polyomino P is called orthogonally convex if and only if for every horizontal or vertical straight line S the set $S \cap P$ is empty or a straight line segment.

Now when we have an orthogonally convex polyomino P consisting of n cells, the number of possible subpolyominoes resulting from cuts is bounded by a polynomial in n. Therefore we can apply the dynamic programming technique and obtain an algorithm the running time of which is bounded by a polynomial in n too.

4 Rectangles

In this section we present some results concerning cell rectangles, because on the one hand, as has been already mentioned in the introduction, there has been done quiet a lot of research on tiling rectangles by squares. On the other hand we think these results are interesting in themselves and leave to us some open questions.

First cell rectangles are special orthogonally convex polyominoes. So we can apply the algorithm given above to compute minimum BTT-s. But we can tune the algorithm so that we end up with a running time in $O(n^2)$ where n is the number of cells the rectangle consists of.

Definition 4.1 Let a and b be positive integers such that $a \leq b$. We set $\mu(a, b) = \mu(R)$ where R is a cell rectangle with side lengths a and b.

The function $\mu(\cdot, \cdot)$ shows some kind of periodicity:

Lemma 4.1 Let a and b be positive integers such that $a\lfloor \frac{a}{2} \rfloor + 2a \leq b$. Then $\mu(a, b) = \mu(a, b-a)+1$.

Remark 4.1 The equation in the above lemma does not hold in general. For example we have $\mu(17, 19) = 11$ but $\mu(17, 36) = 10$.

Another interesting problem is to give bounds for the function $\mu(\cdot, \cdot)$. So far we were only successful in giving a lower bound. We employ a technique which has been already used in [1] and [2]. We are given a rectangle R with side lengths a and b such that $a \leq b$ and a and b are relatively prime. For any BTT $\mathfrak{T}$ of R with $|\mathfrak{T}| = m$ we construct a graph $G(\mathfrak{T})$ with m edges. It is known that the number of spanning trees of $G(\mathfrak{T})$ is bounded from below by b. On the other hand we were able to show that the number of spanning trees of $G(\mathfrak{T})$ is at most the mth Fibonacci number F_m. Hence we have $b \leq F_m$ or with other words a logarithmic lower bound on m in terms of b.

5 Concluding remarks

There remain some unsettled questions: Does an efficient algorithm exist, which given a simple polyomino P computes a minimum BTT for P? What about an upper bound on the function $\mu(\cdot, \cdot)$? The work done in [2] might indicate, that there are connections to number theory. Finally we remark that the condition that every square in the tiling has to touch the boundary is really a restriction with respect to the number of squares in a minimum tiling. Consider for example the cell rectangle R with side lengths 11 and 13. A minimum BTT of R contains 8 squares, but we can tile R with 6 cell squares. Can we say something about how many more squares we need due to the condition of boundary touching?

References

[1] R. L. Brooks, C. A. B. Smith, A. H. Stone, and W. T. Tutte. The dissection of rectangles into squares. *Duke Mathematical Journal*, 7:312–340, 1940.

[2] Richard Kenyon. Tiling a rectangle with the fewest squares. *Journal Combinatorial Theory - Series A*, 76:272–291, 1996.

[3] David A. Klarner. Polyominoes. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 225–240. CRC Press, 1997.

[4] W. T. Trotter, Jr. Tiling bounded open sets with squares that touch the boundary. In *Annals of the New York Academy of Science*, volume 440, pages 304–321. 1985.

Best Fitting Rectangles

Manuel Abellanas* Ferran Hurtado† Christian Icking‡ Lihong Ma‡
Belén Palop§ Pedro A. Ramos¶

Extended Abstract, January 2003

1 Introduction

We solve an interesting optimization problem motivated by facility location and tolerancing metrology, see [5, 6, 9]: What rectangle fits best a given set of points? This problem also arises in dealing with paper position sensing [3]. Although our problem has some similarities to the problem of the largest empty rectangle for which no $O(n \log n)$ time solution is known, see [4, 8], for our problem there is a simple algorithm which runs within that time if the aspect ratio of the rectangle is given and even in time $O(n)$ if not. Other problems of this kind include the fitting of points by a circle, see [7], and offset polygon problems, see [1, 2].

2 Notations

We are given a set P of n point sites in the plane. Our task is to determine the rectangle which is, in some sense, closest to all of them.

As usual, the distance between a point and an extended object means the distance between the point and the closest point on the object, so the distance between a point p and a rectangle R is

$$d(p, R) = \min_{q \in R} d(p, q) \, .$$

Here, $d(p, q)$ denotes the distance in the underlying metric. Note that by rectangle we mean the boundary of the rectangle, not the interior. So a point in the interior of a rectangle has a non-zero distance to the rectangle.

For a certain subset, $\mathcal{R}$, of admitted rectangles, we are looking for the best fitting one, i. e., a rectangle such that

$$\max_{p \in P} \; d(p, R)$$

is minimized over all rectangles R of that kind. In other words, the best fitting rectangle R_{opt} fulfills

$$\max_{p \in P} \; d(p, R_{opt}) \;=\; \min_{R \in \mathcal{R}} \; \max_{p \in P} \; d(p, R) \, .$$

Different kinds of admitted rectangles, $\mathcal{R}$, and different metrics generate different problems to be solved.

An environment of a rectangle is called a frame. More precisely, the set of points whose distance to a rectangle R is less or equal to ε is called the *ε-frame* of R, for short F_R^ε. The two

*Dept. de Matemática Aplicada, Universidad Politécnica de Madrid, Spain
†Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain
‡Praktische Informatik VI, FernUniversität Hagen, Germany
§Dept. de Informática, Universidad de Valladolid, Spain
¶Dept. de Matemáticas, Universidad de Alcalá, Spain

closed boundaries of F_R^ε are called the *outer and inner ε-offsets* of R. The ε-frame of R is also the Minkowski sum of R and the unit circle of the underlying metric scaled by ε.

In this paper we concentrate on axis-parallel rectangles with or without prescribed aspect ratio, and we will use the L_∞-distance as underlying metric, which has axis-parallel squares as unit circles. Therefore, the outer and inner offsets of a rectangle are also rectangles, see Figure 1.

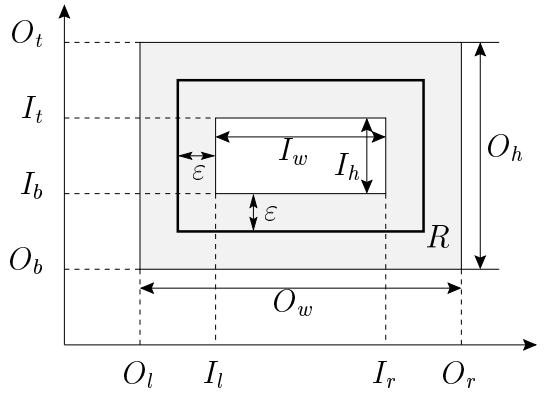


Figure 1: The ε-frame of R and its edges.

The *best fitting rectangle problem* is equivalent to looking for the narrowest covering ε-frame, i. e., the frame with the smallest ε that covers all sites.

3 Arbitrary aspect ratio

If we do not prescribe a certain aspect ratio of the rectangle then to find the best fitting rectangle is an easy problem.

Lemma 1 *Let R be a rectangle whose ε-frame covers P and B the bounding box of P. Then we have for all $p \in P$*

$$d(p, B) \leq 2\varepsilon \ .$$

Proof. Let $p \in P$ and let r be the point of R closest to p (one of them if there are several); we know $d(p, r) \leq \varepsilon$.

The (horizontal or vertical) line through p and r intersects B in two points. It is clear that at least one of the two is not farer than ε away from r because otherwise one of the sites on B would not be covered by the ε-frame of R, thus $d(r, B) \leq \varepsilon$.

We can combine the two inequalities and use the triangle inequality to obtain

$$d(p, B) \leq d(p, r) + d(r, B) \leq 2\varepsilon \ ,$$

which is our claim. □

Now an optimal solution can be obtained as follows.

- Compute B, the bounding box of P, this will turn out to be the outside of an optimal ε-frame.
- Compute $\max_{p \in P} d(p, B)$; call this 2ε.
- Let R be the inner ε-offset of B; this is a best fitting rectangle.

To prove the correctness of the algorithm, which clearly runs in time $O(n)$, it suffices to say that F_R^ε covers all sites and that there is no covering frame of a smaller width, by Lemma 1.

Remark that in most cases the best fitting rectangle with arbitrary aspect ratio is not unique.

4 Given aspect ratio

Now the aspect ratio, $a = \frac{height}{width}$, of the considered rectangles is also given. The problem is more complicated because the bounding box is no longer such a direct key to the solution. Nevertheless, we have the following property.

Lemma 2 *The inner and outer ε-offsets of any best fitting rectangle with given aspect ratio, a, contain a point of P. There is always an optimal solution which contains points of P on at least four of its eight offset edges.*

For the position of the points on the four offset edges, a lot of cases seem to be possible, at first sight. By the next lemma, we reduce the number of cases to three.

As a short and precise notation, we introduce the following abbreviations. For the Y-coordinates of the horizontal offset edges of a certain frame we say O_t, O_b, I_t, and I_b to the outer and inner top and bottom edges, and for the X-coordinates of the vertical offset edges we say O_l, O_r, I_l, and I_r to the outer and inner left and right edges, see Figure 1.

Lemma 3 *There is always a best fitting rectangle with given aspect ratio, a, possibly after a rotation of P by $\pm 90°$ or $180°$, that corresponds to one of three main cases, see Figure 2:*

Case 1 *O_t, O_b, and I_t are determined by points of P.*

Case 2 *O_t, O_b, I_l, and I_r are determined by points of P.*

Case 3 *O_t, I_t, and I_b are determined by points of P.*

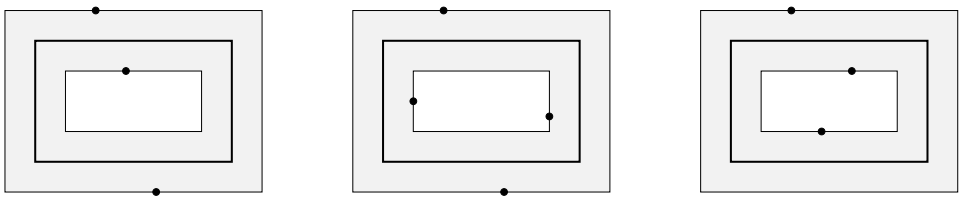


Figure 2: The three main cases for best fitting rectangles.

Our main result says that there is a simple algorithm to solve these cases.

Theorem 4 *A best fitting rectangle with given aspect ratio, a, for n points can be computed in time $O(n \log n)$.*

Proof. Corresponding to the three cases of Lemma 3 and four main orientations (rotations), our algorithm will find a best fitting rectangle in three main steps, each of which must be executed four times. All steps are independent from each other. For the simplicity or the description we assume that no two points of P have identical X- or Y-coordinates. Nevertheless the treatment of the general case is not difficult at all.

Due to the lack of space we sketch only the **algorithm for Case 1**. We perform a sweep from the middle of the interval (O_b, O_t) simultaneously to the top and to the bottom.

$O_t = y_{\max}$, $O_b = y_{\min}$, $O_h = O_t - O_b$, they correspond to the bounding box.
For all points $p = (x, y)$ of P compute $\varepsilon(p) = \frac{1}{2}\min(O_t - y, y - O_b)$.
Sort the points by decreasing ε-value and renumber the points, p_1, p_2, ..., in that order.

Let $\varepsilon_i = \varepsilon(p_i)$ and $\varepsilon_{start} = \frac{O_t - O_b}{4} \min(1, \frac{2}{a+1})$.
Let T be an empty balanced tree to contain points according to their X-coordinates.
Insert all points p_i with $\varepsilon_i > \varepsilon_{start}$ into T.
For all remaining points $p_i = (x_i, y_i)$ in this order do
    Let $I_w = \frac{O_b}{a} - 2\varepsilon_i(1 + \frac{1}{a})$ be the width of the inner offset.
    Search the two subsequent $x_l, x_r \in T$ with $x_l < x_i < x_r$.
    Let $h_l = \min(x_{\min}, x_i - 2\varepsilon_i, x_r - 2\varepsilon_i - I_w)$
    and $h_r = \max(x_l - 2\varepsilon, x_i - 2\varepsilon_i - I_w, x_{\max} - 4\varepsilon_i - I_w)$.
    If $h_l \leq h_r$ then we have found a narrower covering frame with
        $O_l = h_l$, $I_l = h_l + 2\varepsilon_i$, $I_r = I_l + I_w$, $O_r = I_r + 2\varepsilon_i$,
        $I_t = O_t - 2\varepsilon_i$, and $I_b = O_b + 2\varepsilon_i$.
    Insert x_i into T.

The algorithm chooses the initial value ε_{start} such that the inner offset is just a line segment, and all points lie between $O_b = y_{\min}$ and $O_t = y_{\max}$. Then ε is decreased and more and more points do no longer lie between O_b and $O_b + 2\varepsilon$ or $O_t - 2\varepsilon$ and O_t. These points are stored in T, and we have to find a position of the vertical edges that corresponds to Case 1, i. e., they lie between O_l and $I_l = O_l + 2\varepsilon$ or $I_r = O_r - 2\varepsilon$ and O_r while the current point p_i must lie between I_l and I_r; the test if $h_l \leq h_r$ takes care of exactly this.

The running time of this algorithm (also for the other cases) is in $O(n \log n)$ since for each point we perform just one insert and one query operation. □

References

[1] G. Barequet, A. J. Briggs, M. T. Dickerson, and M. T. Goodrich. Offset-polygon annulus placement problems. *Comput. Geom. Theory Appl.*, 11:125–141, 1998.

[2] G. Barequet, M. T. Dickerson, and M. T. Goodrich. Voronoi diagrams for convex polygon-offset distance functions. *Discrete Comput. Geom.*, 25(2):271–291, 2001.

[3] M. Bern and D. Goldberg. Paper position sensing. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 74–81, 2002.

[4] B. Chazelle, R. L. Drysdale, III, and D. T. Lee. Computing the largest empty rectangle. *SIAM J. Comput.*, 15:300–315, 1986.

[5] J. M. Díaz-Bañez, J. A. Mesa, and A. Schöbel. Continuous location of dimensional structures. Report in Wirtschaftsmathematik 79, Department of Mathematics, Universität Kaiserslautern, Germany, 2002.

[6] Z. Drezner and H. W. Hamacher, editors. *Facility Location: Applications and Theory.* Springer-Verlag, Berlin, Germany, 2002.

[7] J. García-López, P. Ramos, and J. Snoeyink. Fitting a set of points by a circle. *Discrete Comput. Geom.*, 20:389–402, 1998.

[8] M. Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5:65–73, 1990.

[9] R. K. Walker and V. Srinivasan. Creation and evolution of the ASME Y14.5.1M standard. *Manufacturing Review*, 7(4):16–23, 1994.

Algorithms for Placing and Connecting Facilities and their Comparative Analysis

Klara Kedem, Irina Rabaev and Neta Sokolovsky

Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel

1 Introduction

Base stations are fixed stations used to send, receive and transmit signals. Each base station consists of a tower, communication equipment and antenna(s). An antenna transmits and receives radio waves. A base station serves users in a specific region defined by the area spanned by the antenna(s). Usually these regions are circular, but other considerations can be taken into account, such as topographic data, propagation model, number of covered customers, etc. If two base stations are not in the range of each other due to transmission constrains, we may use a relay stations to connect them. A relay station is a station with equipment to receive a signal and retransmit it. It is mounted on a tower or on an existing base station.

In this paper we discuss algorithms for placing base stations (also referred to as *facilities* or *servers*) and their interconnection: "Given a set of customers and a set of potential locations for base stations, pick the minimum number of base stations that serve all the customers and then connect the chosen base stations." This problem is referred to in the literature as the Connected Facility Location Problem [1, 2, 4]. We divide the problem into two sub problems and solve each one separately. The first sub problem is to choose a set of base stations that serves all the customers. We present the Integer Programming method that yields the optimal solution for this problem and then review three approximation solutions (Section 2). The second sub problem deals with adding connectivity between the chosen facilities. Here we present two heuristics: one is based on Dynamic Programming and the other is based on Integer Programming (Section 3). We have implemented all the algorithms discussed in the paper, the description of our implementation and the experimental results are given in Section 4.

2 Finding the set of facilities that serve all the customers

In this section we discuss the algorithms for finding the smallest set of base stations that serves **all** the customers. Given n locations where base stations can be placed, and given m customers, the goal is to build the minimal number of base stations, such that their union serves all the customers. ¿From the geometric point of view: there is a set of customers represented by points $P = \{p_1, p_2, \ldots, p_m\}$ and a set of regions $R = \{r_1, r_2, \ldots, r_n\}$. Pick the smallest number of regions such that their union covers all the customers. We discuss four algorithms that solve the stated problem.

2.1 The optimal solution

We use Integer Programming to find the optimal set of facilities that serves all the customers. The variables x_i, $i = 1, \ldots, n$, in our problem represent binary decisions whether to build a base station at a given location or not, where a positive decision is represented by 1, and a negative decision is represented by 0.

Let us formulate the constraints of the problem. Denote by T_i the set of base stations that can serve customer p_i, $i = 1 \ldots m$, (the customer p_i is in their range). At least one base station should serve customer p_i, therefore at least one variable from T_i is assigned the value 1. Thus, we have the constrains $\sum_{x_j \in T_i} x_j \geq 1, \forall\, i \in \{1, \ldots, m\}$. The cost of the solution is the total cost of the selected servers, so the goal is to minimize $\sum_i c_i x_i$, where c_i is the cost of building base station x_i.

2.2 The greedy algorithm

Our first sub problem is actually the Set Cover problem. One method to approximate the Set Cover problem is the greedy algorithm. It is quite straightforward: at each step choose the region

with the maximum number of remaining customers in it. Erase these customers from P and proceed iteratively. The algorithm continues until all customers have been covered. Performance ratio of the greedy algorithm is $O(\ln m)$ [3, 5].

We implemented this greedy algorithm, but improved its output in order to get rid of redundant base stations. In order to find the redundant servers, we divide the solution set into connected components (two servers a and b are connected if and only if there exist servers $a = x_1, x_2, \ldots, x_k = b$, $k \geq 2$, such that for $1 \leq i \leq k-1$ the regions spanned by x_i and x_{i+1} intersect). For each base station x_i in the connected component C we check if for each customer p in its range, there exists another base station x_j in C that covers p. If it is so, the server x_i is redundant and we discard it.

2.3 A randomized algorithm

In this section we discuss a randomized algorithm for finding the cover set. The algorithm uses an assumption that the number of intersections between regions spanned by the base stations is bounded by a constant l (e.g. each region intersects at most l other regions). The algorithm works as follows. At each step randomly pick a customer $p_i \in P$ and add **all** the servers that cover it to the set S of picked base stations. Update P by deleting from it all the customers currently covered by S and repeat until all customers are covered. We discard redundant base stations as in section 2.2.

To estimate the performance of this algorithm let us define OPT be the optimal set cover for the problem. At each step i the algorithm adds n_i base stations to the solution, where $n_i \leq l$ (the assumption). Since p_i must be covered, one of this n_i base stations must appear in OPT. The algorithm stops when all the base stations from OPT have been added to S. So, for each base station from OPT the algorithm might pick at most $l-1$ other base stations, hence $|S| \leq l|OPT|$ and the approximation ratio of the algorithm is $O(l)$.

2.4 Adding the greedy approach into Randomized algorithm

The algorithm we discuss in this section is a variation of the random algorithm in section 2.3. Instead of randomly picking the next customer from P we pick a customer covered by the least number of servers. For each customer p_i we assign a number n_i - the number of facilities that can serve customer p_i. At each step the algorithm picks a customer p_i with minimal n_i and adds to the solution all the base stations that cover it. We discard redundant servers as in section 2.2.

3 Connecting the base stations

We now want to connect the picked base stations. In a connected system each base station can transmit to any other (either directly or through auxiliary base stations). If it is impossible to connect two servers a and b due to the transmission radius or propagation constrains, we have to add relay stations to connect a and b. We build the connected system in three stages:
(1) We first find the base stations that can be connected by *only* adding relay stations to them.
(2) Next we divide the servers into clusters: a and b are in one cluster if there exist servers $a = x_1, x_2, ..., x_k = b$, $k \geq 2$, such that for $1 \leq i \leq k-1$ x_i and x_{i+1} are in the range of each other.
(3) In the final step we connect clusters by building new servers and adding relay stations.

Stages 1 and 2 can be done simultaneously by an MST algorithm. In stage 3 we use a greedy strategy: at each step connect the nearest pair of clusters by adding a small number of new servers. Next, replace these clusters by their union and the additional new base stations. Repeat until only one cluster is left.

To connect a pair of clusters we use two different approaches: Dynamic Programming and Integer Programming. We describe them below.

3.1 Dynamic Programming

The input to the Dynamic Programming algorithm consists of two clusters of base stations A and B and the set of the possible base station locations, D.

We build a full graph $G = (V, E)$, where $V = A \cup B \cup D$. We define a weight function $w : E \to R$ that maps edges to real-valued weight $w(u, v)$, where $w(u, v)$ = cost of connecting u and v. If u and v belong to the same cluster, then $w(u, v) = 0$, if it is impossible to connect u and v, then $w(u, v) = \infty$, if one or both of the nodes are in D then $w(u, v)$ = cost of building new server(s) + cost of connecting u and v by relay stations. The problem of connecting two clusters of base

stations is now posed as finding the least-weight path from u to v in G, where $u \in A$, $v \in B$ and the weight of a path is the sum of the weights of its edges. We use Dynamic Programming to find all the shortest paths between pairs of vertices in G and then find the shortest among the paths from a node in A and a node in B.

3.2 Integer Programming

We define two types of binary variables x_i and $y_{i,j}$: x_i represents the decision whether to create a new base station i, and $y_{i,j}$ whether to add a relay station on a server i that transmits towards server j. Each variable may get value 0 or 1, where 1 means that we choose to build the corresponding base station or to add the corresponding relay station, and 0 means we don't. In order to build a path between clusters A and B we keep the following properties:

(1) For cluster A (B) we choose exactly one relay station (which will be placed on a new base station) that has the ability to connect directly to the cluster A (B). We define T_A (resp. T_B) the set of relay stations that can connect directly to one of the antennas in cluster A (B). Then the constrains we have are: $\sum_{y_{ij} \in T_A} y_{ij} = 1, \sum_{y_{ij} \in T_B} y_{ij} = 1$.

(2) If we choose to add relay station $y_{i,j}$ on base station i, we have to add relay station $y_{j,i}$ on base station j, i.e. if $y_{ij} = 1$ then $y_{ji} = 1$. Hence, for each i and j, $y_{ij} - y_{ji} = 0$.

(3) On each new base station should be exactly two relay stations in order to continue the connectivity. Let BT_i be a set of relay stations that can be placed on base station x_i (this set is either empty or consists of exactly two relay stations), then for each $x_i \notin A \cup B \quad 2x_i - \sum_{y_{ij} \in BT_i} y_{ij} = 0$.

It is clear that if these properties hold, then we get a valid connection between the two clusters. The cost of the solution is the total cost of new servers and relay stations. Our goal is to find the cheapest solution, so the objective function is to minimize $\sum_{x_i \ \ y_{ij} \in BT_i} (x_i c_i + y_{ij} c_{ij})$, where c_i is the cost of building a base station x_i and c_{ij} is the cost of adding a relay station y_{ij}. If $x_i \in A \cup B$ then $c_i = 0$.

4 Implementation and the experimental results

We have implemented the two algorithms based on Integer programming (sections 2.1 and 3.2) in C++, and the approximation algorithms for finding cover set (sections $2.2 - 2.4$) and the Dynamic Programming algorithm (section 3.1) in Java . The algorithms were tested on Linux operating system on Pentium-III machine with 512 MB memory. While implementing the algorithms, we tried to use efficient data structures, but our code was not fully optimized.

The algorithms for the first sub problem were tested and compared on the following inputs:
(1) the number of customers varied between 100 to 5000,
(2) the customers were randomly picked in clusters (standing for villages) within a large square with side size 15000 units,
(3) the number of possible base station locations varied between 50 to 3500,
(4) the transmission radius of the base station was R = 250 units,
(5) potential locations for base stations were chosen in the following way:
- at grid points G with distance 2R between two points
- at grid points of G shifted by a vector ($\sqrt{2}$R, $\sqrt{2}$R)
- additional 70% of random points.

Table 1 shows that the outputs of the approximation algorithms were very close to optimal. We ran these algorithms on about 50 examples and got encouraging results: all of the approximation algorithms found near-optimal solution.

			Cover size			
	# customers	# bs	Randomized	Greedy	Alg.3	IP
1	5000	3060	215	215	213	210
2	5000	3056	233	235	235	229
3	5000	3162	239	241	237	231
4	3000	3267	206	208	204	201
5	3000	3162	207	211	205	203

Table 1. Finding the set of facilities that serve all the customers

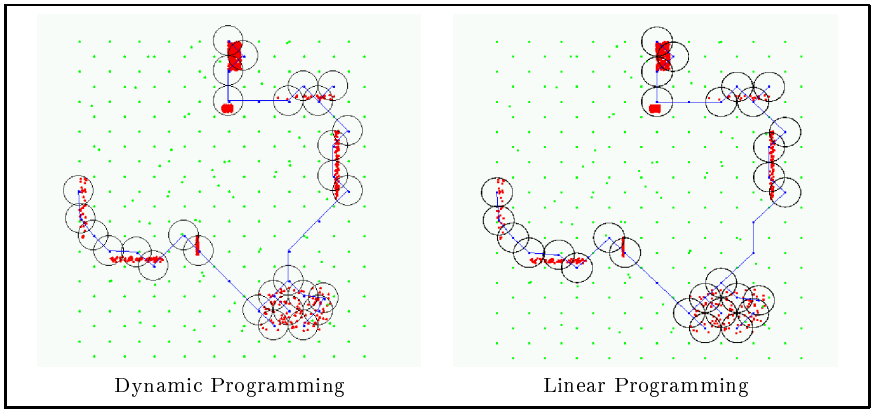


Figure 1. Connecting the base stations

The example illustrated in Fig.1 compares the performances of the Dynamic Programming and Integer Programming algorithms for the 2^{nd} sub problem. The input for the two programs consisted of 29 existing base stations and 235 possible locations for new base stations. The programs gave different connections with the same cost, but the run-time of the Dynamic Programming algorithm was 38308 msec, while the run-time of Integer Programming algorithm was 53581 msec. We used 5 kinds of relay stations with transmission radius 300, 350, 400, 450 and 500 units. In Fig.1 the red points represent the set of customers, green points represent the set of potential locations for base stations, black circles represent the regions spanned by the base stations that were picked in order to serve all the customers, blue points represent the base stations we use for connection and blue lines show the transmission. We run the programs on many example and in all of them, the costs of connecting the base stations produced by Integer Programming and Dynamic Programming algorithms were the same. Some of the results are summarized in Table 2.

	# existing bs	# potential places for bs	cost of connecting	Time in msec DP	IP
1	15	39	15084	2	15
2	9	74	30162	10	22
3	19	66	19110	9	38
4	17	51	17078	6	17
5	21	99	25730	3699	10287

Table 2. Summary of experimental results

In our experimentations we found that approximation algorithms are good in the sense they provide near-optimal solutions. For the problem of connecting the facilities into interconnected system we have described two heuristics based on Dynamic and Integer Programming accordingly. The solutions founded by these algorithms were different in connections, but had the same cost, which is not surprising since both optimize. In addition, the run-time of the Dynamic Programming algorithm was better in all our experiments.

References

[1] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. In *European Symposium on Algorithms*, pages 179–193, 1996.

[2] S. Guha and S. Khuller. Connected facility location problems. In *IMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 40, pages 179–190, 1997.

[3] P. Slavík. A tight analysis of the greedy algorithm for set cover. In *ACM Symposium on Theory of Computing*, pages 435–441, 1996.

[4] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. In *APPROX*, pages 256–270, 2002.

[5] U.Feige. A threshold of $\ln n$ for approximating set cover. In *The Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 314–318, 1996.

Chips on Wafers, or Packing Rectangles into Grids

Mattias Andersson* Joachim Gudmundsson† Christos Levcopoulos*

In the VLSI wafer industry it is nowadays common that multiple projects share a single fabrication matrix (the wafer); this permits fabrication costs to be shared among the participants. No a priori constraints are placed on either the size of the chips nor on the aspect ratio of their side lengths (except the maximum size of the outer bounding box). After fabrication, in order to free the separate chips for delivery to each participant, they must be cut from the wafer. A diamond saw slices the wafer into single chips. However cuts can only be made all the way across the bounding box, i.e., all chips must be placed within a grid. A grid is a pattern of horizontal and vertical lines (not necessarily evenly spaced) forming rectangles in the plane. There are some practical constraints, for example, the distance between two parallel cuts cannot be infinitely small, since machines with a finite resolution must be programmed with each cut pattern. Although some of these constraints may simplify the problem we will not consider them in this paper. This application leads us to define grid packing as follows.

Definition 1 *A set of rectangles $\mathcal{S}$ is said to be* grid packed *if there exists a rectangular grid such that every rectangle lies in the grid and there is at most one rectangle of $\mathcal{S}$ in each cell, as illustrated in Fig. 1. The area of a grid packing is the area of a minimal bounding box that contains all the rectangles in the grid packing.*

The general problem considered in this paper is now stated.

MAGP [Minimum area grid packing] Given a set $\mathcal{S}$ of rectangles find a minimum area grid packing of $\mathcal{S}$.

We also consider several interesting variants of the problem, for example:

MAkGP [Minimum area k-grid packing] Given a set of n rectangles and an integer $k \leq n$ compute a minimum area grid packing containing at least k rectangles.

MAGPAR [Minimum area grid packing with bounded aspect ratio] Given a set $\mathcal{S}$ of rectangles and a real number $\mathcal{R}$, compute a minimum area grid packing whose bounding box aspect ratio is at most $\mathcal{R}$.

MWP [Maximum wafer packing] Given a set of rectangles $\mathcal{S}$ and a rectangular region $\mathcal{A}$ compute a grid packing of $\mathcal{S}' \subseteq \mathcal{S}$ on $\mathcal{A}$ such that $|S'|$ is maximized.

MNWP [Minimum number of wafers packing] A plate is a pre-specified rectangular region. Given a set of rectangles $\mathcal{S}$ compute a grid packing of $\mathcal{S}$ onto a minimal number of plates.

MDGP [Minimum diameter grid packing] Given a set $\mathcal{S}$ of rectangles find a minimum diameter grid packing of $\mathcal{S}$.

A problem that is similar to grid packing is the tabular formatting problem [5]. In the most basic tabular formatting problem one is given a set of rectangular table entries and the aim is to construct a table containing the entries in a given order for each row and column. A problem more similar to ours, with the exception that the rectangles cannot be rotated, was considered by Beach [3] under the name "Random Pack". Beach showed that Random Pack is strongly NP-hard.

Our main result is a PTAS for the MAGP-problem and some of its variants. Surprisingly, if the value of ε is a large enough constant the algorithms will run in linear time.

The approximation algorithms all build upon the same ideas. The main idea is that for every possible grid $\mathcal{G}$ there exists a grid $\mathcal{G}'$ that can be uniquely coded using only $\mathcal{O}(\log n)$ bits such that the area of $\mathcal{G}'$ is at most a factor $(1+\varepsilon)$ larger than the area of $\mathcal{G}$. Now, let $\mathcal{F}$ be the family of these grids that can be uniquely coded using $\mathcal{O}(\log n)$ bits. It trivially follows that there

*Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. E-mail: mattias@cs.lth.se, christos@cs.lth.se.

†Technical University Eindhoven, Department of Computing Science, P.O. Box 513, 5600 MB Eindhoven E-mail: h.j.gudmundsson@tue.nl

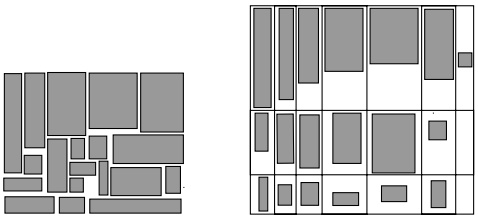


Figure 1: Input is a set of rectangles $\mathcal{S}$. Output a grid packing of $\mathcal{S}$

are only a polynomial number of grids in $\mathcal{F}$. Hence, every grid $\mathcal{G}'$ in $\mathcal{F}$ can be generated and tested. The test is performed by computing a maximal packing of $\mathcal{S}$ into $\mathcal{G}'$, which in turn is done by transforming the problem into an instance for the max-flow problem, i.e, given a directed graph with a capacity function for each edge, find the maximum flow through the graph. To obtain a PTAS one uses $\mathcal{O}(\log_{1+\varepsilon} n)$ bits for coding a grid in $\mathcal{F}$. In a similar way only $\log n/2$ bits are used to obtain a linear time approximation algorithm. More details about the family of grids, called the family of (α, β, γ)-*grids* are given in Section 2 together with two important properties. Then, in Section 3 we show how a grid is tested, and finally, in Section 4, we present the main results.

We will assume that width, height and weight of each rectangle $r \in \mathcal{S}$ is between $[1, n^c]$, for some constant c. We argue in [2] that this assumption can be made without loss of generality with respect to our approximation results.

1 Approximation algorithm

The structure of the approximation algorithm is given below. The two non-trivial steps, lines 10 an 11, will be described in detail in Sections 2 and 3 respectively. The last step, PackIntoGrid, is obtained by slightly modifying the procedure TestGrid. As input to the algorithm we will be given a set $\mathcal{S}$ of n rectangles and a real value $\varepsilon' > 0$.

Algorithm GridPack($\mathcal{S}, \varepsilon'$)
1. $bestVal \leftarrow \infty$,
   $\alpha, \beta = \sqrt{1+\varepsilon'}$, $\gamma = \frac{1}{\sqrt{1+\varepsilon'}-1}$
2. **for** each $1 \le i, j \le \log_\alpha n^c$ **do**
3. $\mathcal{S}_{i,j} \leftarrow \emptyset$
4. **for** each $r \in \mathcal{S}$ **do**
5. $i \leftarrow \lceil \log_\alpha \text{ width}(r) \rceil$
6. $j \leftarrow \lceil \log_\alpha \text{ height}(r) \rceil$
7. $\mathcal{S}_{i,j} \leftarrow \mathcal{S}_{i,j} \cup \{r\}$
8. **end**
9. **for** $k \leftarrow 1$ **to** $n^{f(\alpha,\beta,\gamma)}$ **do**
10. $\mathcal{G} \leftarrow$ GenerateGrid$(\alpha, \beta, \gamma, k)$
11. $val \leftarrow$ TestGrid$(\mathcal{G}, \mathcal{S}, \alpha, \beta, \gamma)$
12. **if** $val < bestVal$ **then**
13. $bestVal \leftarrow val$ and $bestGrid \leftarrow \mathcal{G}$
14. **end**
15. **Output** PackIntoGrid$(\mathcal{S}, bestGrid)$

The initialisation is performed on lines 1 to 3. On lines 4-8, the rectangles are partitioned into groups in such a way that a rectangle $r \in \mathcal{S}$ belongs to $\mathcal{S}_{i,j}$ if and only if the width of r is between α^{i-1} and α^i, and the height of r is between α^{j-1} and α^j. Lines 1-8 obviously run in linear time. Next, a sequence of grids $\mathcal{G}$ are produced in a loop of lines 9-14. They are the members of the so-called family of (α, β, γ)-grids which is described in Section 2. (It consists of $n^{f(\alpha,\beta,\gamma)}$ grids, where $f(\alpha, \beta, \gamma) = (2c \log(\alpha\beta\gamma))/(\log \alpha \log \beta)$.) The generated grid is tested and the weight of an approximative grid packing of $\mathcal{S}$ into the grid $\mathcal{G}$ is computed. If the grid packing is better than the previously tested grids then $\mathcal{G}$ is saved as the best grid tested so far. Finally, when all grids in the family of (α, β, γ)-grids have been generated and tested a call to PackIntoGrid performs a grid packing of $\mathcal{S}$ into the best grid found. This procedure is a simple modification of the TestGrid-step.

2 (α, β, γ)-grids

The aim of this section is to define the family $\mathcal{F}$ of (α, β, γ)-grids and prove two properties about $\mathcal{F}$. Before the properties can be stated we need the following definition. A grid G_1 is said to *include* a grid G_2 if every possible set of rectangles that can be grid packed into G_2 also can be grid packed into G_1. $\mathcal{F}$ has the following two properties.

1. For every grid G there exists a grid $\mathcal{G} \in \mathcal{F}$ that includes G and whose width and height is at most a factor $\left(\frac{\alpha\beta\gamma}{\alpha\gamma-1}\right)$ times larger than the width and height of G, and
2. $\#\mathcal{F} \le n^{f(\alpha,\beta,\gamma))}$.

The definition of an (α, β, γ)-grid is somewhat complicated, therefore we choose to describe this step by step.

A trivial observation is that two grid-packings are equivalent if the one can be transformed to the other by exchanging the order of rows and/or the columns. Hence we may assume that the columns are ordered with respect to decreasing width from left to right and that the rows are ordered with respect to decreasing height from top to bottom. This ordering will be assumed throughout the paper.

Consider an arbitrary grid G and let α be a real constant greater than 1. An *α-restricted* grid is a grid where the width and height of each cell in the grid is an integral power of α (multiple of α^i for some integer i).

Let G be a α-restricted grid. If the number of columns/rows of each size is an integral power of β then G is a (α, β)-restricted grid. The columns/rows in an α-restricted grid of width/height α^i are said to have column/row *size* i. A grid G is said to be γ-*monotone* if the number of columns (rows) of size i is at most a factor $\gamma \geq 1$ times smaller than the number of columns (rows) of size $i+1$ for every i.

The following lemma is proven in [2], hence, $\mathcal{F}$ is shown to have Property 1.

Lemma 2 *For any grid G there exists a γ-monotone (α, β)-restricted grid $\mathcal{G}$ (an (α, β, γ)-grid for short) that includes G and whose width and height is at most a factor $\left(\frac{\alpha\beta\gamma}{\alpha\gamma - 1}\right)$ greater than the width and height of G.*

Most often we do not need the actual grid, instead we are interested in the number of cells in the grid of a certain size. That is, the grid $\mathcal{G}$ is represented by a $[1..\log_\alpha n^c, 1..\log_\alpha n^c]$ integer matrix, where $\mathcal{G}[i,j]$ stores the number of cells in $\mathcal{G}$ of width α^i and, height α^j. We call this a matrix representation of a grid.

Now we turn our attention to the second property for the family $\mathcal{F}$ of (α, β, γ)-grids, i.e., the number of grids that are members of $\mathcal{F}$ is at most $n^{f(\alpha,\beta,\gamma)}$. Assume that we are given a member $f \in \mathcal{F}$ and that f has c_i columns of size i, $1 \leq i \leq \log_\alpha n^c$, and r_j rows of size j, $1 \leq j \leq \log_\alpha n^c$. Recall that r_j and c_i are integral powers of β. The idea of the scheme is as follows. The bit string, denoted S, is built incrementally. Consider a generic step of the algorithm. Assume that the bit string, denoted S_{j+1} has been built for all the row sizes greater than j and that the number of rows of size $(j+1)$ is $\beta^{(\#Rows)}$. Initially $S_{\log_\alpha n^c}$ is the empty string and $\#Rows = 0$. Consider the row size j. We will have two cases, either $r_j \leq (\beta^{\#Rows}/\gamma)$ or $r_j > (\beta^{\#Rows}/\gamma)$. In the first case, add '1' to S_{j+1} to obtain S_j. In the latter case, when $r_j > (\beta^{\#Rows}/\gamma)$, add $(\#Rows - (\log_\beta r_j - \log_\beta \gamma))$ zeros followed by a '1' to S_{j+1} to obtain S_j. Decrease the value of j and continue the process until $j = 0$, and hence, $S_0 = S$.

The same approach is used to generate the columns, hence we obtain the following observation that proves Property 2.

Observation 3 *S has length $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n)$.*

We obtain the following corollary:

Corollary 4 *Given a bit string B of length $2(\log_\alpha n^c(1 + \log_\beta \gamma) + \log_\beta n)$ one can in time $\mathcal{O}(\log^2 n)$ construct the unique matrix representation of the corresponding (α, β, γ)-grid, or decide that there is no corresponding (α, β, γ)-grid.*

3 Testing a grid

In the previous section we showed a simple method to generate all possible (α, β, γ)-restricted grids. For the approximation algorithm, shown in Section 1, to be efficient we need a way to pack a maximal number of rectangles of $\mathcal{S}$ into the grid. As input we are given a matrix representation of an (α, β, γ)-grid $\mathcal{G}$, and a set $\mathcal{S}$ of n rectangles partitioned into groups $\mathcal{S}_{i,j}$ depending on their width and height. Let $\mathcal{C}_{p,q}$ denote the number of cells in $\mathcal{G}$ that have width α^p and height α^q. We will give an exact algorithm for the problem by reformulating it as a max-flow problem. The problem could also be solved by reformulating it as a matching problem but in the next section we will show that the max-flow formulation can be extended to the weighted case. The max-flow problem is as follows.

Given a directed graph $G(V, E)$ with capacity function $u(e)$ for each edge e in E. Find the maximum flow f through G.

The flow network $\mathcal{F}_{\mathcal{S},\mathcal{G}}$ corresponding to a grid $\mathcal{G}$ and a set of rectangles $\mathcal{S}$ contains four levels, numbered from top-to-bottom, and will be constructed level-by-level.

Level 1. Contains the source node $\nu^{(1)}$.

Level 2. Contains $\log_\alpha^2 n^c$ nodes. A node $\nu_{i,j}^{(2)}$ at level 2 represents the group $\mathcal{S}_{i,j}$. For each node $\nu_{i,j}^{(2)}$, there is a directed edge from $\nu^{(1)}$ to $\nu_{i,j}^{(2)}$. The capacity of this edge is equal to the number of rectangles in $\mathcal{S}$ that belongs to $\mathcal{S}_{i,j}$.
Level 3. Also contains $\log_\alpha^2 n^c$ nodes. A node $\nu_{p,q}^{(3)}$ on level 3 represents the set of cells in $\mathcal{C}_{p,q}$. For each node $\nu_{p,q}^{(3)}$ there is a directed edge from node $\nu_{i,j}^{(2)}$ to node $\nu_{p,q}^{(3)}$ if and only if $p \geq i$ and $q \geq j$ (or $q \geq i$ and $p \geq j$), i.e., if a rectangle in $\mathcal{S}_{i,j}$ can be packed into a cell in $C_{p,q}$. All the edges from level 2 to level 3 have capacity n.
Level 4. Contains the sink $\nu^{(4)}$. For every node $\nu_{p,q}^{(3)}$ on level 3 there is a directed edge from $\nu_{p,q}^{(3)}$ to $\nu^{(4)}$ with capacity equal to the number of cells in $\mathcal{G}$ that belongs to $\mathcal{C}_{p,q}$.

The following results are straight-forward.

Observation 5 *The maximal grid packing of $\mathcal{S}$ into $\mathcal{G}$ has size k if and only if the max flow in the flow network is k.*

In 1998 Goldberg and Rao [4] presented an algorithm for the maximum flow problem with running time $\mathcal{O}(n^{2/3} m \log(n^2/m) \log U)$. If we apply their algorithm to the flow network we obtain the following lemma followed by the first grid packing theorem.

Lemma 6 *Given a matrix representation of an (α, β, γ)-grid $\mathcal{G}$ and a set $\mathcal{S}$ of n rectangles partitioned into the groups $\mathcal{S}_{i,j}$ w.r.t. their width and height. (1) The size of an optimal packing of $\mathcal{S}$ in $\mathcal{G}$ can be computed in time $\mathcal{O}(\log^9 n)$. (2) An optimal packing can be computed in time $\mathcal{O}(\log^9 n + k)$, where k is the number of rectangles in a grid packing of $\mathcal{S}$ in $\mathcal{G}$.*

Theorem 7 *Given a set of rectangles $\mathcal{S}$ and an $\epsilon > 0$, algorithm* GridPack *produces a grid packing whose area is most $(1+\varepsilon)$ times larger than a minimum area grid packing of $\mathcal{S}$ in time $\mathcal{O}(n^{f(\sqrt{1+\varepsilon})} \log^{19/3} n + n)$, where $f(\chi) = \frac{2c \log(\chi/(\sqrt{\chi}-1))}{\log^2 \chi}$).*

Even though the expression for the running time in the above theorem looks somewhat complicated it is not hard to see that by choosing the value of ε appropriately we obtain that, algorithm GridPack is a PTAS for the MAGP-problem, and if ε is set to be a large constant GridPack produces a grid packing that is within a constant factor of the optimal in linear time. Note also that the approximation algorithm easily can be generalised to d dimensions.

4 Results

The approximation algorithm presented above can be extended and generalised to variants of the basic grid packing problem by performing some small modifications to the procedure TestGrid. We obtain the following corollary.

Corollary 8 *Algorithm* GridPack *is a τ-approximation algorithm with time complexity $\mathcal{O}(n^{f(\sqrt{1+\varepsilon})} \log^{19/3} n + n)$, where:*
- *$\tau = (1+\varepsilon)$ for the problems MAkGP, MAGPAR and MDGP,*
- *$\tau = (1-\varepsilon)$ for the MWP-problem, and*
- *$\tau = ((\lfloor 2(1+\varepsilon) \rfloor)^2)$ for the MNWP-problem*

Finally we consider the hardness of three variants of the MAGP-problem.

Theorem 9 *(1) MNWP cannot be approximated within a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$, unless $P = NP$. (2) The MAGPAR-problem and the MWP-problem are NP-hard.*

5 Acknowledgements

We thank Esther and Glenn Jennings for introducing us to the problem. We would also like to thank Bernd Gärtner for pointing us to "The table layout problem" [1, 3, 5, 6].

References

[1] R. J. Anderson and S. Sobti. The Table Layout Problem. Proc. of SoCG, 1999.

[2] M. Andersson, J. Gudmundsson and C. Levcopoulos. Tech rep., Dept. of Comp. Sci., Lund University, 2002.

[3] R.J. Beach. Setting tables and illustrations with style. PhD thesis, Dept. of computer science, University of Waterloo, Canada, 1985.

[4] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. Journal of the ACM, 45(5):783–797, 1998.

[5] K. Shin, K. Kobayashi and A. Suzuki. TAFEL MUSIK, formatting algorithm of tables. Proc. of Principles of Document Processing, 1994.

[6] X. Wang and D. Wood. Tabular formatting problems. Proc. of Principles of Document Processing, 1996.

Unit Height k-Position Map Labeling

F. Rostamabadi* M. Ghodsi†

Computer Engineering Department
Sharif University of Technology
Tehran, Iran

Abstract

We explore a new variation of k-Position Map Labeling problem, first introduced by Doddi *et. al.* [DMM00] which is: Given a set of points in the plain, a set of up to k allowable positions for each point and, a set of feasible labels (or labeling models) for each position, choose the maximum number of non-intersecting labels so that no point receive more than one label. We define and solve this problem with unit height rectangular labels, in 1D and 2D cases, and also in fixed and slider models.

We present two simple-to-implement 3-approximation algorithms for this problem in both fixed and slider models in 2D case and for unbounded k. We also show that even the problem in 1D is NP-Complete. We then present two 2-approximation algorithms for solving the problem in 1D case.

1 Introduction

Automated label placement is an important problem for map generation in geographical information systems (GIS). The problem is to attach one or more labels (regularly a text) to a point, a line, a curve, or a region in a given map. The point feature label placement has received good attention within computational geometers. Two basic requirements of such labeling are [MS91]: (1) The selected labels should be pair wise disjoint, and (2) Each label should be close enough to the corresponding feature to be identified as such. Other variations of this problem let the features receive more than one labels [KT98, ZQ01], or use specific shapes as labels [DMM+97, vKSW99, SW01]. There are also two labeling models, fixed and slider model. The former is when fixed positions are given as possible label positions [MS91] and the latter is for the cases where the labels can be placed at any position while touching the feature [SvK00, KM00].

Many different approaches have been proposed to solve this problem, including zero-one integer programming [Zor86], approximation algorithms [FW91, DMM+97, ZP00], expert systems [DF89], simulated annealing [ZP00] and force driven algorithms [Hir82].

Doddi *et. al.* [DMM00] introduced the problem of label size maximization k-Position Map Labeling (KPML). In this problem, a set of points in the plain is given and, for each point, we have a set of up to k allowable positions. The problem is to place uniform and non-intersecting labels of maximum size at each point in one of the allowable positions. They have focused on circular labels and have proposed a 3.6-approximation algorithm for it.

Our problem definition is different from that in [DMM00]; we are concerned with selecting the maximum number of labels from a set of feasible labels given for each position. We use nKPML to denote this

*frostam@linux.sharif.edu
†ghodsi@sharif.edu

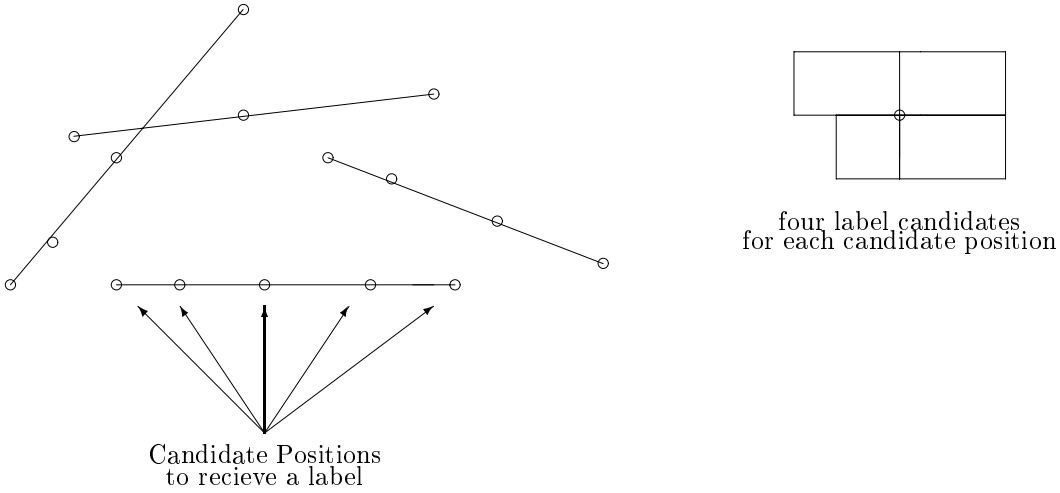


Figure 1: A simple usage of nKPML in line labeling

new problem. More precisely, nKPML is defined as follows: Given a set of points in the plain and a set of up to k allowable positions for each point, and a set of feasible label placements (or a labeling model) for each position, choose the maximum number of non-intersecting labels such that no point receive more than one label.

We focus on nKPML problem with unit height rectangular labels and consider both fixed and slider models. In the fixed model, at each position a finite set of feasible label placements is allowed (see Figure 1).

We will show that this problem is NP-Complete even in 1D case. We propose two different 2-approximation algorithms for both fixed and slider model in 1D case. We then will generalize this solution to obtain 3-approximation algorithms for both models in 2D case.

2 Problem Definition of 2D-nKMPL in Fixed Model

Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of points where each $S_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_{n_i}}\}$ shows all possible positions of point S_i, and let L_{ij} be all unit-height labels for object s_{i_j} of S_i. It is also assumed that

$$\forall_{1 \le i \le n} \cap_{1 \le j \le n_i} L_{i_j} \neq \emptyset$$

(i.e., each pair of labels of a given site must have non-empty intersection.)

The problem is to find the maximum number of labels such that following conditions hold:

- At most one label is chosen from the set $\cup_{1 \le j \le n_i} L_{i,j}$ (for each i).
- All chosen labels should be pair wise disjoint.

This problem is a generalization of the standard point label placement problem, since any instance of label placement problem is in fact an instance of this problem by setting $S_i = \{p_i\}$ for each i, where

$P = \{p_1, p_2, \ldots, p_n\}$ is the set of point locations. Therefore, this problem is also NP-Complete [MS91] and can not be approximated with a factor better than 2 in time $\Omega(n \log n)$ [FW91, Wag94].

2.1 1D-nKPML in Fixed Model

We define 1D-nKPML in fixed model as follows. A set $G = g_1, g_2, \ldots, g_m$ of interval group $g_i = I_{i1}, I_{i2}, \ldots, I_{in_i}$ is given, where each I_{ij} is an interval on x-axis. The problem is to choose at most one interval from every interval group such that all chosen intervals are pair wise disjoint.

The general idea of defining the problem in 1D is to solve the problem in a very special case where there exists a single horizontal line intersecting all possible label positions.

2.2 1D-nKPML Problem is NP-Complete

It can be shown that there is a polynomial time reduction from standard unit-length square labeling in fixed model to 1D-nKPML problem. The standard unit-length labeling problem is:

> Suppose that the point set P is given and we are asked to find the maximum number of disjoint unit-length squares such that each point is adjacent to exactly one square in just one of its vertices and each square is adjacent to exactly one point.

Given an instance of unit-length labeling problem, we define a set of horizontal lines $H_l : y = l$, for all l that there is a label position intersecting H_l. Now, define intersection of label candidates of point p_i with H_j as intervals I_{ij1} and I_{ij2}, and grouping these two intervals into one group G_{i1} of p_i. Since each point in standard unit-height labeling has four label candidates, it is obvious that exactly two horizontal lines H_j and H_{j+1} (for some j) will intersect with all these four label candidates, and two groups G_{i1} and G_{i2} will be generated per point. We merge the horizontal lines H_l into a single horizontal line (e.x. x-axis) with serializing the intervals from left to right i.e. Put intervals of H_{l_1} somewhere on x-axis, then put intervals of H_{l_2} after them, and so on. Now, we have an instance of 1D-nKPML.

It is known that if we restrict the point coordinates to be integers, the problem still remains in NP-Complete. Using this restriction, it is obvious that a solution to unit-length labeling problem can be constructed from a solution of 1D-nKPML which is known to be NP-Complete and a 2-approximation can be achieved in time $\Omega(n \log n)$ [Wag93].

2.3 2-Approximation Algorithm of 1D-nKPML

A simple greedy algorithm can approximate the 1D-nKPML problem within factor two. The main idea to solve the 2D case, is to repeatedly choose the interval with minimum right end point such that a consistent solution is found.

Algorithm Sort all intervals according to their right end point in ascending order and save them in a sorted list I. As long as the sorted list I is not empty, pick the interval I_{ij} with minimum right end point. If the interval I_{ij} has no intersection with previously selected intervals, and if no interval of the group G_i is not selected before, then select interval I_{ij}, otherwise just remove interval I_{ij} from I. □

It is obvious that the running time of the above algorithm is $O(l \log l)$ where l is total number of intervals. The initial sorting of intervals need no more than $O(l \log l)$ and the while loop runs once per interval. Each iteration of while loop requires $O(1)$ to check if the interval I_{ij} is free and another constant time is required to check if the group G_i is already labelled or not. So, the overall running time of the algorithm is $O(l \log l)$.

Lemma 1 *Given an instance of fixed model 1D-nKPML, a solution of size at least $\lambda/2$ can be computed, where λ is the maximum number of intervals in an optimal solution.*

Proof Let S^*, S be an optimal solution and output of our algorithm accordingly. We show that for every selected interval $I_{ij} \in S$ at most two intervals from the optimal solution, S^*, will be missed.

Let $I_{ij} \in S$ be the interval with the minimum right end point, then the first missing label might be $I_{ij'} \in S^*$ which can not be selected since the I_{ij} is selected (Note that $I_{ij'}$ is the optimum interval selected for group G_i in $S*$). The second missing label might be the label with minimum end point in $I_{xy} \in S^*$ (that may have an intersection with I_{ij} and can not be selected anymore).

Now it is obvious that by selecting the interval I_{ij} the algorithm will miss at most two intervals of a given optimal solution. So, the overall approximation factor of this algorithm is two. □

2.4 3-Approximation Algorithm for Unit Height 2D-nKPML in Fixed Model

A simple greedy algorithm can approximate the 2D-nKPML problem within factor three. The main idea is to repeatedly choose the left most label such that a consistent solution is found.

Algorithm Sort all labels according to their right edge in ascending order and save them in a sorted list L. As long as the sorted list L is not empty, pick the first label L_{ij} from the list. If the label L_{ij} has no intersection with previously selected labels, and if no label is assigned to feature S_i, then select label L_{ij}. □

It is obvious that the running time of the above algorithm is $O(l \log l)$ where l is total number of label candidates. The initial sorting of labels needs no more than $O(l \log l)$ and the while loop runs once per label. Each iteration of while loop requires $O(\log l)$ to check if the Label L_{ij} has an intersection with previous selected labels and a constant time is required to check if the feature S_i is already labelled or not. So, the overall running time of the algorithm is $O(l \log l)$.

Lemma 2 *Given an instance of fixed model 2D-nKPML, a solution of size at least $\lambda/3$ can be computed, where λ is the maximum number of labels in an optimal solution.*

Proof Let S^*, S be an optimal solution and output of our algorithm accordingly. We show that for every selected label $L_{ij} \in S$ at most three labels from the optimal solution, S^*, will be missed.

Let $L_{ij} \in S$ be the label with leftmost right edge, then the first missing label might be $L_{ij'} \in S^*$ which can not be selected since the L_{ij} is selected (Note that $L_{ij'}$ is the optimum label selected for feature S_i in $S*$). The next two missing labels might be labels with minimum right edge in $L_{xy} \in S^*$ (that may have an intersection with I_{ij} and can not be selected anymore). Also note that since all labels has unit height, at most two such labels from the may exist that has an intersection with the first leftmost label.

Now it is obvious that by selecting the Label L_{ij}, above algorithm will miss at most three labels from a given optimal solution. So, the overall approximation factor of this algorithm is three. □

3 Problem Definition of 2D-nKMPL in Slider Model

Let $S = \{(S_1, S_2, \ldots, S_n\}$ be a set of points where each $S_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_{n_i}}\}$ shows all possible positions of point S_i, and a slider labeling model (1S, 2S or 4S) is given. The parameter l_{ij} specifying the length label at s_{i_j} is also given. It is also assumed that all labels have unit height and

$$\forall_{1 \le i \le n} \cap_{1 \le j \le n_i} L_{i_j} \neq \emptyset$$

(i.e., each pair of labels of a given site must have non-empty intersection.)

The 2D-nKMPL problem is to find the maximum number of non-intersecting labels such no point receives more than one label.

3.1 3-Approximation Algorithm of 2D-nKPML in Slider Model

A simple greedy algorithm can approximate the 2D-nKPML problem in slider model within factor three. The general idea is the same as the one in slider model algorithm, but with some modifications.

Algorithm Use the slide labeling algorithm [vKSW99] and whenever a label is assigned to a feature, remove other positions of the feature. □

Lemma 3 *Given an instance of slider model 2D-nKPML, a solution of size at least $\lambda/3$ can be computed, where λ is the maximum number of intervals in an optimal solution.*

Proof Since the algorithm of van Kreveld, Strijk and Wolff chooses the leftmost labels first, so the proof will be the same as fixed model. □

4 Conclusion

We considered a new variation of KPML problem introduced in [DMM00]. We show that the problem with unit height rectangular labels is in NP-Complete even in one dimensional case. We then propose two different, yet simple to implement, 2-approximation algorithms for solving the 1D-nKPML in fixed. Generalizing the idea, we obtain two 3-approximation algorithms for 2D-nKPML in both models.

We do not yet know the lower bounds of approximation algorithms and it can be interesting problem to work on.

References

[DF89] Jeffrey S. Doerschler and Herbert Freeman. An expert system for dense-map name placement. In *Proc. Auto-Carto 9*, pages 215–224, 1989.

[DMM+97] Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M.E. Moret, and Binhai Zhu. Map labeling and its generalizations. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 148–157, New Orleans, LA, 4–7 January 1997.

[DMM00] Srinivas Doddi, Madhav V. Marathe, and Bernard M.E. Moret. Point set labeling with specified positions. In *Proc. 16th Annu. ACM Sympos. Comput. Geom. (SoCG'00)*, pages 182–190, Hongkong, 12–14 June 2000.

[FW91] Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom. (SoCG'91)*, pages 281–288, 1991.

[Hir82] Stephen A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[KM00] Gunnar W. Klau and Petra Mutzel. Optimal labelling of point features in the slider model. In D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, editors, *Proc. Sixth Annual International Computing and Combinatorics Conference (COCOON'00)*, volume 1858 of *Lecture Notes in Computer Science*, pages 340–350, Sydney, 26–28 July 2000. Springer-Verlag.

[KT98] Konstantinos G. Kakoulis and Ioannis G. Tollis. On the multiple label placement problem. In *Proc. 10th Canadian Conf. Computational Geometry (CCCG'98)*, pages 66–67, 1998.

[MS91] Joe Marks and Stuart Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard CS, 1991.

[SvK00] Tycho Strijk and Marc van Kreveld. Practical extensions of point labeling in the slider model. Technical Report UU-CS-2000-08, Department of Computer Science, Utrecht University, 2000.

[SW01] Tycho Strijk and Alexander Wolff. Labeling points with circles. *International Journal of Computational Geometry and Applications*, 11(2):181–195, April 2001.

[vKSW99] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.

[Wag93] Frank Wagner. Approximate map labeling is in $\Omega(n \log n)$. Technical Report B 93-18, Fachbereich Mathematik und Informatik, Freie Universität Berlin, December 1993.

[Wag94] Frank Wagner. Approximate map labeling is in $\Omega(n \log n)$. *Information Processing Letters*, 52(3):161–165, 1994.

[Zor86] Steven Zoraster. Integer programming applied to the map label placement problem. *Cartographica*, 23(3):16–27, 1986.

[ZP00] Binhai Zhu and Chung Keung Poon. Efficient approximation algorithms for two-label point labeling. *International Journal of Computational Geometry and Applications*, 2000. To appear.

[ZQ01] Binhai Zhu and Zhongping Qin. New approximation algorithms for map labeling with sliding labels. *Journal of Combinatorial Optimization*, 2001. To appear.

Good NEWS: Partitioning a Simple Polygon by Compass Directions*

Marc van Kreveld
marc@cs.uu.nl

Iris Reinbacher
iris@cs.uu.nl

Institute of Information and Computing Sciences
Utrecht University, The Netherlands

1 Introduction

Nearly all Internet search engines use term matching to create a list of hits and rank the query results. Normally this yields satisfying results. But when a user seeks for "ruins in Eastern Greece" or "castles near Paris", standard search engines are less appropriate. The problem is that concepts like "Eastern Greece" and "near Paris" use a spatial relationship with respect to a geographical object, and the terms "Eastern" and "near" themselves are not relevant for term matching. Such problems have lead to research on geographic information retrieval [7, 8]. Addressing spatial searches on the Internet and building a spatially-aware search engine is the focus of the SPIRIT project [6]. By using geometric footprints associated to Web pages, geographic ontologies, methods to determine spatial relationships, and query expansion, spatial searches can most likely be performed much better.

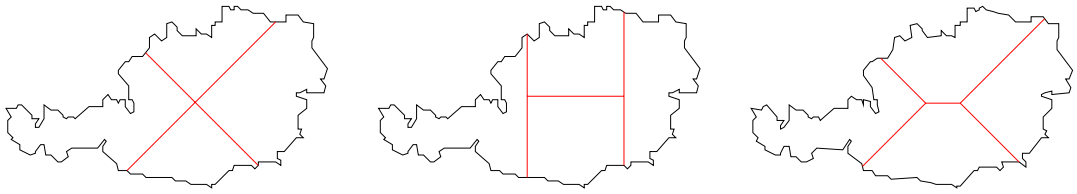


Figure 1: Three partitions of Austria by compass directions.

In this paper we address the problem of dividing a country into four subregions by compass directions (NEWS: North, East, West, South). This is one of the aspects of the SPIRIT project, namely to define and compute spatial relationships that are needed in spatial information retrieval. The partitioning is also useful in geographic user interfaces, where the user can select a region of interest by clicking in the partitioning of the region. The regions of interest, like "the South of Austria", do not have a well-defined boundary, but are used in a more loose sense. This fuzziness of geographic objects due to language issues is well-known [3, 11]. Spatial relationships between two geographic objects that describe proximity or relative position are discussed in [1, 9].

In order to compute a good NEWS partition, we describe an algorithm to determine — for a simple polygon P and a positive real A — all wedges with a given angle and orientation that contain exactly area A of P. We solve the standard version of the wedge problem in optimal $O(n^2)$ time, and the version where the wedge is restricted to its simply-connected part inside P in $O(n \log n)$ time. However, the corresponding NEWS partitions are computed in $O(n \log n)$ time in both cases.

Related in computational geometry is research on area partitionings and continuous ham-sandwich cuts of polygons [2, 5]. Shermer [10] shows how to partition a simple polygon by a vertical line in two equal-area halves in linear time. Díaz and O'Rourke show in [4] how to partition a convex polygon into equal-size parts using an orthogonal four-partition.

*This research is supported by the EU-IST Project No. IST-2001-35047 (SPIRIT).

2 Criteria for a good NEWS partition

There are many criteria one can use to partition a country into four regions by compass directions. Some criteria are especially relevant for the application in query answering of geographic information retrieval whereas others apply more to geographic user interfaces. We list the most important criteria next:

- the relative orientation of any two points should be conserved (no point in North should be farther to the South than any point in South)
- regions should be non-overlapping but adjacent
- all regions should have the same proportion of the area

The first criterion is essential because is contains the essence of the compass directions. The other criteria seem especially important for the user interface application. We would like to develop a simple, efficient algorithm which works well for most countries. With the criteria in mind, certain algorithmic problems can be formulated to find a NEWS partition of a country. They are stated in the following three suggestions. The partitionings for these suggestions are shown in Figure 1.

Suggestion 1 Compute the center of gravity of the polygon and draw two lines with slope $+1$ and -1 through this point.

Suggestion 2 Use horizontal and vertical lines to iteratively cut the polygon into four regions which each cover 25% of the polygon's area.

Suggestion 3 Divide the polygon into four equal-size regions such that the sum

$$\mathfrak{S} = dist_y(C_N, C_P) + dist_y(C_S, C_P) + dist_x(C_E, C_P) + dist_x(C_W, C_P)$$

is maximized. Here C_P denotes the center of gravity of polygon P, and C_N, C_E, C_W, C_S denote the centers of gravity of the North, East, West, and South region. $dist_x$ and $dist_y$ denote the distance by x-coordinate and by y-coordinate.

The center of gravity in Suggestion 1 can be computed in linear time. For Suggestion 2, we let the x- and y-extent determine whether we first split by horizontal or vertical lines. We can apply the algorithm of Shermer [10] three times, which gives a linear time solution.

In this paper, we will focus on finding a NEWS partitioning by an algorithm following the last suggestion. For now we will consider only the more general setting of partitioning the polygon into not necessarily simply connected NEWS regions. We will prove that a NEWS partitioning by Suggestion 3 can only lead to a partitioning with a shape described in Lemma 1.

We will denote with Υ a construction made of a vertical line in the middle and two lines with slope $+1$ and -1 at the top and at the bottom. Similarly, we denote with $\succ\!\prec$ the rotated shape with a horizontal line as middle part. We call the nodes where the three segments meet the **focal points**. In a $\succ\!\prec$ partitioning, we have a West and an East focal point. In a Υ partitioning, we have a North and a South focal point.

Lemma 1 *If an arbitrary simple polygon P is divided into four (not necessarily connected) parts, such that each part covers exactly 25% of the polygon's area and the sum $\mathfrak{S} = dist_y(C_N, C_P) + dist_y(C_S, C_P) + dist_x(C_E, C_P) + dist_x(C_W, C_P)$ is maximized, then it has inner boundaries shaped Υ or $\succ\!\prec$. Here C_P denotes the center of gravity of P,and C_N, C_E, C_W, C_S denote the centers of gravity of the NEWS regions. Furthermore, the partitioning is unique.*

3 NEWS partitions with arbitrary regions

In this section we give an algorithm to compute all wedges with fixed angle and orientation that contain a given area A. We also show how to compute a NEWS partitioning of a simple polygon following Suggestion 3.

Definition 1 *For a simple polygon P, we call a wedge of the form $(y \geq x + a) \cap (y \geq -x + b)$ that contains 25% of the area of P a* **North-wedge** *of P. East-, South-, and West-wedges are defined similarly.*

Definition 2 *The* **North-trace** *is the locus of all points that are apex of a North-wedge. East-, South-, and West-traces are defined similarly.*

The outline of an algorithm to compute a NEWS partitioning according to Suggestion 3 is as follows:

1. Compute the East-trace T_E and the West-trace T_W of polygon P.
2. Scan T_E and T_W simultaneously from top to bottom, to determine if there is a pair of points $p_E \in T_E$ and $p_W \in T_W$ with the same y-coordinate and p_E to the right of p_W (or coinciding) and which gives a North area of 25% of P.
3. If such a pair exists, return the $\succ\!\prec$ partitioning.
4. Otherwise, compute the $\curlyvee$ partitioning similarly.

The algorithm can be implemented to run in $O(n^2)$ time, which will be shown next. For convenience, we rotate the polygon by 45 degrees, so that a West-wedge is now a quadrant of the form $(x \leq a) \cap (y \leq b)$.

Lemma 2 *After rotation, the West-trace is an infinite, continuous, piecewise quadratic curve consisting of $\Theta(n^2)$ pieces in the worst case, and any line with positive slope intersects it only once.*

We now describe the sweep algorithm that computes the West-trace for the rotated polygon. Since the algorithm is the same for any fixed area of P inside the wedge, we set $A = \text{Area}(P)/4$ and give an algorithm to compute all West-wedge positions that have area A inside the intersection of P and the wedge.

We initialize by computing a vertical line that has area A left of it. Then we determine the highest point of P left of or on this vertical line. The highest point is the first break point p_0 of the trace. The initial part of the trace is a vertical half-line down to p_0. We initialize two lists L_x and L_y with all vertices of P sorted on increasing x- and decreasing y-coordinate, starting at p_0. During the sweep we maintain: a balanced binary search tree $\mathcal{T}$ on the edges of P that intersect the sweep line, the leaf in $\mathcal{T}$ that stores the edge of P vertically below the current position of the trace, and the equation of the currently valid curve. The leaves of $\mathcal{T}$ are linked into a list. One of three event-types can occur:

1. The vertical line through the current position of the trace (the sweep line) reaches a vertex of P.
2. The horizontal line through the current position of the trace reaches a vertex of P.
3. The trace reaches the boundary of P.

The next event can be determined in $O(1)$ time from the equation of the curve, the first element in each list, and the edges of P above and below the current position. An event of type 1 requires an update of $\mathcal{T}$ and possibly the pointer to the leaf with the edge below the trace. If the vertex of P is below the position of the trace, we output the next break point and update the equation of the curve. For the second and third type of event similar actions are taken. Note that the sweep also continues if the position of the trace goes outside P. Later it may enter P again. We conclude:

Theorem 1 *Given a simple polygon P with n vertices and a value A, the set of all positions of apexes of wedges with a fixed shape and orientation that contain a portion of area A of P inside can be computed in optimal $O(n^2)$ time.*

Completing the NEWS partitioning by step 2 takes $O(n^2)$ time, so we can compute a NEWS partitioning for Suggestion 3 in $O(n^2)$ time. Actually the running time for computing a trace is $O(n \log n + k)$, where k is the complexity of the trace. In practice k will be much less than quadratic, so the algorithm is efficient on real world data.

We can improve this result to a worst case $O(n \log n)$ time algorithm. We do not compute the whole traces. Instead we determine by binary search and an adaptation of Shermer's algorithm [10] the two vertices of P that have consecutive $(x-y)$-values of their coordinates, and the $\succ\!\prec$ partitioning has its focal points at some $(x-y)$-value in between. Here we can compute the West- and East-trace explicitly by the sweep described above. Now we have at most $O(n)$ events of type 3.

Theorem 2 *Given a simple polygon P with n vertices, we can determine a partitioning by $\succ\!\prec$ or $\curlyvee$ where each region contains exactly 25% of the area in $O(n \log n)$ time. The partitioning maximizes the sum of distances of centers of gravity $\mathfrak{S} = dist_y(C_N, C_P) + dist_y(C_S, C_P) + dist_x(C_E, C_P) + dist_x(C_W, C_P)$.*

4 Extensions

The three suggestions for partitioning algorithms have been implemented (see Figure 1). The results for ten different country outlines can be found in [12].

The idea of partitioning a simple polygon into four equal-size not necessarily simply-connected parts, while maximizing the sum-of-center-of-gravity distances immediately generalizes to polygons that are not simply connected. The following extensions have been studied more thoroughly, see [12]:

Fair partitioning: Find a simply-connected partition into four regions by $\rightarrowtail\!\!\!\prec$ or $\curlyvee$ such that each region has 25% of the area, or decide that no such partition exists.
Maxmin: Find a simply-connected partition by $\rightarrowtail\!\!\!\prec$ or $\curlyvee$ that maximizes the area of the smallest region.
Minmax: Find a simply-connected partition by $\rightarrowtail\!\!\!\prec$ or $\curlyvee$ that minimizes the area of the largest region.

The fair partitioning uses a modified version of the sweep algorithm. Because of the simply-connectedness of the partition itself we can show the following:

Theorem 3 *For a simple polygon P with n vertices, we can determine in $O(n\log n)$ time if there exists a simply-connected $\rightarrowtail\!\!\!\prec$ or $\curlyvee$ partitioning into four simply-connected regions that all have 25% of P's area.*

For the Maxmin and Minmax partitioning the following holds:

Theorem 4 *For a simple polygon P with n vertices, we can compute a simply-connected $\rightarrowtail\!\!\!\prec$ and $\curlyvee$ partitioning that minimizes the maximum area subregion or maximizes the minimum area subregion in $O(n^3)$ time.*

Another extension is to define a center region of a country as well. By restating Suggestion 3 as partitioning a polygon into five regions, each with 20% of the area and applying the same proof ideas, we can show that the center is an axis-parallel rectangle and the other boundaries are lines with slope $+1$ or -1. It can also be useful to define a degree of North, East, West and South. This could be done simply by coordinates or by growing the center region of the polygon.

References

[1] A.I. Abdelmoty and B.A. El-Geresy. An intersection-based formalism for representing orientation relations in a geographic database. In *2nd ACM Workshop on Advances in GIS*, pages 44–51, 1994.

[2] P. Bose, J. Czyzowicz, E. Kranakis, D. Krizanc, and A. Maheswari. A note on cutting circles and squares in equal area pieces. In *Proc. FUN with Algorithms '98*, 1998.

[3] P.A. Burrough and A.U. Frank, editors. *Geographic Objects with Indeterminate Boundaries*, volume II of *GISDATA*. Taylor & Francis, London, 1996.

[4] M. Díaz and J. O'Rourke. Ham-sandwich sectioning of polygons. pages 282–286, 1990.

[5] Susan Hert. Connected area partitioning. In *Abstracts 17th European Workshop Comput. Geom.*, pages 35–38. Freie Universität Berlin, 2001.

[6] C.B. Jones, R. Purves, A. Ruas, M. Sanderson, M. Sester, M.J. van Kreveld, and R. Weibel. Spatial information retrieval and geographical ontologies – an overview of the spirit project. In *Proc. 25th Annu. Int. Conf. on Research and Development in Information Retrieval (SIGIR 2002)*, pages 387–388, 2002.

[7] R. Larson. Geographic information retrieval and spatial browsing. In L.C. Smith and M. Gluck, editors, *Geographic Information Systems and Libraries: Patrons, Maps, and Spatial Information*, pages 81–124. 1996.

[8] M. Li, S. Zhou, and C.B. Jones. Multi-agent systems for web-based map information retrieval. In *Proc. 2nd GIScience*, number 2478 in Lect. Notes in Computer Science, pages 161–180, 2002.

[9] D.J. Peuquet and Z. Ci-Xiang. An algorithm to determine the directional relationship between arbitrarily-shaped polygons in the plane. *Pattern Recognition*, 20(1):65–74, 1987.

[10] T. C. Shermer. A linear algorithm for bisecting a polygon. *Information Processing Letters*, 41:135–140, 1992.

[11] L. Talmy. How spoken language and signed language structure space differently. In *Proc. COSIT 2001*, number 2205 in Lect. Notes in Computer Science, pages 274–262, 2001.

[12] M. van Kreveld and I. Reinbacher. Good NEWS: Partitioning a simple polygon by compass directions. Manuscript, 2002.

Significant-Presence Range Queries in Categorical Data

(extended abstract)

Mark de Berg* Herman J. Haverkort†

Abstract

In traditional colored range-searching problems, one wants to store a set of n objects with m distinct colors for the following queries: report all colors such that there is at least one one object of that color intersecting the query range. Such an object, however, could be an 'outlier' in its color class. Therefore we consider a variant of this problem where one has to report only those colors such that at least a fraction τ of the objects of that color intersects the query range, for some parameter τ. We present efficient data structures for such queries with orthogonal query ranges in sets of colored points, and for point stabbing queries in sets of colored rectangles.

1 Introduction

Motivation. The range-searching problem is one of the most fundamental problem in computational geometry. In this problem we wish to construct a data structure on a set S of objects in $\mathbb{R}^d$, such that we can quickly decide for a query range which of the input objects it intersects. The range-searching problem comes in many flavors, depending on the type of objects in the input set S, on the type of allowed query ranges, and on the required output (whether one wants to report all intersected objects, to count the number of intersected objects, etc.). The range-searching problem is not only interesting because it is such a fundamental problem, but also because it arises in numerous applications in areas like databases, computer graphics, geographic information systems, and virtual reality. Hence, it is not surprising that there is an enormous literature on the subject—see for instance the surveys by Agarwal [1], Agarwal and Erickson [2], and Nievergelt and Widmayer [4].

In this paper, we are interested in range searching in the context of databases. Here one typically wants to be able to answer questions like: given a database of customers, report all customers whose ages are between 20 and 30, and whose income is between \$50,000 and \$75,000. In this example, the customers can be represented as points in $\mathbb{R}^2$, and the query range is an axis-parallel rectangle.[1] This is called the (planar) *orthogonal range-searching problem*, and it has been studied extensively.

There are situations, however, where the data points are not all of the same type but fall into different categories. Suppose, for instance, that we have a database of stocks. Each stock falls into a certain category, namely the industry sector it belongs to—IT, energy, banking, food, chemicals, etc. Then it can be interesting for an analyst to get answers to questions like: "In which sectors companies had a 10–20% increase in their stock values over the past year?" In this simple example, the input data can be seen as points in 1D (namely for each stock its increase in value), and the query is a one-dimensional orthogonal range-searching query.

Now we are no longer interested in reporting all the points in the range, but only the categories that have points in the range. This means that we would like to have a data structure whose query time is not sensitive to the total number of points in the range, but to the total number of

*Department of Computer Science, TU Eindhoven, P.O.Box 513, 5600 MB Eindhoven, the Netherlands.

†Institute of Information and Computing Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands. The work by H.H. is supported by the Netherlands' Organization for Scientific Research (NWO).

[1] From now on, whenever we use terms like "rectangle" or "box" we implicitly assume these are axis-parallel.

categories in the range. This can be achieved by building a suitable data structure for each category separately, but this is inefficient if the number of categories is large. This has led researchers to study so-called *colored range-searching problems*: store a given set of colored objects—the color of an object represents its category—such that one can efficiently report those colors that have at least one object intersecting a query range [3, 5, 6].

We believe, however, that this is not always the correct abstracted version of the range-searching problem in categorical data. Consider for instance the stock example sketched earlier. The standard colored range-searching data structures would report all sectors that have *at least one* company whose increase in stock value lies in the query range. But this does not necessarily say anything about how the sector is performing: a given sector could be doing very badly in general, but contain a single 'outlier' whose performance has been good. It is much more natural to ask for all sectors for which *most* stocks, or at least a significant portion of them, had their values increase in a certain way. Therefore we propose a different version of the colored range-searching problem: given a fixed threshold parameter τ, with $0 < \tau < 1$, we wish to report all colors such that at least a fraction τ of the objects of that color intersect the query range. We call this a *significant-presence query*, as opposed to the standard *presence query* that has been studied before. (We also have some results on the case where τ is not fixed beforehand, but part of the query. Due to lack of space, these results are omitted.)

Problem statement and results. We study significant-presence queries in categorical data in two settings: orthogonal range searching where the data is a set of colored points in $\mathbb{R}^d$ and the query is a box, and stabbing queries where the data is a set of colored boxes in $\mathbb{R}^d$ and the query is a point. In this extended abstract, we only discuss our results on orthogonal range searching. We also omit several of the proofs.

Let $S = S_1 \cup \cdots \cup S_m$ be a set of n points in $\mathbb{R}^d$, where m is the number of different colors and S_i is the subset of points of color class i. Let τ be a fixed parameter with $0 < \tau < 1$. We are interested in answering significant-presence queries on S: given a query box Q, report all colors i such that $|Q \cap S_i| \geq \tau \cdot |S_i|$. For $d = 1$, we present a data structure that uses $O(n)$ storage, and that can answer significant-presence queries in $O(\log n + k)$ time, where k is the number of reported colors. Unfortunately, for $d \geq 2$, we have not been able to design a data structure using near-linear storage with logarithmic query time for this problem. As a data structure with quadratic or more storage is prohibitive in practice, we study an approximate version of the problem. More precisely, we study *ε-approximate significant-presence queries*: here we are required to report all colors i with $|Q \cap S_i| \geq \tau \cdot |S_i|$, but we are also allowed to report colors with $|Q \cap S_i| \geq (1-\varepsilon)\tau \cdot |S_i|$, where ε is a fixed positive constant. For such queries we have developed a data structure that uses only $O((1/(\tau\varepsilon)^{2d-1}m)^{1+\delta})$ storage, for any $\delta > 0$, and that can answer queries in $O(\log n + k)$ time, where k is the number of reported colors. Note that the amount of storage does not depend on n, the total number of points, but only on m, the number of colors. This should be compared to the results for the previously considered case of presence queries on colored points sets. Here the best known results are: $O(n)$ storage with $O(\log n + k)$ query time for $d = 1$ [6], $O(n \log^2 n)$ storage with $O(\log n + k)$ query time for $d = 2$ [6], $O(n \log^4 n)$ storage with $O(\log^2 n + k)$ query time for $d = 3$ [5], and $O(n^{1+\delta})$ storage with $O(\log n + k)$ query time for $d \geq 4$ [3]. Note that these bounds all depend on n, the total number of points; this is of course to be expected, since these results are all on the exact problem, whereas we allow ourselves approximate answers.

2 Orthogonal range queries

One of the difficulties in significant-presence queries is that the problem is not readily decomposable: we cannot decide whether a color is significantly present in a range Q if we just know whether or not certain subsets of that color are significantly present in Q. In this respect, standard presence queries are easier: a color is present in Q iff a subset of that color is present in Q. Hence, our approach is to first reduce significant-presence queries to standard presence queries. We do this by introducing so-called *test sets*.

Test sets for orthogonal range queries Let S be a set of n points in $\mathbb{R}^d$, and let τ be a fixed parameter with $0 < \tau < 1$. A set T of boxes—that is, axis-parallel hyperrectangles—is called a *τ-test set* for S if the following holds:

- any box from T contains at least τn points from S;
- any query box Q that contains at least τn points from S fully contains at least one box from T.

This means that we can answer a significant-presence query on S by answering a presence query on T: a query box Q contains at least τn points from S if and only if it contains at least one box from T. We did not yet reduce the problem to a standard presence-query problem, because T contains boxes instead of points. However, we can map the set T of boxes in $\mathbb{R}^d$ to a set of points in $Reals^{2d}$, and the query box Q to a box in $\mathbb{R}^{2d}$, in such a way that a box $b \in T$ is fully contained in Q if and only if its corresponding point in $Reals^{2d}$ is contained in the transformed query box.[2] This means we can apply the results from the standard presence queries on colored point sets.

It remains to find small test sets. As it turns out, this is not possible in general: below we show that there are point sets that do not admit test sets of near-linear size. Hence, after studying the case of exact test sets, we will turn our attention to approximate test sets.

Exact test sets. It is easy to see that any minimal box containing at least τn points from S—that is, any box b containing at least τn points from S such that there is no box $b' \neq b$ with $b' \subset b$ and containing τn or more points—must be a box in T, and that the collection of all such minimal boxes forms a τ-test set. Hence, the smallest possible test set consists exactly of these minimal boxes. In the 1-dimensional case a box is a segment, and a minimal segment is uniquely defined by the point from S that is its left endpoint. This means that any set of n points on the real line has a test set of size $(1 - \tau)n + 1$. Unfortunately, the size of test sets increases rapidly with the dimension, as the next lemma shows.

Lemma 2.1 *For any set S of n points in $\mathbb{R}^d$, there is a τ-test set of size $O(\tau^{d-1}n^{2d-1})$. Moreover, there are sets S for which any τ-test set has size $\Omega(\tau^{d-1}n^{2d-1})$. (proof omitted from this abstract)*

Approximate test sets. The worst-case bound from Lemma 2.1 is quite disappointing. Therefore we now turn our attention to approximate test sets: a set T of boxes is called an *ε-approximate τ-test set* for a set S of n points if

- any box from T contains at least $(1 - \varepsilon)\tau n$ points from S;
- any query box Q that contains at least τn points from S fully contains at least one box from T.

This means we can answer ε-approximate significant-presence queries on S by answering a presence query on T.

Lemma 2.2 *For any set S of n points in $\mathbb{R}^d$ ($d > 1$) and any ε with $0 < \varepsilon < 1/2$, there is an ε-approximate τ-test set of size $(2d-1)^{2d-1}/(\varepsilon^{2d-1}\tau^{2d-2})$. Moreover, there are sets S for which any ε-approximate τ-test set has size $\Omega((1/\varepsilon)^{2d-1}(1/\tau)^d)$.*

Proof. We will prove an upper bound of $((2d-1)/(\varepsilon\tau))^{2d-1}$ here; an improvement of a factor τ can be attained with a divide-and-conquer variation of the construction described below. Due to lack of space, we omit the details of the divide-and-conquer approach from this extended abstract.

To prove the upper bound of $((2d-1)/(\varepsilon\tau))^{2d-1}$, we proceed as follows. We construct a collection H_1 of $(2d-1)/(\varepsilon\tau)$ hyperplanes orthogonal to the x_1-axis, such that there are $\varepsilon\tau n/(2d-1)$ points of S between any pair of consecutive hyperplanes.[3] We do the same for the other axes, obtaining sets $H_2, \ldots, H_d$ of hyperplanes. From these collections of hyperplanes we construct our test set as follows. Take any possible subset H^* of $2d-1$ hyperplanes from $H_1 \cup \cdots \cup H_d$ such that H_1 up to H_{d-1} each contribute exactly two hyperplanes to H^*, and H_d contributes one hyperplane. Let $b(H^*)$ be the smallest box that is bounded by the hyperplanes from H^*, contains

[2] In fact, the transformed box is unbounded to one side along each coordinate-axis, so it is a d-dimensional 'octant'.

[3] If there are more points with the same x_1-coordinate, we have to be a bit careful. The details are omitted from this extended abstract.

exactly $(1-\varepsilon)\tau n$ points from S, and lies above the hyperplane we picked from H_d. If $b(H^*)$ exists, we add it to our test set T. Clearly, the size of T is at most $((2d-1)/(\varepsilon\tau))^{2d-1}$.

We now argue that T is an ε-approximate τ-test set for S. By construction, every box in T contains at least $(1-\varepsilon)\tau n$ points, so it remains to show that every box Q that contains at least τn points from S fully contains at least one box from T. To see this, observe that for any i with $1 \leq i \leq d$, there must be a hyperplane $h_1^{(i)} \in H_i$ that intersects Q and has at most $\varepsilon\tau n/(2d-1)$ points from $Q \cap S$ 'below' it. Similarly, there is a hyperplane $h_2^{(i)} \in H_i$ intersecting Q with at most $\varepsilon\tau n/(2d-1)$ points from $Q \cap S$ 'above' it. Note that $h_2^{(i)} \neq h_1^{(i)}$. Now consider the box b bounded by the hyperplanes in the set $H^* := \{h_1^{(1)}, h_2^{(1)}, \ldots, h_1^{(d-1)}, h_2^{(d-1)}, h_1^{(d)}\}$ and unbounded in the positive x_d-direction. The number of points from S in $b \cap Q$ is at least $|Q \cap S| - (2d-1) \cdot \varepsilon\tau n/(2d-1) \geq (1-\varepsilon)\tau n$. Hence, the box $b(H^*) \in T$ is not larger than $b \cap Q$ and, hence, it is contained in Q.

The lower-bound construction is omitted in this extended abstract. □

Putting it all together. To summarize, the construction of our data structure for ε-approximate significant-presence queries on $S = S_1 \cup \cdots \cup S_m$ is as follows. We construct an ε-approximate τ-test set T_i for each color class S_i. This gives us a collection of $O(1/(\varepsilon^{2d-1}\tau^{2d-2})m)$ boxes in $\mathbb{R}^d$. We map these boxes to a set P of colored points in $\mathbb{R}^{2d}$, and construct a data structure for the standard colored range-searching problem (that is, presence queries) on P, using the techniques of Agarwal et al. [3]. Their structure was designed for searching on a grid, but using the standard trick of normalization—replace every coordinate by its rank, and transform the query box to a box in this new search space in $O(\log n)$ time before running the query algorithm—we can employ their results in our setting.

The same technique works for exact queries, if we use exact test sets. This gives a good result for $d = 1$, if we use the results from Gupta *et al.* [5] on quadrant range searching.

Theorem 2.1 *Let $S = S_1 \cup \cdots \cup S_m$ be a colored point set in $\mathbb{R}^d$, and τ a fixed constant with $0 < \tau < 1$. For $d = 1$, there is a data structure that uses $O(n)$ storage such that exact significant-presence queries can be answered in $O(\log n + k)$ time, where k is the number of reported colors. For $d > 1$, there is, for any ε with $0 < \varepsilon < 1/2$ and any $\delta > 0$, a data structure for S that uses $O((1/(\varepsilon^{2d-1}\tau^{2d-2})m)^{1+\delta})$ storage such that ε-approximate significant-presence queries on S can be answered in $O(\log n + k)$ time.*

Acknowledgements We thank Joachim Gudmundsson and Jan Vahrenhold for inspiring discussions about the contents of this paper.

References

[1] P.K. Agarwal. Range Searching. In: J. Goodman and J. O'Rourke (Eds.), *CRC Handbook of Computational Geometry*, CRC Press, pages 575–598, 1997.

[2] P.K. Agarwal and J. Erickson. Geometric range searching and its relatives. In: B. Chazelle, J. Goodman, and R. Pollack (Eds.), *Advances in Discrete and Computational Geometry*, Vol. 223 of *Contemporary Mathematics*, pages 1–56, American Mathematical Society, 1998.

[3] P.K. Agarwal, S. Govindarajan, and. S. Muthukrishnan. Range searching in categorical data: colored range searching on a grid. In *Proc. 10th Annu. European Sympos. Algorithms* (ESA 2002), pages 17–28, 2002.

[4] J. Nievergelt and P. Widmayer. Spatial data structures: concepts and design choices. In: J.-R. Sack and J. Urrutia (Eds.) *Handbook of Computational Geometry*, pages 725–764, Elsevier Science Publishers, 2000.

[5] J. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. In *Proc. 3rd Workshop on Algorithms and Data Structures*, LNCS 709, pages 361–373, 1993.

[6] R. Janardan and M. Lopez. Generalized intersection searching problems. *Internat. J. Comput. Geom. Appl.* 3:39–70 (1993).

Efficient Contour Tree Construction and Computation of Betti Numbers in Scalar Fields

Tobias Lenz* Günter Rote*

Submitted to the *19th European Workshop on Computational Geometry, 2003*

Abstract

A new algorithm to construct contour trees is introduced which improves the runtime of known approaches. It also generates additional topological information about the data which can be used to compute the Betti numbers for all possible level sets.

1 Introduction

Visualizing Contours in Scalar Fields. A commonly used technique to store large amounts of data is in a scalar field. This data could be measurement results in any dimension, e.g. elevation information in geographic information systems. The visualization of the data is usually done by drawing level sets which are points of equal value. A connected component in a level set is called *contour*. In two dimensions the scalar value can be interpreted as the height over a two-dimensional domain. In general the domain should be given as a simplicial complex and the scalar values are interpolated over discreet values at the vertices.

The Contour Tree. To grant high speed for interactive systems drawing contours, an efficient structure is needed. Quickly drawing a level set includes the knowledge of how many connected components the system has to draw and where they are. It is sufficient then to have a single simplex through which the requested contour passes, a so called *seed*. By recursively checking the simplices in its neighborhood the whole connected component of the level set can be traced.

The contour tree is helpful for creating a sufficient set of seeds. Every node in the contour tree represents a point at a level where the number of components changes. Every edge in the contour tree spans the interval between the values of the represented points of the two incident nodes and represents a connected component. This works for any dimension.

Betti Numbers. In some applications topological information about the surface is needed, e.g. the Betti numbers. In topology the i-th Betti number β_i is defined as the rank of the i-dimensional homology group. For a d-dimensional object the Betti numbers $\beta_d, \beta_{d+1}, \ldots$ are all zero. In this paper the Betti numbers are only computed for up to three dimensions, therefore we only care about $\beta_0, \beta_1, \beta_2$. They have a very graphic meaning: β_0 is the number of connected components, β_1 is the number of tunnels through them and β_2 is the number of enclosed voids.

Previous Work. Carr et al. [1] presented a fast sweep algorithm. They do two simple sweeps over the data in order of increasing and decreasing scalar values to create two trees. In a third step these trees are combined to the final contour tree. The required time is $O(N + n \log n)$ for any dimension where N denotes the size of the simplicial complex and n the number of vertices.

Pascucci [2] augmented the contour tree for three

*Institut für Informatik, Freie Universität Berlin, Takustraße 9, 12247 Berlin, Germany,
tlenz, rote@inf.fu-berlin.de

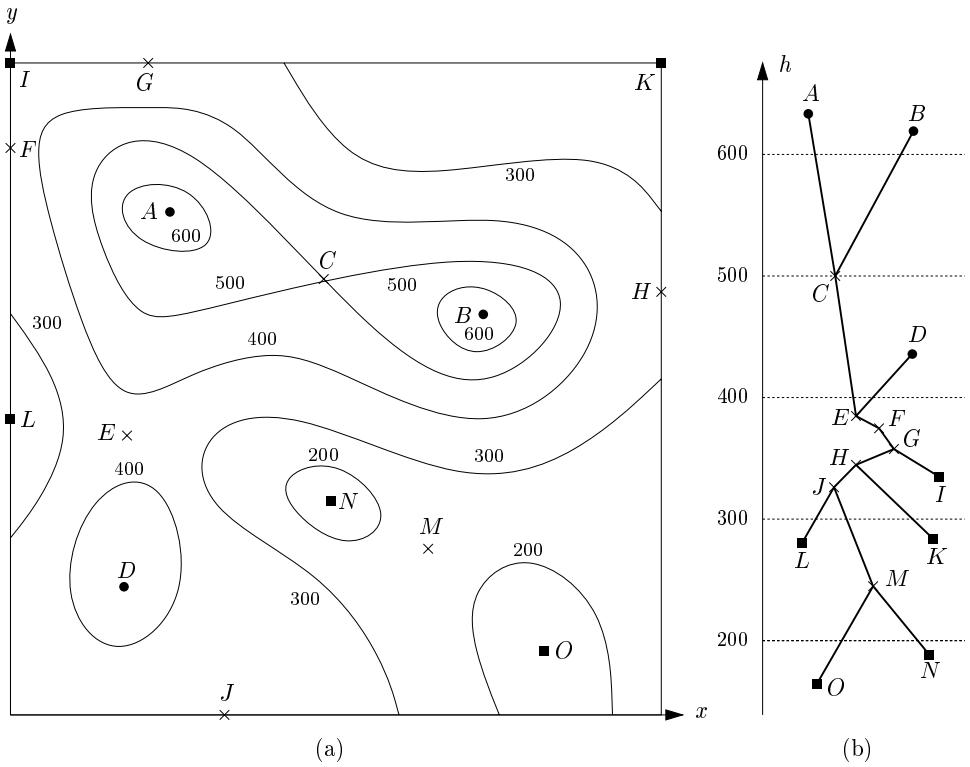


Figure 1: A contour map and the corresponding contour tree. Minima and maxima are indicated by squares and circles, and crosses denote saddle points.

dimensional meshes with the Betti numbers in additional $O(N \log N)$ time. The contour tree already gives β_0. β_2 can be obtained by checking the boundary — a closed surface encloses a void, an open one does not. β_1 can than be obtained by computing the Euler characteristic χ and than solving the formula $\chi = \sum_{i=0}^{d-1} (-1)^i \beta_i$.

2 Definitions

The Input. The input for the algorithm consists of a d-dimensional simplicial complex S which forms a bounded volume $D \subset \mathbb{R}^d$. The scalar field is represented by a function $f : D \to \mathbb{R}$ which assigns a scalar value to every vertex in S and interpolates the rest of D linearly.

N denotes the number of d-dimensional simplices, so the size of the input is $O(N)$. The number of vertices in S is denoted by n. In d dimensions $N = O\left(n^{\lceil d/2 \rceil}\right)$ but for "nicely shaped" D, a small triangulation can be found such that $N = O(n)$.

It is assumed for simplicity that the values of f are distinct over all vertices in S. The same offect can easily be achieved e.g. by ordering points lexicographically after their coordinates if they have the same function value.

Critical Points. In this paragraph the term *neighborhood* $N(v)$ of a vertex v denotes the subgraph of S induced by the vertices in S adjacent to v with v being excluded. Let $N_+(v)$ and $N_-(v)$ be the subgraphs of $N(v)$ induced by the vertices w with $f(w) > f(v)$,

$f(w) < f(v)$ respectively. $C_+(v), C_-(v)$ denote the number of connected components of $N_+(v), N_-(v)$.

A *local maximum* obviously only has lower neighbors, therefore $C_+ = 0$, while a *local minimum* only has higher neighbors so $C_- = 0$. A *regular point* v is a point with $C_+(v) = C_-(v) = 1$. All other points are *saddle points*. The saddle points together with the local extrema are the *critical points*. Topological changes, including changes in the number of components, only occur at critical points.

Using these definitions, it is possible to determine the type for every point in S locally by scanning its neighborhood. This takes $O(N)$ time.

3 The Contour Tree

Join Tree and Split Tree. The upper level set, the lower level set, and the (equality) level set of f at value h are defined as

$f_>(h) := \{ x \in D \mid f(x) > h \}$,
$f_<(h) := \{ x \in D \mid f(x) < h \}$,
$f_=(h) := \{ x \in D \mid f(x) = h \}$.

A *contour* is a connected component of a level set $f_=(h)$. Sweeping through the data by decreasing h, the set $f_>$ grows while $f_<$ shrinks continuously, and $f_=$ is their common boundary. The *join tree* represents the evolution of the components of the set $f_>(h)$ as h varies. The *split tree* is defined similarly for sweeping by increasing h.

Looking at the components of $f_>(h)$ as h decreases the following events can happen.

(a) A new component may appear, starting out at a local maximum.

(b) Several components may merge into a single one at a saddle point.

(c) There may be topological changes which do not change the number of components

Also, events of types (b) and (c) may occur together.

Join Tree Construction. Assume all saddle points are known and they are sorted by their function value. We scan all saddle points v in decreasing order of $f(v)$. For each v, we select a neighbor w in each of the $C_+(v)$ components of $N_+(v)$. We process each w by starting a monotone increasing path at w, continuing until we get stuck in a local maximum or we hit a previously visited vertex.

To maintain the connected components a UNION-FIND data structure is used with the set of saddle points as the ground set. If a monotone path hits a vertex z which was already visited, we FIND its component. If z is not already in the same component as v, we add an edge from v to the lowest vertex in the component of z to the join tree and we perform the UNION of the components containing v and z.

The case when a monotone path ends in a local maximum r is easy: We simply add an edge from v to a new vertex representing r to the join tree.

It may happen that a saddle point receives only one outgoing edge in the join tree because it just reflects a change in topology and not in connectivity (like a change from a torus to a sphere). We leave these vertices of degree two in the tree during the construction, and can eliminate them in a final purification step after combining the join and split tree to the contour tree.

Theorem 1 *Creating the join tree takes time for saddle points identification, for sorting and for the monotone paths. This is in total* $O(N + t \log t + m + s\alpha(s,t))$ *time for* t *saddle points,* m *denoting the number of edges in* S *and* $s = \sum^t C_+(v)$ *is the sum over all higher neighboring components of all saddles. This simplifies to* $O(N + t \log t)$.

The theorem holds for the split tree respectively.

Combining Two Trees. The Algorithm to combine the join tree with the split tree is explicitly shown in [1] and it takes $O(t)$ time for t nodes in the final contour tree.

4 Computing Betti Numbers

Contour Tree with All Critical Points. Usually the contour tree only gives information about connectivity which is sufficient for drawing connected

components. In three and higher dimensions saddle points exist which do not change the number of connected components. Following the algorithm in section 3 these saddle points are nodes in the contour tree with degree two and in this section the tree containing all critical points is refered to as *ECT* — extended contour tree.

Algorithmic Idea. The following only holds for up to three dimensions. The ECT provides all points which cause changes in topology and also the information on which connected component the change occurs. This is sufficient for calculating the Betti numbers for every possible level set.

Assume the critical points v_i ordered by decreasing function value. The Betti numbers for every interval $(f(v_i), f(v_{i+1}))$ are stored in a tree structure with the following simple loop.

The nodes in the ECT are processed from top to bottom. For every node the type is determined and the Betti numbers for the next interval are changed according to that type.

For every node in the ECT the type is known after the construction of the ECT. Possible types are local maximum or local minimum, a saddle uniting two components in the join tree (*join*) or uniting a component with itself (*pseudo-join*). The same types of saddle points can occur in the split tree denoted by *split* and *pseudo-split*.

Betti Numbers for Volumes. The object of interest may be the volumetric object $\{x \in D \mid f(x) \geq w\}$ instead of the surface $f^{-1}(w) = \{x \in D \mid f(x) = w\}$. In this case, parts once connected always stay connected.

For volumetric objects the edges do not always correspond to the number of components. The edges from a split are complementary components also known as voids.

In a pre-processing step, every leaf in the ECT corresponding to a boundary vertex in S has to be removed if the neighbor has a higher function value. This has to be done iteratively, as new such leafs may appear during the process. This removes the topological changes on the "outside" of the object.

The following changes occur at critical points sweeping from higher to lower function values.

Local maximum v_i: A new component starts at $f(v_i)$ and at $f(v_i) - \varepsilon$ it becomes a solid ball. This implies to increment β_0 and β_1, β_2 are not changed.

Join: Several components are connected. β_0 decreases by the number of edges to nodes with higher function value. β_1, β_2 are not changed.

Pseudo-join: A component is connected to itself (maybe several times) which creates one or more "handles" and thereby increases the number of tunnels. β_1 is increased by the number of how many times the component is connected to itself.

Pseudo-split: A tunnel or several ones through the object close. This is the analogous case to a pseudo-join, so β_1 is decreased in the same fashion.

Split: A ball in the complement is split into several ones. The resulting number of voids is one for each neighboring node with lower function value.

Local minimum v: A void becomes very small for $f(v)+\varepsilon$, is only a point at $f(v)$ and vanishes at $f(v)-\varepsilon$. Therefore β_2 decreases by one.

Betti Numbers for Surfaces. Every closed surface contains a void, therefore a component usually increases the number of components and the number of voids. A surface with boundary is open and therefore does not contain a void. A surface can only be open if it touches the boundary of D. This is easy to check if the contour tree does not only contain all critical points but also special boundary vertices of S. This increases the runtime of the contour tree algorithm.

References

[1] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *11th ACM/SIAM Symposium on Discrete Algorithms*, pages 918–926, 2000.

[2] Valerio Pascucci. On the topology of the level sets of a scalar field. In *Lawrence Livermore National Laboratory technical report UCRL-JC-142262*, February 2001.

Author Index