Faculty of Mathematics, Informatics and Mechanics of Warsaw University
Interdisciplinary Centre for Mathematical and Computational Modelling
Stefan Banach Centre of Excellence
Fundation for Information Technology Development

Proceedings of the Eighteenth European Workshop
on Computational Geometry

April 10-12, 2002
Institute of Informatics
Warsaw University
Poland
Foreword

The papers in this collection were presented at the Eighteenth European Workshop on Computational Geometry, held in Warsaw, Poland, April 10-12, 2002. The workshop was organized at the Faculty of the Mathematics, Informatics and Mechanics of the Warsaw University in cooperation with the Interdisciplinary Centre for Mathematical and Computational Modelling, the Stefan Banach Centre of Excellence and the Foundation for Information Technology Development. The assistance of the Department of Computer Science of the University of Kentucky is also acknowledged.

The goal of this annual workshop is to bring together the researchers and students from academia and industry interested in Computational Geometry and related fields and to promote - in a relaxed and informal atmosphere - discussion and diffusion of the most recent work, leading to the establishment of new collaborations and research projects.

Following the tradition of the workshop, many of the included papers represent reports of continuing research, and it is expected that most of them will appear in their complete and final versions in scientific journals. Selected papers from the workshop will appear in a special issue of the "Computational Geometry: Theory and Applications". We thank everyone who responded to the Call for Papers and wish to thank Drs. Herbert Edelsbrunner, Zhiguniew Marciniak, and Gert Vegter for giving invited plenary lectures.

The organizers are very grateful to the sponsoring institutions and the numerous individuals who helped in organizing and running the workshop. The experience and advice of the chairs of the previous meetings of the European Workshop on Computational Geometry, Dr. Helmut Alt (17th EWCG, Berlin, Germany), Dr. Matt Katz and Dr. Klara Kedem (16th EWCG, Elk, Israel), were extremely useful to our work.

Jerzy W. Jaromczyk
Mirosław Kowaluk

April 2002, Warszawa
Table of Contents

Preliminary Program

Invited Lectures

Bio-Geometric Modeling
Herbert Edelsbrunner, Duke University .................................................. 1

Units in Group Rings of Crystallographic Groups
Zbigniew Marciniak, Warsaw University ..................................................... III

Evolution of apparent contours
Gert Vegter, University of Groningen ........................................................ V

Contributions

Three Dimensional Euclidean Voronoi Diagrams of Lines with a Fixed Number of Orientations
V. Koltun, M. Sharir ................................................................. 1

An exact predicate for the optimal construction of the Additively Weighted Voronoi diagram
F. Anton, J.-D. Boissonnat, D. Mioc, M. Yvinec ........................................ 4

The weighted farthest color Voronoi diagram on trees and graphs
F. Hurtado, V. Sacristán, R. Klein, E. Langetepe ........................................ 8

Vertex $\gamma$-guards in Simple Polygons
B. Speckmann, C.D. Tóth ............................................................... 12

The Orthogonal Fortresses Problem with Floodlights
A. Spillner, H.-D. Hecker ................................................................. 16

Rectilinear Trees under Rotation and Related Problems
B.K. Nielsen, P. Winter, M. Zachariasen ................................................... 18

Alternating Paths through Disjoint Line Segments
M. Hoffmann, C.D. Tóth ................................................................. 23

A simple kinetic visibility polygon
S. Hornus, C. Puech ................................................................. 27

A kinetic view of the shooter problem
J.W. Jaromczyk, M. Kowaluk ................................................................. 31

Balanced Partition of Minimum Spanning Trees
M. Andersson, J. Gudmundsson, Ch. Levcopoulos, G. Narasimhan ........... 36

Approximating the Geometric Minimum-Diameter Spanning Tree
J. Gudmundsson, H. Haverkort, S.M. Park, Ch.-S. Shin, A. Wolff ............. 41

Drawing Free Trees Inside Rectilinear polygons Using Polygon Skeleton
A. Bagheri, M. Razzazi ................................................................. 46

Optimal Projections onto Grids
J.M. Díaz-Báñez, F. Hurtado, M.A. López, J.A. Sellarés ......................... 51
Efficient Construction of the Union of Geometric Objects
E. Ezra, D. Halperin, M. Sharir ........................................ 56
Exact \textit{l}_\infty Nearest Neighbor Search in High Dimensions
L. Heinrich-Litan ...................................................... 61
Euclidean position in 2-orbifolds
C. Cortés, A. Márquez, J. Valenzuela .............................. 65
Optimal Polygonal Schema on an Orientable Surface
E. Colin de Verdiere, F. Lazarus ..................................... 68
Grid representations of graphs on surfaces
N. de Castro ............................................................ 72
Flat embeddings of certain 2-complexes
C. O'Dunlaing .......................................................... 75
On Flips in Polyhedral Surfaces: a new development
L. Alou, R. van Damme .............................................. 80
Computing the symmetries of non-convex polyhedral objects in 3-space
P. Braes, Ch. Knauer ............................................... 84
Chromatic Variants of the Erdos-Szekeres Theorem on Points in Convex Position
O. Devillers, F. Hurtado, C. Seara ................................ 87
On the Crossing Number of Complete Graphs
O. Aichholzer, F. Aurenhammer, H. Krasser .................... 90
Planar graphs and metrically complete graphs
J. Cáreres, C.I. Grima, A. Márquez, A. Moreno-González .... 93
On Simplifying Dot Maps
M. de Berg, P. Bose, O. Cheong, P. Morin ....................... 96
Improved Output-Sensitive Construction of the Vertical Decomposition of Three-Dimensional Arrangements
H. Shaul, D. Halperin ................................................. 101
A better approximation algorithm for covering polygons with squares
G. Zwoźniak ............................................................... 106
Approximation by skin curves
N. Krüthof, G. Vegter ................................................ 109
An Application of the Free Form Deformation to Phytoplankton Cells Modeling
A.M. Lyakh .......................................................... 112
Isomorphic-free generation of some classes of triangulations without repetitions
L. Alou, A. Netchaev ................................................ 116

Author Index

List of Participants
Wednesday, April 10

8:00-8:50 Coffee - Registration

8:50-9:00 Opening

9:00-9:20 V. Koltun, M. Sharir
Three Dimensional Euclidean Voronoi Diagrams of Lines with a Fixed Number of Orientations

An exact predicate for the optimal construction of the Additively Weighted Voronoi diagram

9:40-10:00 F. Hurtado, V. Sacristan, R. Klein, E. Langetepe
The weighted farthest color Voronoi diagram on trees and graphs

10:00-10:15 Coffee break

10:15-10:35 B. Speckmann, C.D. Tóth
Vertex γ-guards in Simple Polygons

10:35-10:55 A. Spilker, H.-D. Hecker
The Orthogonal Fortresses Problem with Floodlights

10:55-11:15 B.K. Nielsen, P. Winter, M. Zachariasen
Rectilinear Trees under Rotation and Related Problems

11:15-11:30 Coffee break

11:30-11:50 M. Hoffmann, C.D. Tóth
Alternating Paths through Disjoint Line Segments

11:50-12:10 S. Hornus, C. Puech
A simple kinetic visibility polygon

12:10-12:30 J.W. Jaromczyk, M. Kowaluk
A kinetic view of the shooter problem

12:30-14:30 Lunch break

14:30-15:30 Invited talk:
Herbert Edelsbrunner, Duke University
Bio-Geometric Modeling

15:30-15:45 Coffee break

15:45-16:05 M. Andersson, J. Gudmundsson, Ch. Levcopoulos, G. Narasimhan
Balanced Partition of Minimum Spanning Trees
Invited Lectures
Bio-Geometric Modeling

Herbert Edelsbrunner
Computer Science and Mathematics,
Duke University and Raindrop Geomagic,
Research Triangle Park, North Carolina, USA
edels@cs.duke.edu

Molecules have for ever been modeled geometrically, either as stickdiagrams, emphasizing the covalent bonds between atoms, or as spacefilling diagrams, representing the space they occupy. This talk aims at further developing the geometric view of the molecular world. It introduces the filtration of alpha complexes, which are combinatorial objects dual to the space filling diagrams obtained by continuously growing the atom balls. These combinatorial objects lead to fast and robust algorithms for visualization and analysis.

We demonstrate that each space filling diagram is connected the same way as its dual complex, meaning the two are homology equivalent. We can therefore express the connectivity of the space filling diagram by the homology groups of the dual complex. We also introduce persistent homology groups to capture the scale-dependence of a topological feature.

We show that the dual complex of a space filling diagram can be used to compute the volume and surface area without constructing the diagram. Similarly, it can be used to compute the weighted area derivative of the surface, which is believed to have a significant contribution to the force that drives the folding process simulated by molecular dynamics.
Units in Group Rings of Crystallographic Groups

Zbigniew Marciniak
Institute of Mathematics,
Warsaw University, Poland
zbimar@impan.pl

A group ring $RG$ is made up of two ingredients: a group $G$ and a commutative ring $R$. It consists of finite formal sums $\sum r_g g$ with $r_g \in R$ and $g \in G$. We add such sums in an obvious way to multiply, we just open the brackets and multiply the monomials $r_g g$ and $s_h h$ by the rule $(r_g s_h) (gh)$.

Group rings play a very important role in representation theory of groups, algebraic number theory and topology. One of the important aspects of studying group rings is the description of its invertible elements (units), i.e., elements $a, b \in RG$ such that $ab = ba = 1$.

In particular, algebraists try to describe the units of $ZG$ for crystallographic groups $G$. These groups are classical objects, since long very important in mathematics, physics and, of course, in crystallography. They are defined as discrete, compact subgroups of the group of isometries of the Euclidean space.

In the talk we shall outline a method of study of units in $ZG$, using a computational approach. In particular, we shall present some achievements from the past as well as a couple of open problems in this area which could be attacked by constructing smart algorithms.
Evolution of apparent contours

Gert Vegter
Department of Mathematics and Computing Science,
University of Groningen, The Netherlands
gert@cs.rug.nl

The contour generator of a smooth surface, under parallel or perspective projection, locally separates backward and forward facing regions on the surface. It consists of those points on the surface where the normal is perpendicular to the direction of projection. The apparent contour is the projection of the contour generator onto the image plane. See Figure 1. The contour generator and the apparent con-

Figure 1: The contour generator and the apparent contour. The red segments correspond to tangent lines of the surface, parallel to the direction of projection. The locus of the points of tangency is the contour generator, its projection onto the view plane is the apparent contour.
tour play are important visibility features of a smooth surface, e.g., in connection with visualization. Koenderink [Koe90] describes in a very intuitive style the geometric features of curves and surfaces relevant for geometric computing. The silhouette of a surface is the curve in the view plane that separates the projected surface from the background. It is a subset of the apparent contour, that plays a prominent role in non-photorealistic rendering, cf., [BH98] and silhouette clipping [SGG+00]. In computer vision [CG00] techniques have been developed for partial reconstruction of the geometry of a surface from a sequence of apparent contours corresponding to a discrete set of projection directions.

Generically, the apparent contour is a smooth curve consisting of several components, and the contour generator consists of regular points and isolated cusp points, like the sharp peak-shaped points in the rightmost column of Figure 2. Moreover, the apparent contour may have self-intersections, which correspond to distinct points on the contour generator being projected to the same point on the apparent contour.

Figure 2: A break-to-break bifurcation. Top row: a sequence of views of a smooth surface. Middle row: the corresponding apparent contours. Bottom row: blow up of the apparent contour near the bifurcation event.

The connected components of the contour generator may merge or split when the projection direction changes, or when we rotate or deform the surface. Such visual events, or singularities, correspond to non-generic views of the surface, but are unavoidable if the position or shape of the surface changes continuously. Other types of visual events are related to the birth or death of components of the apparent contour.

Algorithms computing the apparent contour are reported to crash if the the evolution of the surface gives rise to the occurrence of these visual events. Therefore, a better understanding of these events is a first step towards a more robust computation of apparent contours. The classification of the various types of visual events, as well as the derivation of simple local models of the surface exhibiting these events, has been achieved using sophisticated methods from singularity theory and differential geometry. In particular, these mathematical methods do not transfer directly to robust numerical computation. An accessible description of the singularity theory behind this classification is contained in [Bru84].

In this talk we review an algorithm for the computation of the apparent contour of implicit surfaces, and describe its generic evolution as the direction of projection changes over time. In particular, we obtain approximate local models describing the generic visual events of evolving surfaces. Furthermore, we discuss how to detect the occurrence of these events for evolving implicit surfaces. Details are described in a forthcoming technical report [SV02].

References


Contributions
Three Dimensional Euclidean Voronoi Diagrams of Lines with a Fixed Number of Orientations

Vladlen Koltun
School of Computer Science,
Tel Aviv University,
Tel Aviv 69978, Israel
vladlen@tau.ac.il

Micha Sharir
School of Computer Science,
Tel Aviv University,
Tel Aviv 69978, Israel;
Courant Institute of Mathematical Sciences,
New York University,
New York, NY 10012, USA.
sharir@cs.tau.ac.il

ABSTRACT
We show that the combinatorial complexity of the Euclidean Voronoi diagram of a lines in $\mathbb{R}^3$ that have at most $c$ distinct orientations, is $O(c^{\frac{5}{2}}\log c)$, for any $c \geq 6$. This result is a step towards proving the longest pending conjecture that the Euclidean Voronoi diagram of lines in three dimensions has near-quadratic complexity. It provides the first natural instance in which this conjecture is shown to hold. In a broader context, our result adds a natural instance to the (rather small) pool of instances of general 3-dimensional Voronoi diagrams for which near-quadratic complexity bounds are known.

1. INTRODUCTION
The study of Voronoi diagrams in the plane has been very extensive over the past 20 years, and the structure of such diagrams is by now thoroughly understood. The study has covered diagrams for many kinds of sites, and for many kinds of metrics or distance functions, and has also considered various variations of the problem, such as intersection diagrams, constrained Delaunay triangulations, and more. Surveys of the state of the art are given in [3, 6].

In contrast, Voronoi diagrams in three and higher dimensions have been much less studied, and many basic problems are still wide open. Most variations of planar Voronoi diagrams have linear complexity, which is usually a consequence of the planarity of the diagram. In three dimensions, a pre-

The Eighteenth European Workshop on Computational Geometry
April 10-12, 2002, Warsaw, Poland
2. TWO OR THREE ORIENTATIONS

Let \( R \) be a set of \( n \) lines in \( \mathbb{R}^3 \), which have up to three distinct orientations. Thus \( R \) can be written as \( R = R \cup G \), where all the lines in \( R \) (called "red lines") have the same orientation, and the same holds for the lines of \( B \) ("blue lines") and those of \( G \) ("green lines").

We begin by bounding the number of versions of the Voronoi diagram \( \text{Vor}(L) \) of \( L \). Let \( \xi \) be a version of the diagram, incident to the cells of four lines \( \ell_1, \ell_3, \ell_4, \ell_5 \). At least two of them must be of the same color, suppose \( \ell_1, \ell_3, \ell_4, \) and let \( R \subseteq R \). Project the lines of \( R \) onto a plane \( \pi \) not tangent to the lines of \( R \). Then such lines cut a fourgon into a point, and \( e \) projects onto a vertex \( e' \) of the plane Voronoi diagram of the projected points within \( \pi \). The number of such vertex projections is at most \( 2n \), as every fourgon \( e \) can project onto the same point \( e' \) at most \( 2n \). This is because the radius of the ball centered at \( e' \) and touching \( e_1, e_3, e_4, e_5 \) equals the radius of the disk (within \( \pi \)) centered at \( e' \) and touching the projections of those three lines. As we did this ball parallel to the lines of \( R \), we reach at most \( 2n \) placements where it touches another line, giving rise to the Voronoi vertices of the kind under consideration.

The overall number of such vertices is \( \Theta(n^2) \).

Suppose then that exactly two of the four lines are of the same color, say \( \ell_1, \ell_3, \) and let \( R \subseteq R \). We project the lines of \( R \) onto the plane \( \pi \), and let \( e_1, e_3, e_4, e_5 \) be the intersection of \( e_4 \) and \( e_5 \) in \( \pi \). We obtain that the projection of \( e_4, e_5 \) on a Voronoi edge of the planar diagram of the point project lines of the red lines. The number of such edges in \( \Theta(n^2) \).

Fix such an edge \( e \) and consider the 3-dimensional slice \( \Sigma_e \) defined as the union of all lines incident to \( e \) and perpendicular to \( e \) by construction, \( e \in \Sigma \). Moreover, \( e \) is the locus of all centers of balls that touch \( e_1, e_3, e_4, e_5 \) and no other red line. Let \( H_e \) denote the plane containing \( e \) and \( e_1, e_3, e_4, e_5 \). Let \( e_3-\ldots-\ell_4 \) be the midline of the 2-dimensional slice spanned by \( e_1, e_3, e_4, e_5 \). Denote the two halflines bounded by \( e_1, e_3, e_4, e_5 \) and their endpoints by \( e_1, e_3, e_4, e_5 \). See Figure 1.

For a point \( p \in \Sigma \), consider the line \( e \) passing through \( p \) in the plane \( H_e \) and orthogonal to \( e \). Parameterize \( e \) by a linear parameter \( s \), where \( 0 \leq s \leq 1 \) within \( e_1, e_4 \). Let \( \psi(s) \) denote the distance between \( e_1, e_4 \), which is a quadratic polynomial in \( s \).


Figure 1: The bisector of \( \ell_1 \) and \( \ell_3 \).


3. FOUR ORIENATIONS

Theorem 3.1. The complexity of the Voronoi diagram of a set of \( n \) lines with at most four distinct orientations is \( O(n^2) \), for any \( \alpha > 0 \), where the constant of proportionality depends on \( \alpha \).

4. MORE THAN FOUR ORIENATIONS

The case of an arbitary (but constant) number of orientations is easy to handle using Theorem 3.1, by noting that any version of the full Voronoi diagram \( \text{Vor}(L) \) is also a vertex of the diagram of the set of all lines whose orientations are equal to the (at most) four orientations of the lines whose cells are incident to \( e \). Hence, by applying Theorem 3.1 to each of the \( \Theta(n^2) \) quadrangulations of the orientations, we obtain the main result of the paper:

Theorem 4.1. The combinatorial complexity of the Voronoi diagram of \( n \) lines in \( \mathbb{R}^3 \), for any \( \alpha > 0 \), where the constant of proportionality depends on \( \alpha \).
An exact predicate for the optimal construction of the Additively Weighted Voronoi diagram

Extended Abstract

Francois Anton
University of British Columbia
Department of Computer Science
201-2366 Main Mall, Vancouver, B.C., Canada, V6T 1N4
fanton@cs.ubc.ca

Jean-Daniel Boissonnat, Darka Mioc, Mariette Yvinec
INRIA Sophia Antipolis
2004, route des Lucioles, BP 93
06902 Sophia Antipolis Cedex, France
Jean-Daniel.Boissonnat@sophia.inria.fr, dmio@cs.ubc.ca,
Mariette.Yvinec@sophia.inria.fr

Keywords
Additively Weighted Voronoi diagram, Power Voronoi diagram, algebraic predicates

1. INTRODUCTION

The ordinary Voronoi diagram has been extended or generalized in several directions, and those generalized Voronoi diagrams have found many practical applications (see [2] for a general survey on Voronoi diagrams). Abstract Voronoi diagrams [5] include a large number of specific Voronoi diagrams, e.g., Voronoi diagrams for point sites under any $L_p$ optics, or under any convex distance function, whose unit circle is algebraic. Furthermore, they comprise weighted Voronoi diagrams and Voronoi diagrams for line segments or circles under the Euclidean metric.

The weighted Voronoi diagrams (additively, multiplicatively, combined, etc.) differ from the ordinary Voronoi diagram in that the generators do not all have the same weight and the distance depends on these weights. The Additively Weighted Voronoi diagram is defined using the additively weighted distance: $d_{sw}(p, y) := |x - x_p| - w_i$, where $x$ is the position vector of $p$, $x_p$ is the position vector of the point $p_i$, and $w_i$ is the weight of the point $p_i$.

The algorithm proposed by Aurenhammer [1] is based on the relation between the Additively Weighted Voronoi diagram in dimension $d$ and the Power Voronoi diagram in dimension $d + 1$.

In this work, we have developed an exact algebraic predicate and an optimal algorithm for constructing the Additively Weighted Voronoi diagram. The vertices of the weighted point in the lower envelope of the set of cones $(C_i)$. Then, it is easy to show that the contribution of such one $C_i$ to this lower envelope is just the intersection of the cone $C_i$ with the region of the sphere $S$ in the Power Voronoi diagram of the set $(C_i)$. In the next section we review the relation between the Additively Weighted Voronoi diagram and the Power Voronoi diagram. In Section 3, we introduce the algorithm based on this relation. In Section 4, we explore the algebraic predicate that is the core of the algorithm presented in Section 3.

2. PRELIMINARIES

We place ourselves in the Euclidean plane $E^2$. The algorithm uses the mapping $\phi$ of the weight circle (centered at $(x, y)$ and radius $w$) corresponding to a weighted point $P_i$ with coordinates $(x, y)$ and weight $w_i$ to the sphere $S$ of center $M$ with coordinates $(x, y, w)$ and radius $r_i = \sqrt{w_i}$.

In Figure 1: The cone inscribing the sphere image of the weighted point

The $\phi$ vertex angle cone $C_i$ that inscribes the weighted sphere $S$ has its apex at $(x, y, w)$ (see Fig. 1). The vertical projection $\pi(M)$ of a point $M$ of $E^2$ on the cone $C_i$ is the image by the map of the circle of center $M$ and tangent to the weight circle associated with $P_i$. The signed distance from $M$ to the weighted point $P_i$. Then, the additively Voronoi diagram of the set $(P_i, w_i)$ is the projection onto $E^2$ of the lower envelope of the set of cones $(C_i)$. Then, it is easy to show that the contribution of such one $C_i$ to this lower envelope is just the intersection of the cone $C_i$ with the region of the sphere $S$ in the Power Voronoi diagram of the set $(C_i)$. The second vertex can be computed in a similar way.

3. THE ALGORITHM

The algorithm consists of three $\delta$ steps: first compute all the spheres $S$ for each $i$, compute the Power Voronoi diagram of these spheres, and finally for each $i$, compute the intersection with the cone $C_i$ of the cell of the Power Voronoi diagram that corresponds to $S_i$.

To compute, the intersection of the power cell of $(S_i)$ with the cones $C_i$, we propose to consider in turn each edge of this cell. Each edge of the Power Voronoi diagram is determined by five spheres. The algorithm is based on a predicate that determines if such an edge intersect the cone $C_i$ and the number (at most two) of intersection points if any.

The worst case running time of the algorithm is the same as the one of the Power Voronoi diagram, i.e., $O(n^3)$ (see [3] for a description of the algorithm for constructing the Power Voronoi diagram and its optimality). However, as we will see it in the next section, this algorithm requires algebraic predicates of degree 16 only.

4. THE ALGEBRAIC PREDICATE FOR THE CONSTRUCTION OF THE ADDITIVELY WEIGHTED VORONOI DIAGRAM

This predicate is based on the intersection of the cone inscribing a weighted sphere resulting from the mapping of a weighted point to the 3D space with an edge of its Power Voronoi cell. Each edge of the Power Voronoi cell connects two Power Voronoi vertices and involves five spheres. Each vertex is at the center of the circle orthogonal to four spheres. The centres of these spheres form a tetrahedron.

4.1 The Power Voronoi vertices

An edge between the vertices corresponding to two adjacent tetrahedra is determined by the four vertices of one of the two adjacent vertices of the Power Voronoi cell. For the computations of the first vertex, the first four weighted points are needed. In 3D space they form a tetrahedron.

Let $S_1(x_1, y_1, w_1, \sqrt{w_1})$, $S_2(x_2, y_2, w_2, \sqrt{w_2})$, $S_3(x_3, y_3, w_3, \sqrt{w_3})$, and $S_4(x_4, y_4, w_4, \sqrt{w_4})$ be the four spheres defining the tetrahedron.

The intersection of these linearly independent radical planes gives the four Power Voronoi vertices:

\[
\begin{align*}
S_1 &= \left\{ (x, y) \mid -2(x - x_1) -2(y - y_1) -2(w_1 - w_i) \right\} \\
S_2 &= \left\{ (x, y) \mid -2(x - x_2) -2(y - y_2) -2(w_2 - w_i) \right\} \\
S_3 &= \left\{ (x, y) \mid -2(x - x_3) -2(y - y_3) -2(w_3 - w_i) \right\} \\
S_4 &= \left\{ (x, y) \mid -2(x - x_4) -2(y - y_4) -2(w_4 - w_i) \right\}
\end{align*}
\]

where $D = \left\{ (x, y, w) \mid -2(x - x_1) -2(y - y_1) -2(w_1 - w_i) \right\}$.

The coordinates of the first Power Voronoi vertex are:
The degree of the numerator of $\Delta$ is 16. Since the common denominator is a perfect square, $(4D^3)$, the sign of $\Delta$ is the sign of its numerator. Thus, its evaluation requires the evaluation of a polynomial of degree 16 in the input (the coordinates of the sphere centers and the sphere radii).

The equation in $(x, y, z)$ of the algebraic cone corresponding to the weighted point $(a, b, c)$ (where $(a, b, c)$ is one of the three sphere centers) is:

$$g = (x-a)^2 + (y-b)^2 + (z-c)^2 = 0.$$  

Thus, the intersection of the straight line defined by the two Voronoi vertices and the cone is given by the following quadratic equation in $\lambda$:

$$g(\lambda') = (\lambda - (\lambda' - x)^2) + (y - (\lambda' - y)^2) + (z - (\lambda' - z)^2) = 0.$$  

This equation can be written in the following form:

$$\lambda' = 2A + 2B + C = 0$$

where $A = (a - x)^2 + (b - y)^2 + (c - z)^2$, $B = (a - x)(b - y) + (a - x)(c - z) + (b - y)(c - z)$, $C = (b - y)(c - z) + (a - x)^2 + (b - y)^2 + (c - z)^2$, and $\Delta = 4B - 4AC = (B - AC)$.  

Let $\lambda = \lambda' - x$, $\lambda = \lambda' - y$, $\lambda = \lambda' - z$, $d = \lambda - x$, $e = \lambda - y$, and $f = \lambda - z$. Then, $A = \lambda^2 + \lambda d + \lambda e + \lambda f + d e + e f + f d + d^2 + e^2 + f^2$, $B = d e + e f + f d$, and $C = d^2 + e^2 + f^2$. Then, by developing and simplifying, we can rewrite the discriminant of the quadratic equation as follows:

$$B^2 = (AC - \lambda d^2 - \lambda e^2 - \lambda f^2 - 2\lambda d e - 2\lambda d f - 2\lambda e f - 2\lambda d f + 2\lambda d^2 + 2\lambda e^2 + 2\lambda f^2).$$

Let $\Delta = 2A + 2B + C = 0$ and $g = 2A + 2B + C = 0$. The left hand side equation 1 has the same sign $A$ outside the roots of that equation. $C$ is the left hand side of relation 1 when $D = 1$. When $A = 1$ and $B = 2C$, the left hand side of this equation is less than or equal to 1 when $A = 1$. When $A 

5. CONCLUSIONS

We have provided an exact predicate for the computation of Additively Weighted Voronoi diagrams in the plane. The degree of this predicate is 8. Even though the worst case running time of the algorithm may not be optimal if we consider algebraic computations as O1 running time complexity, the algebraic complexity of this algorithm is optimal (degree 16 predicate). The algebraic predicates involved in order to construct incrementally the Additively Weighted Voronoi diagram seem to have a higher algebraic complexity. This is an open problem to know if degree 16 algebraic predicates are sufficient to incrementally construct the Additively Weighted Voronoi diagram.

6. ACKNOWLEDGMENTS

The first author received a research support from the French Government (BDF) and from the Institut National de Recherche en Informatique et en Automatique (INRIA).

7. REFERENCES


Translated from the 1995 French original by Hervé Brönnimann.

The weighted farthest color Voronoi diagram on trees and graphs

[Extended Abstract]

Ferran Hurtado, Vera Sacristán
Dept. de Matemàtica Aplicada II
Univ. Politecnica de Catalunya, Barcelona, Spain

ABSTRACT

Let a point site be situated on the vertices or edges of a geometric graph \( G \) over \( n \) edges. Each site can be assigned a multiplicative weight and a color. We discuss the complexity, and provide efficient algorithms for the construction, of the Voronoi diagram in which each point of \( G \) belongs to the region of that site whose color is the closest of the farthest color. Special algorithms are presented for the cases when all weights are identical, when all colors are identical, when each graph is a tree, or when \( G \) is a tree.

Keywords

Voronoi diagram, graph, network, local planning.

1. INTRODUCTION

The Voronoi diagram is one of the most interesting structures in computational geometry; many variants and applications have been described in the surveys [1, 2, 3]. It decomposes a given space, \( G \), into Voronoi regions, one to each element \( p \) of a given site set, \( S \), such that the region of site \( p \) contains all points of \( G \) that are closer to \( p \) than to any other site in \( S \). In most cases, \( G \) is a space of dimension at least 2.

Some variants of the Voronoi diagram are of particular interest to applications in local planning. First, each site \( p \) can be assigned a positive weight factor, \( w(p) \), to model its importance. The distance from \( p \) to some point \( x \) is then the standard distance \( d(p, x) \) in space \( G \), times \( w(p) \). (Note that a bigger weight factor yields a smaller Voronoi region.) For the Euclidean plane \( G \), Aurenhammer and Edelsbrunner [2] have shown that the weighted Voronoi diagram of \( n \) point sites can be of complexity \( O(n^2) \), and provided an optimal algorithm.

The second variant applies to a situation where each of the \( n \) sites represents one of \( k \) different types of facilities, like schools, shopping malls, hospitals, etc. A person choosing a new residence might want to have one representative site of each type as close as possible. This problem can be solved by means of the farthest color Voronoi diagram. Let us assume that each site \( i \) is colored according to its type. For each point \( x \) in the plane, let \( c(i, x) \) denote the closest site of color \( i \). Then, by definition, \( x \) belongs to the Voronoi region of the site among \( c(1, x), c(2, x), \ldots, c(k, x) \) that is furthest away from \( x \). This distance is minimized by a circle centered at a point on the Voronoi diagram.

For the Euclidean plane, Klein and Langetepe [7] have shown how to construct this diagram in time \( O(n \log n) \). Abudllamon et al. [4] gave efficient algorithms for finding the smallest rectangle, or a parallel strip or rectangle containing a point of each color, that were not based on Voronoi diagrams.

In this paper we study both, the weighted and the colored variant, and their combination, in a two-dimensional setting, where \( G \) is a geometric tree or graph. So far, only the standard (i.e., unweighted and uncolored) Voronoi diagram on graphs has received attention. In their monograph [8], Garey and Johnson introduce the standard Voronoi diagram and discuss its structure. Among their papers, they cite Hikdim and Elbassioni [9] who gave an \( O(m \log m) \) algorithm based on shortest path computations; here \( \log m \) denotes the number of vertices, respectively. If the sites are vertices of the graph, Mehlhorn [10] avoids the factor \( \log m \) by a more efficient way of using Dijkstra's shortest path algorithm, and Renegar [11] improves on this by giving an \( O(m \log \log m) \) algorithm. Throughout this paper, we consider a geometric tree or graph \( G \) of \( n \) vertices over \( c \) edges, and \( n \) point sites that are placed on edges or vertices of \( G \). Just for simplicity we assume that \( G \) is planar, and that its edges are straight line segments.

2. WEIGHTED FARthest COLOR

VORONOI DIAGRAM ON A TREE

Consider a tree \( T = (V, E, \ell) \) of \( m \) edges, where \( V \) is the set of vertices, \( E \) the set of edges, and \( \ell : E \to \mathbb{R}^+ \). For any graph, \( E + \ell \) is a directed graph. The weighted Euclidean distance from \( x \) to \( y \) along the unique path that connects the two points.

Given a set \( S = \{p_1, \ldots, p_n\} \) of \( n \) points on the tree, we define the Voronoi diagram of \( S \) on the tree, \( VD(T, S) \), as a partition \( \{V_{\ell}(p_1), \ldots, V_{\ell}(p_n)\} \) of \( T \), where the Voronoi region of each site \( p_i \) is the set \( V_{\ell}(p_i) \) of all the points \( x \) of \( T \) that are closer to \( p_i \) than to any other site \( p_j \), in terms of the tree distance \( d \). We consider such Voronoi diagram completely described if the following information is known:

- The set of bisector points, i.e., all the boundary points between adjacent Voronoi regions.
- For each bisector point, its location in the tree (the edge to which it belongs, or the vertex it coincides with), the sites of its adjacent Voronoi regions, and the distance to them.
- For each vertex of the tree, \( (x, y) \) closest site(s) and its distance to it (chem).

Theorem 1. The Voronoi diagram of a set of \( n \) points on a tree of size \( m \) has complexity \( n + m - 1 \) and can be computed in optimal time \( O(m \log m + n \log n) \) time.

Sketch of the proof. The complexity of the diagram follows from the fact that the Voronoi regions are connected. The construction of the diagram can be obtained by the following algorithm:

1. Sort the sites on each edge, producing the Voronoi partition of each edge between its extreme sites, if there are more than two sites on some edge the two extreme sites are relevant for the next steps.
2. Compute the spanning tree \( S_T \) of the remaining sites in \( T \). Let \( T' \) be the result of the following operation: First we remove all maximal subtrees of \( T - S_T \); by the hanging node \( h(x) \) of a we denote the unique node \( x \) that belongs to \( S_T \). Next we replace every maximal chain of edges of \( S_T \) that contains only degree-2 vertices by a single edge of the same length.

3. Partition the remaining sites into two sets of about the same cardinality and compute their Voronoi diagrams on the contracted tree \( T' \) by divide and conquer.

4. Traverse the original tree, placing the bisectors in their real position, and assign each subtree to the Voronoi region of its hanging node.

The correctness of the algorithm is based on the connectivity of the Voronoi regions, and its complexity analysis strongly relies on the fact that only \( O(n) \) sites survive step 1, and that the contracted tree has complexity \( O(n) \).

Observations. Notice that for \( m = O(1) \) the previous theorem matches the known results for the Voronoi diagram of \( n \) sites on a line.

We now consider the weighted Voronoi diagram \( VD(T, S) \), where each site \( p_i \) of \( G \) pays a weight \( w(p_i) \), a multiplicative factor to its associate distance function. In other words, we can define weighted Voronoi regions as \( VW(R) = \{ x \in T \mid d_G(x, p_i) \leq w(p_i), i \in \{1, \ldots, k\} \} \neq \emptyset \). Note that the weighted Voronoi regions do not need to be connected and subtrees do not necessarily belong to the same weighted Voronoi region as their hanging node. Therefore we need more information in the description of the diagram, i.e., for each edge we keep the bisector or points sorted by their order on the edge.

Theorem 2. The weighted Voronoi diagram of a set of \( n \) points on a tree of size \( m \) has complexity \( O(n \log m) \) and can be computed in \( O(n \log n + m \log m) \) time.

Sketch of the proof. The complexity of the diagram follows from the known fact that the complexity of the diagram on a line is \( O(n) \). In fact, a linear number of bisector points may
The upper bound is obtained by applying Corollary 6.3 of [11] to each edge of the tree.

The algorithm that constructs the diagram combines the $k$-monotone weighted Voronoi diagrams.

**Observation:** When all the sites are of the same type ($k = 1$) this result matches that of Theorem 2. On the other hand, for the case of the tree being a line ($m = |Q| + 1$), there is still space for a small improvement, since the running time of the algorithm turns out to have an extra $o(k)$ factor.

3. **EXTENSION TO GRAPHS**

In case of graphs things are a bit different. For example, the shortest path between two points is no longer unique. Despite such effects similar arguments can be applied.

Let $n$ denote the number of nodes of a graph $G = (V, E)$. The Voronoi diagram on graphs has complexity $O(n^3)$. Let $m$ denote the number of edges of an edge $i$. Each edge is divided into at most $m + 2$ pieces; two bisectors may have sites on either side. Summing up yields $\sum (m + 2) = n + 3k$.

For computing the diagram we again apply a divide and conquer approach. Divide the set of edges into two subsets of about the same cardinality, compute their Voronoi diagrams on the graph. Fortunately, the merge step can be done in $O(n^2)$ time. The size of one of the diagrams is at a vertex, i.e., it is the closest site to the vertex. This closest site may be a new bisector with some sites of the other diagram. Fortunately, this can be done vertex by vertex, every edge may be concerned.

Similar techniques and adaptations work also for the more complicated cases giving rise to the results stated in Table 1.

4. **REFERENCES**


Vertex π-guards in Simple Polygons

Bettina Speckmann *
Institute for Theoretical Computer Science
ETH Zürich, CH-8092 Zürich, Switzerland
speckmann@inf.ethz.ch

Csaná D. Tóth *
Institute for Theoretical Computer Science
ETH Zürich, CH-8092 Zürich, Switzerland
toth@inf.ethz.ch

Abstract

We consider the problem of finding the worst-case optimal number of vertex π-guards that collectively monitor a simple polygon with n vertices. Variants of the problem differ in conditions on the allowable orientations of π-guards at vertices. We show that any simple polygon with n vertices, of which are convex, can be monitored by at most \( \lfloor (2n - k)/3 \rfloor \) vertex π-guards, which are edge-aligned. This bound is tight for the worst-case families of polygons known so far. We also show that \( \lfloor n/2 \rfloor \) general vertex π-guards are always sufficient, improving upon the weaker bound of \( \lfloor 3n/4 \rfloor \). To achieve these bounds we allow edge vertex π-guard vertices using a pseudo-triangulation of the polygon such pseudo-triangulation is monitored by vertex π-guards on its boundary.

1 Introduction

Urutuya [2] asked the following question: What is the minimum number of vertex π-guards that can collectively monitor any simple polygon \( P \) with \( n \) vertices. A vertex π-guard is given by \((v, H')\) where \( v \) is a vertex of \( P \) and \( H' \) is a closed half-plane \( H' \) such that \( v \) is on the boundary of \( H' \). There may be at most one π-guard at each vertex of \( P \). A π-guard \((v, H')\) monitors \( P \) if and only if the closed line segment \( 2v \) is in \( P \cap H' \). A vertex \( v \) of a reflex vertex cannot monitor the complete angular domain. Restrictions on the orientation of the half-plane \( H' \), where \( v \) is a reflex vertex, lead to these variants of the problem. We may consider that at every reflex vertex \( v \), the possible guard \((v, H')\) is:

- inward-facing, i.e., the two sides of \( P \) adjacent to \( v \) are disjoint from the interior of \( H' \);
- outward-facing, i.e., inward-facing and the boundary of \( H' \) is collinear with one of the edges adjacent to \( v \);
- general, i.e., we do not restrict the orientation of \( H' \).

Since adjacent pseudo-triangulation of the decomposition share two vertices and optimal guard placements in two pseudo-triangulations sharing a vertex \( v \) can generally consider considerant 

2 Preliminaries

A pseudo-triangle \( T \) is a simple polygon with exactly three convex vertices, called ears. If \( b, c \), and \( a \) are the corners of \( T \), we denote by \( T(\mathbf{a}, \mathbf{b}, \mathbf{c}) \) and \( T(\mathbf{a}, \mathbf{b}, \mathbf{c}) \) the chain of reflex vertices between \( b \) and \( c \) in clockwise direction on the boundary of \( T \). For a simple polygon \( P \), a \((\alpha, \beta)\)-pseudotriangulation is a set \( D \) of non-overlapping pseudo-triangles such that \( P = \cup D \), vertices of pseudo-triangles are vertices of \( P \), and every vertex of \( P \) is either in \( \cup D \) or reflex in a pseudo-triangle \( T \in D \). As triangles \( \alpha \) and \( \beta \) are pseudo-triangles, any triangulation of a simple polygon \( P \) is a decompression into pseudo-triangles. Note, though, that we consider exclusively decompositions (i.e., pseudo-triangulations) where each vertex is a reflex vertex of a face, either of the outer face, or of a pseudo-triangle. This property is also known as axis-orientedness [3]. Every polygon simple polygon has at least one pseudo-triangulation, but this pseudo-triangulation is not necessarily unique. Observe that a pseudo-triangle in a pseudo-triangulation is uniquely determined by its three corners; hence we use the (pseudo-triangulation) notation \( \sigma \) for a pseudo-triangle with vertices \( b, c \), and \( a \). The number of pseudo-triangles in a \((\alpha, \beta)\)-pseudo-triangulation of a simple polygon \( P \) is always the number of convex vertices of \( P \) minus two. The dual graph \( \mathcal{D}(\sigma) \) of a \((\alpha, \beta)\)-pseudo-triangulation \( \sigma \) is defined by choosing \( E(\sigma) \) as the node set of \( \mathcal{D}(\sigma) \) and connecting two nodes by an edge if the corresponding pseudo-triangles have a common edge (a diagonal of \( \sigma \)). Note that \( \mathcal{D}(\sigma) \) is always a tree if \( \sigma \) is a pseudo-triangulation of a simple polygon.

The Eighteenth European Workshop on Computational Geometry
April 10-12, 2002, Warsaw, Poland

3 Edge-aligned π-guards

Proof [Theorem 1]: Following, and generalizing the idea of Proposition 2, we describe three different guard allocations for \( \mathcal{D}(\sigma) \) that totally \( (2n - k)/3 \) edge-aligned vertex π-guards. We define these allocations with respect to an arbitrary but fixed \((\alpha, \beta)\)-pseudo-triangulation \( \sigma \). In all three guard allocations, we assign each pseudo-triangle \( T \in \sigma \) a vertex \( v \) and \( \sigma \) will be there \( v \) edge-aligned π-guards at one corner of \( T \) and along the two adjacent chains to that corner such that these guards collectively monitor \( T \). The three guard allocation schemes will jointly use all three corners (and thus different pairs of chains) for each \( T \in \sigma \).

Figure 2: A pseudo-triangulation and its dual graph for a simple polygon.

Figure 3: Guards at a corner and along two adjacent chains monitor the complete pseudo-triangle.
facing r-guards along the two directed chains adjacent to $g(T)$ (see Fig. 5). By Property 3, these guards collectively monitor $T$. Note that it is always sufficient to place at most one r-guard at each vertex since every vertex in a reflex vertex of at most one pseudo-triangle of $D_m$ and $r$-guards at reflex vertices are overallocating. (This already has an upper bound of $[2n - 2h]/3$ in the general model.)

In the last step, we route every vertex r-guard $(v, H)$ at every reflex vertex $v$ until it is self-blocking. Recall that $r$ is a reflex vertex of exactly one $T_r \in D_m$, and its guard monitors $T_r$ only if $v$ is on a chain adjacent to $g(T_r)$. But at $v$, $(v, H)$ so that the area becomes $\angle(T)$ while monitoring the angular domain of $g(T)$ in $T_r$ by the choice of $g(T)$ (see Fig. 6).

Consider the pseudo-triangle $T_1$ at the root of $G(D_m)$. We can guard $T_1$ in three different ways by choosing one of its three corners and placing guards at that corner (the guarded corner $g(T_1)$ of $T_1$) and along its two adjacent chains (see Fig. 3). Every choice of guard allocation for $T_1$ induces a direction on its chains. The sides on the chains adjacent to $g(T_1)$ are directed away from $g(T_1)$, and the sides on the remaining chain have no direction. Note that we do indeed distinguish between these directions for each side—the two standard directions and no direction. Fixing the directions for one chain of a pseudo-triangle uniquely determines the directions for the remaining ones. Furthermore, each side of the pseudo-triangle can have one direction, i.e., a pseudo-triangle induced by a direction from its parent via its joined side. Therefore, choosing a guarded vertex $g(T_1)$ in $T_1$ not only induces a unique direction on the sides of $T_1$, but also determines the direction of all sides of $D_m$ (see Fig. 4). Furthermore, since each pseudo-triangle $T$ now has exactly one corner with two outgoing edges, we define this corner to be the guarded corner $g(T)$.

Finally, note that for each corner $v$ of every $T \in D_m$, there is a choice of the guarded corner $T_v$ such that the resulting orientation of $D_m$ implies that $v$ is the guarded corner of $T_v$. After choosing $g(T_1)$ and propagating the directions along $G(D_m)$, we place a second step vertex $r$-guards along the directed edges of $D_m$. More specifically, in each $T \in D_m$, we place a vertex r-guard at the corner $g(T)$ and outward-facing $r$-guards along the two directed chains adjacent to $g(T)$ (see Fig. 5). By Property 3, these guards collectively monitor $T$. Note that it is always sufficient to place at most one r-guard at each vertex since every vertex in a reflex vertex of at most one pseudo-triangle of $D_m$ and $r$-guards at reflex vertices are overallocating. (This already has an upper bound of $[2n - 2h]/3$ in the general model.)

In the last step, we route every vertex r-guard $(v, H)$ at every reflex vertex $v$ until it is self-blocking. Recall that $r$ is a reflex vertex of exactly one $T_r \in D_m$, and its guard monitors $T_r$ only if $v$ is on a chain adjacent to $g(T_r)$. But at $v$, $(v, H)$ so that the area becomes $\angle(T)$ while monitoring the angular domain of $g(T)$ in $T_r$ by the choice of $g(T)$ (see Fig. 6).

5 Upper bound for the general model

To establish the upper bound in the general model, we rely on a special pseudo-triangle of $P$, which corresponds to the successful pulling triangulation of convex polygons. A pulling pseudo-triangle of a simple polygon is obtained by choosing (i.e., pulling) a convex vertex $v$ and connecting it to all of its convex neighbors of $v$ via pseudo-guts (see Fig. 8). The pulling pseudo-triangulation has the special property that every pseudo-triangle $T \in D_m$, with the exception of the root, has at least one corner adjacent to its parent, i.e., at least one of the two vertices which $T$ shares with its parent is with respect to $T$.

Proof (Theorem 1) [sketch]: The basic idea of the proof is to construct a partition $F$ of the vertices $V_0$ of $P$ and to place vertex $r$-guards at each at most $|F|$ vertices for each $F \in F$. If follows that the total number of vertex $r$-guards will be at most $|F|/2$.

In order to define the partition $F$, consider an ordered pulling pseudo-triangulation $D_m$. We assign a subset $\Phi_0 \subset V_0$ to each pseudo-triangle $T \in D_m$, such that $F_0 := \{T : T_r \in D_m\}$ is a partition of $\{v : v \in \Phi_0\}$. Then we place vertex $r$-guards at some vertices $\Phi \subset \Phi_0$. The pseudo-guts of $D_m$ are processed from bottom to top, that is, in $T \in D_m$ is processed when all its successors have already been processed. Processing $T \in D_m$ involves the following steps:

1. Defining $\Phi_0 \subset V_0$.
2. Specifying $\Phi \subset \Phi_0 \subset V_0$ with $|\Phi| = 2|\Phi_0|$.
3. Verifying that
   (a) the vertex $r$-guards placed on the boundary of $T$ (not only those in $\Phi_0$) can be oriented such that they collectively monitor $T$.
   (b) for each vertex $v$, the orientation of a r-guard at $v$ is consistent in every $T \in D_m$ adjacent to $v$.

The consistency of the orientation relies heavily on the monotonicity of our pseudo-triangulations: a r-guard at a vertex $v$ can monitor all convex angular domains adjacent to $v$ and at the same time it can be an outward-facing r-guard with several possible orientations in a pseudo-triangle where $v$ is a reflex vertex. This means that if we include at one step in the construction a vertex $v$ in $\Phi$ where $v$ is a corner in $T$, then we are free to choose the orientation of the r-guard at $v$ whenever we are processing a pseudo-triangle $T$ in which $v$ is a reflex vertex as long as the guards remain outward-facing in $T$.

References


Figure 9: A pulling pseudo-triangulation with respect to $z$.
The Eighteenth European Workshop on Computational Geometry
April 10-12, 2002, Warsaw, Poland

The Orthogonal Fortress Problem with Floodlights

Andreas Spiller
Institut für Informatik
Friedrich Schiller Universität Jena
07743 Jena, Germany
spiller@minet.uni-jena.de

Hans-Dietrich Hecker
Institut für Informatik
Friedrich Schiller Universität Jena
07743 Jena, Germany
hdh@mipool.uni-jena.de

ABSTRACT
We study the problem of illuminating the exterior of a polygon with finitely many light sources at the vertices of a polygon. For some classes of rectilinear polygons we give bounds for the number of light sources that are always sufficient for illumination.

1. INTRODUCTION
We suppose that the reader is familiar with the concepts of convex polygons, simple polygons and rectilinear polygons. Given a simple polygon \( P \) we denote the interior with \( \text{int}(P) \), the exterior with \( \text{ext}(P) \) and the boundary with \( \partial(P) \). In this context a polygon \( P \) is called a convex polygon if for every two points \( a \) and \( b \) in \( P \) the line segment \( ab \) is contained in \( P \). A simple polygon is called a rectilinear polygon if \( \partial(P) \) is a union of \( x \) - and \( y \)-intervals.

2. UPPER BOUNDS
We study the problem for rectilinear polygons too, and we are able to prove that for every \( \alpha \in [\frac{1}{2}, 1] \) a simple rectilinear polygon \( P \) can be illuminated by \( \frac{1}{1 - \alpha} k \) light sources at the vertices of \( P \), where \( k \) is the number of light sources that we want to use. In this context we define a polygon \( P \) to be \( \alpha \)-illuminated if \( \frac{1}{1 - \alpha} k \) light sources at the vertices of \( P \) are sufficient for illumination of \( P \).

3. CONCLUDING REMARKS

4. REFERENCES

THEOREM 1. Let \( P \) be an orthogonally convex polygon and \( \alpha \in [\frac{1}{2}, 1] \). Then we can illuminate the exterior of \( P \) with \( \frac{1}{\alpha} k \) light sources.

PROOF. We know that \( P \) is a \( \alpha \)-illuminated polygon if \( \alpha \) is the ratio of the number of light sources at the vertices of \( P \) and the number of light sources that are always sufficient for illumination of \( P \).

Figure 1: An orthogonally convex polygon.

Figure 2: A section of three Edges in region \( A_1 \).

Figure 3: One possible placement of the floodlights.

Figure 4: Another possible placement for the floodlights.

THEOREM 3. Let \( P \) be an orthogonally convex polygon with \( k \) vertices, and \( \alpha \in [\frac{1}{2}, 1] \). Then the exterior of \( P \) can be illuminated with \( \frac{1}{\alpha} k \) light sources at the vertices of \( P \) when \( \alpha \) is the ratio of the number of light sources at the vertices of \( P \) and the number of light sources that are always sufficient for illumination of \( P \).

The idea for the proof is the same as in the proof of theorem 1. But instead of only three vertices we consider \( k \) vertices of \( P \).

On the other hand we are able to prove that for every \( \alpha \in [\frac{1}{2}, 1] \) the problem of finding the minimum number of \( \alpha \)-illuminated polygons is NP-hard.

REFERENCES


Rectilinear Trees under Rotation and Related Problems

[Extended Abstract]

Benny K. Nielsen
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
Denmark
benny@diku.dk

Pawel Winter
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
Denmark
pawel@diku.dk

ABSTRACT

We consider the problem of finding a minimum spanning tree for a set of points in the plane where the orientations of edges are restricted to a uniformly distributed set of orientations, \( \lambda = 2, 3, 4, \ldots \), and where the coordinate system can be rotated around a point to any arbitrary angle. The most important case with applications in VLSI design arises when \( \lambda = 2 \); in this, so-called rectilinear case, the edges have to be parallel with the \( x \) or \( y \) axis. We suggest a straightforward algorithm to solve this problem. We also discuss how to solve the rectilinear Steiner tree problem in the rotational setting. Finally, we provide some computational results indicating the average savings for different values of \( \lambda \) and \( \lambda \) for both spanning and Steiner trees.

1. INTRODUCTION

Suppose that we are given a set \( P \) of \( n \) points in the plane. We are interested in finding minimum spanning trees (MST) or Steiner minimum trees (SMT) for \( P \) under the assumption that the edges are permitted to have a limited number of orientations. In fact, we will assume that these orientations are evenly spaced. More specifically, let \( \lambda \) be a non-negative integer, \( \lambda \geq 2 \), let \( \omega = \frac{2\pi}{\lambda} \). Permissible directions are then defined by the steepest of the angle and making angles \( \omega \), \( 2\omega \), \( 3\omega \), \( \ldots \), \( \lambda\omega \) with the positive \( x \)-axis. Finding such an MST is not a challenging problem since it can be defined as a minimum spanning tree problem in a complete graph with appropriate edge length.

Suppose that we are permitted to rotate the coordinate system. Rotation by \( \omega \) (or a multiple of \( \omega \)) will have no impact on the lengths of the edges. But for any angle \( \omega \), the edge lengths will change. This is the value of \( \omega \) minimizing the length of an MST for \( P \). Once \( \omega \) is fixed, finding an MST is straightforward. Determining similar SMTs seems to be more complicated. However, for the most important rectilinear case, we will show that the search for such SMTs can be reduced to \( O(\lambda^2 \log \lambda) \) in linear time.

2. LENGTH OF A SEGMENT

Let us first examine the situation where there are only two points \( u = (a_1, a_2) \) and \( v = (b_1, b_2) \). It is obvious that the best minimum spanning tree is obtained when the coordinate system is rotated so that the segment \( uv \) overlaps with one of the permissible orientations. So our problem is trivial. However, it is still interesting to determine how the length of \( uv \) changes as the coordinate system is rotated.

Assume that \( a_1 > b_1 \) and \( a_2 > b_2 \). When the coordinate system is rotated by \( \gamma \), \( 0 \leq \gamma \leq \omega \), then the length of the edge, denoted by \( |uv| \), changes. It increases until \( \gamma = \frac{\omega}{2} \), and then decreases until \( \gamma = \omega \) (Fig. 1). More specifically,

\[
|uv| = |uv| \cos \gamma + \sin \gamma
\]

In particular, if \( \gamma = \frac{\omega}{2} \), then

\[
|uv| = |uv| \cos \frac{\omega}{2} + \sin \frac{\omega}{2}
\]

It is obvious that the function \( f_\gamma(x) = |uv| \) is periodically strictly concave (with the period \( \omega \)). Furthermore, the minimum is attained when the direction of the segment \( uv \) overlaps with one of the direction rays.

3. MINIMUM SPANNING TREE

Consider a collection \( S \) of segments. Their total length will be minimized when the rotation of the coordinate system is such that the orientation of one of the segments to overlap with one of the direction rays. This follows immediately from the fact that segment lengths are piecewise strictly concave functions of the rotation angle. The sum of piecewise strictly concave functions is also piecewise strictly concave.

Consider any \( \lambda \) of \( n \) points in the plane. Assume that the coordinate system has been rotated in such a way that an MST \( T \) for \( P \) is shortest possible. In view of the above remark, one of the edges of \( T \) overlaps with one of the direction rays.

The algorithm to determine \( T \) is straightforward. For each pair of points \( a \) and \( b \) of \( P \), consider the segment \( ab \). Rotate the coordinate system so that the orientation of one of the direction rays overlaps with one of the direction rays.

For a fixed rotation of the coordinate system, an MST can be computed in \( O(n \log n) \) time. Since it is only necessary to consider \( O(\lambda) \) different rotations, the total running time for computing a rotational MST is \( O(\lambda n \log n) \). We will in Section 5 return to the issue of making this algorithm much more efficient in practice.

4. RECTILINEAR STEINER TREE

Consider the problem of constructing the set \( P \) of points by a tree of minimum total length, but allowing additional junctions, called Steiner points. When a set of \( \lambda \geq 2 \), uniformly spaced legal orientation is given, this is denoted the \( \lambda \)-shortest tree problem in uniform orientation metric [1, 4].

The rectilinear \( (\lambda = 2) \) and Euclidean \( (\lambda = \infty) \) Steiner tree problems have received considerable attention in the literature [2, 3].

In this section we consider the rotational variant of the rectilinear Steiner tree problem: Find a shortest set intersection of \( P \) using only two perpendicular orientations, note that the two orientations are not given, but are restricted to be perpendicular.

Assume that the coordinate system is rotated at an angle \( \theta > 0 \leq \omega \), \( \omega > \frac{\pi}{2} \). In this problem the problem can be solved in the usual rectilinear \( \lambda \) shortest tree problem, that is, to find a rectilinear \( \lambda \) shortest tree minimum tree (RMST). An RMST is a union of \( \lambda \) Steiner trees (STs) in which all terminals are leaves (and all internal nodes are Steiner points). Huang [2] proved that there exists an RMST for which every ST has a particular shape: The ST consists of a backbone, which is a shortest path between two terminals, say \( a \) and \( b \), using at most one vertical and at most one horizontal line segment (in Fig. 3 the backbone consists of the line segments \( a, b \), where \( c \) is the corner point of the backbone); all the remaining terminals are connected to the backbone via using exactly one line segment (e.g., segment \( t, g \) in Fig. 3). Furthermore, none of the remaining terminals are connected to the backbone via the corner point of the backbone.

Figure 1: \( f_\gamma(x) = |uv| \), for \( a \in [0, \omega], \lambda = 2, 3, 4, 5. \)

Figure 2: MST length variation.

Figure 3: Rotating a rectilinear ST with one corner point.

Now let us assume that the rotational problem has an optimal solution for a given \( \theta \leq \omega \leq \frac{\pi}{2} \). We denote this optimal solution — which is an RMST for the orientations given by \( \theta \) — as ST of \( P \) having the shape described above. We will now show that at least one of the STS in the RMST must have a backbone that consists of one line segment, i.e., the tree has no corner point.
Assume to the contrary that every FST has a backbone with a corner point. Let $P_k$ be an FST and let $|P_k|$ be the length of the FST on a function of $\alpha$ for $\alpha > \beta$ is sufficiently small. Consider again $P_k$, the FST in Fig. 3. The corner point will move along the indicated halfcylinder, such that the angle of the segment $u$ and $v$ remains at $\pi/2$. Thus the length of the backbone is a strictly concave function of $\alpha$ as shown in Section 2. Similarly, the length of a segment that connects one of the non-terminal vertices to the backbone will be a strictly concave function. Therefore, $|P_k|$ is a strictly concave function of $\alpha$, and the same will hold for the sum of all FST lengths. Therefore, the RMSST can in fact be shortened, which is a contradiction.

Since at least one of the FSTs must have a backbone without a corner point, the orientation of the backbone line segment will overlap with the orientation given by its two endpoints. The optimal solution to the rotational rectilinear Steiner tree problem can therefore be found using an algorithm similar to the one described in Section 3. For each pair of points $u$ and $v$ in $P$, rotate the coordinate system so that one of the direction rays overlaps with the segment $u$. Compute an RMSST for this direction and repeat this procedure for all pairs of points; the shortest RMSST computed will be an optimal solution to the rectilinear rectilinear Steiner tree problem.

For $\beta > 0$ we conjecture that a similar reduction of the necessary angles can be obtained, but it will not be polynomial in the number of terminals as for the rectilinear Steiner tree problem; it is not enough only to consider the orientations given by pairs of points in $P$.

5. COMPUTATIONAL RESULTS

In this section, we give some computational results indicating the effect of allowing rotations of the coordinate system when computing MSTs and rectilinear MSTs. We used two sets of 2D problem instances: VLSI design instances and randomly generated instances (uniformly distributed in a square). The VLSI instances were made available by courtesy of IBM and Research Institute for Discrete Mathematics, University of Bonn. In this study we have focused on one particular chip from 1986.

5.1 Minimum Spanning Tree

Generally most of the edges for a given point set will not be part of an MST for any rotation angle. It is a waste of time to ‘straighten’ those edges and compute their MST. Luckily most of these edges can be pruned by using so-called bottleneck Steiner distance. Consider the complete graph $K_n$, where the vertices represent the points in the plane. Define the weight of the edge between vertices $u$ and $v$ to be $d_{uv}$; in other words, this weight is equal to the maximum distance between $u$ and $v$ when the coordinate system is rotated. Compute bottleneck Steiner distances in $K_n$. It is defined as the minimum of the longest edge encountered in all permutations of $u$ and $v$ in $K_n$, and can be computed in time $O(n^2)$ for all pairs of points see [3] for details. Let $d_{u,v}$ denote the bottleneck Steiner distance between $u$ and $v$. However, it should be noted that it is possible to construct point sets of any size where the number of edges after pruning is $O(n^2)$.

Table 2 shows the MST improvement for various values of $\alpha$ and $\lambda$ where each entry is an average over 100 runs. The table shows that the improvement increases as $\lambda$ increases, but for larger values of $\lambda$ the improvement is not as significant.

5.2 Rectilinear Steiner Tree

We used GeoSteiner [5] to compute MSTs for each of the $O(n^2)$ orientations given by the pairs of terminals (where $n$ is the number of terminals). The MST improvement obtained by rotating the coordinate system can be seen in Table 3. The values are percentages which express the improvement compared to not rotating at all. Results for both random and VLSI instances are given. For small problem instances the improvements are highest for randomly generated instances, while for large instances, the VLSI problems reached a higher improvement.

6. RELATED AND OPEN PROBLEMS

There are several other geometric combinatorial optimization problems which require a selection of a subset of edges and can in the rotational setting be approached in the same way as the MST problem. The travelling salesman problem and matching is probably the most well-known problem where a straightforward generalization occurs when the orientations are fixed but not necessarily equally spaced. Determination of the rotated MSTs in higher dimensions seems to require a straightforward generalization of the 2-dimensional case.

There are several research directions which deserve more attention in the future. First of all, it remains open if the rotated MSTs can be determined more efficiently by some direct methods that do not involve enumeration of a (limited) number of MSTs. Also, the problem of determining rotated MSTs for other than the rectilinear case is only open.

7. CONCLUDING REMARKS

We addressed the problem of determining MSTs and rectilinear MSTs when edge directions are limited to uniformly distributed orientations and where the coordinate system is permitted to rotate by any angle. We suggested a simple polynomial algorithm to solve the MST problem. We also provided some computational results indicating how big the savings can be. As it could be expected, the savings become negligible when $\alpha$ and $\lambda$ grow.

Acknowledgments

The authors would like to thank IBM and theeth Institute for Discrete Mathematics, University of Bonn, for providing invaluable computing facilities. The research was partially supported by the Danish Natural Science Research Council (grant number 66610).
Alternating Paths through Disjoint Line Segments

Michael Hoffmann and Csaba D. Tóth
Institute for Theoretical Computer Science
ETH Zürich, CH-8092 Zürich, Switzerland
{hoffmann, toth}@inf.ethz.ch

ABSTRACT

It is shown that every line segment visibility graph on disjoint line segments in the plane admits an alternating path of length $O(\log n)$, answering a question of Bose [2, 3]. Moreover, we give bounds on the constants hidden by the asymptotic notation.

1. INTRODUCTION

Consider a set $S$ of $n$ disjoint obstacles, represented by line segments, in the Euclidean plane. A mobile agent wishes to visit a maximal number of vertices (i.e., segment endpoints) under various constraints. We are specifically studying paths where the agent moves along a straight line segment between two vertices, and it cannot cross a segment (although it may walk along a segment from one of its endpoints to the other). In contrast to the well-studied problem, the question is not the minimal distance traveled, but rather whether there exists a path through all vertices, or what is the maximal number of vertices that can be visited.

It has been recently shown [4] that there exists a simple Hamiltonian circuit for a set $S$ of disjoint line segments, if no two segments are collinear. Unbehauch and Waldsath [5] gave a construction where $S$ does not always have a simple Hamiltonian circuit, circumventing the Erdős–Stone–Sarkőzy–Szemerédi theorem [6]. However, it remains open whether there exist paths on disjoint line segments for a set $S$ that are not just the alternating Hamiltonian circuits.

Theorem 1. Let $S$ be a set of disjoint line segments in the plane. Then for any $x \in S$, there is an alternating path of length $2 \log_2(n+1)$ passing through $x$.

Apart from a constant factor, this bound is best possible: there are sets of disjoint line segments where the longest alternating path has length $\frac{n^2}{2} \log_2 n$ [7].

We defer Theorem 1 to an easy consequence of the following.

Theorem 2. For any $S$ of disjoint line segments in the plane, and for any $x_1, x_2 \in S$, there exists an alternating path from $x_1$ to $x_2$.

2. PRELIMINARIES
3. THE LOWER BOUND

The key fact in our argument is stated in Lemma 1 below. The proof of Theorems 1 and 2 then follows by elementary graph covering arguments.

Denote by \( D(H_t) \) the Hamiltonian cycle \( H_t \) together with all its segment diagonals. (Observe that \( D(H_t) \) is planar and its maximal degree is three.

**Lemma 1** For any directed edge \( e^+ \) and any edge \( f \) of \( D(H_t) \), there is a directed alternating path from \( e^+ \) to \( f \) if and only if \( D(H_t) \) is planar and its maximal degree is three.

We can, basically, repeat the standard existence proof of paths of length \( \log n + (\log n)^2 \) in a directed acyclic graph with no feedback if the maximal degree is in \( R \).

**Proof.** For a directed alternating path \( P \) starting with a segment edge \( e \) and ending with another segment edge, let \( f(P, e) \) denote the number of segment edges \( f \) such that there is a directed alternating path from \( e \) to \( f \) which forms the last segment edge of \( P \).

The path \( P \) can be extended by a visibility edge of \( G \) in less than \( R \) different ways, and in each case a unique segment edge follows. The possible resulting paths are not necessarily alternating, since some vertices of \( P \) could be reached again. By all means, summing for all possible alternating path extensions \( R \) of \( P \) we get

\[
\sum f(P, e) \geq f(P, e) - 1.
\]

The \( R \)-term appears since we cannot reach the last edge of \( P \) by any segment. The direction of \( e \) can be chosen such that \( f(P, e) \geq 1 \) since any segment can be assumed from \( e \) on an alternating path.

Each time we extend the path \( P \) by a new edge, \( f(P, e) \) decreases by at most a factor of \( R \). The lower bound \( \log(n^2 - (n^2 - 1)) \) on the number of segment edges on this path follows from the degree of \( R \) on a balanced \( [R] \)-ary tree.

4. UPPER BOUND

Complementing the results from the previous section, we will now show an asymptotically matching lower bound, that is constructed using \( S \), \( B \), and \( R \) of line segments that do not have long alternating paths.

**Theorem 2** There are \( n \) disjoint line segments in the plane that do not adopt an alternating path of length greater than \( \frac{n}{\log n} \log n - 2 \cdot \frac{n}{\log n} \log n - 1 \).

**Proof.** Let \( \lambda \) be the length of a longest alternating simple path in \( S \). The set \( S \) is constructed recursively as follows. All line segments are chords of \( R \) circles of three segments arranged in a triangular fashion, i.e., such that \( S \leq R \). One of these vertices on the chords partitions \( e \) into cases. \( S \) is obtained from \( S \) by inserting a sequence of \( n \) segments. Figure 3 shows \( \alpha \) and \( \beta \).

While we will use the results on the segment graphic in the plane, the result on the plane will be used as follows. Let \( \lambda \) be the length of a longest alternating simple path in \( S \). The set \( S \) is constructed recursively as follows. All line segments are chords of a circle. One can construct \( R \), i.e., the plane, by inserting a sequence of \( n \) segments (i.e., a copy of \( S \)) on every arc of \( e \) that is bounded by only one segment from \( S \). Figure 4 shows \( \alpha \) and \( \beta \).
REFERENCES


2. Böhm, P., Hohle, M., and Toussaint, G.
Every set of disjoint line segments admits a binary tree.

3. Désormeures, E. D., and O'Rourke, J.
Open problems from CCCG'99.

4. Hoppe, M., and Tödt, C.
D. Segments of chordal visibility graphs are Hamiltonian.

5. Rappaport, D.
Computing simple circuits from a set of line segments is NP-complete.

Computing simple circuits from a set of line segments.

7. Uras, M., and Watanabe, M.
On a counterexample to a conjecture of minimality.

6. REFERENCES

A simple kinetic visibility polygon

ABSTRACT

Given a set of moving obstacles in the plane, we propose a method for maintaining efficiently the visibility polygon of a (possibly non-convex) polygon and a set of line-segments with a common endpoint.

Keywords

Categories and Subject Descriptors

1. INTRODUCTION

Visibility computations are central in many computer graphics algorithms and in robot motion planning. A very useful tool in the determination of the objects visible from a viewpoint. Thus, in the plane, the visibility polygon is an important visibility structure. It is a set shaped polygon centered at the viewpoint V, whose edges are the visible parts of the object in the scene, and whose interior intersects no object.

Efficient algorithms exist to compute the visibility polygon in the static case, but applications of this problem may apply to moving objects. Computing the visibility polygon at various times on a static "snapshot" of the scene is inefficient, since we do not take into account the temporal coherence that arises from the constancy of the movements of the objects (and possibly the viewpoint); if the time step is too small, we will compute many times the very same (non-moving) visibility polygon.

We use the kinetic data structure framework introduced by Guibas [2, 1] to propose a simple algorithm that maintains the visibility polygon of a view point in a 2-dimensional scene when all objects may move. The structure maintained is in fact a real radial decomposition of the scene. Section 2 treats the case of smooth convex obstacles. Section 3 examines the case of simple (non-convex) polygons.

Kinetic data structures (KDS) are a way to efficiently and accurately maintain a data structure built on top of continuously moving items (e.g., a convex hull). In order to maintain an attribute A over a set of moving items (items as generally points), each test in the pro of correctness of the construction of A is analyzed to detect the time at which it will fail. The idea is that maintaining the visibility of all those tests (called "certificate") guarantees us that the attribute A is maintained also, since the certificates provide a proof of correctness. Certificates are ordered in a priority queue, according to their failure time. When the smallest time passed above the first certificate's failure time, the attribute is modified, and the pro is updated (i.e., some certificates disappear, others are created, and their failure time is computed). This method only requires that the motion of the items be known in the short term. For short, as said could say that kinetic data structures get rid of step-by-step simulations, and implement in fact time sweep algorithms.

We now get interested in maintaining the visibility polygon of a scene. Here, in the KDS terminology, the items are the convex smooth objects, or the polygons' vertices. We maintain a real radial decomposition of the scene, thus, the certificates we use take care of the well ordering (i.e., cyclically sorted) of the segments in the decomposition. Finally, the radial decomposition of the scene allows us to quickly build the visibility polygon.

2. CONVEX OBSTACLES

Let O be a set of convex obstacles in the plane. Let F be the free space; the complement of the union of the obstacles in the plane. Let V be a point in F. We aim at maintaining the visibility polygon of V when V and elements of O move in the plane. We assume we can compute in constant time the visibility tangents of an obstacle, that are defined as the two tangents to the object passing through the viewpoint V. Let T = (t1, t2, ..., t|O|) the tuple of the visibility tangents, sort in the counter-clockwise order.

Let S be a direction. We denote by V(S) the obstacle seen by V in the direction S. V(S) can possibly be the "blue sky" that we denote as V(S) = \infty. An important observation is that V(S) is constant between two consecutive visibility tangents. Thus, a way to define the visibility polygon around V, is to...
2.1 Kinetic visibility polygon

The visibility polygon (VP) changes only when two visibility tangents (VT) cross each other (but two VTs may cross each other without affecting the VP). Thus, we can maintain the VP by detecting when two consecutive VTs will cross, then updating the VP accordingly to the kind of both VT, and swapping the two VTs involved to keep them sorted in counter-clockwise order.

However, having computed the VP at a given time is not sufficient to maintain it efficiently when obstacles move. We need some additional data that will have to be maintained also: for each VT time t_i, we maintain it as hit-item, which is the obstacle that is hit by the VT beyond the tangency point (it can be∞). In fact, we maintain a weak radial decomposition of the scene, where only the farthest object hit by a VT is registered and not the near object.

For each crossing, the update of the visibility polygon is done in constant time, by distinguishing 8 cases. First, we need to characterize the VTs. Half of them will be left if seen from V, they pass to the left of the obstacle. The other half will be right visibility tangent a.

We explain the naming of the crossing events with an example. Figure 1 shows an LL and an LR crossing events (from left to right, the figure presents the obstacles involved in the event, just before, during and after the crossing). LR means that the first VT (in counter-clockwise order) is a left tangent, the consecutive VT is a right tangent, and the last one is far. When the crossing occurs, the tangency point of the left tangent is furthest from V than the tangency point of the right tangent. Hence, the 8 cases are named LL, LR, LLR, LR, RL, RR, LLR, RR.

![Figure 1: (a) example of an LL event (up) and an LR event (down); (b) all events](image.png)

Now we need to update the VP and the hit-item when the LR crossing occurs, see Figure 2. The VTs that cross each other are consecutive in our 3D order of ordered VTs. Let t be the time at which the crossing occurs. Then at times t" and t', no other VT can lie between the two VTs we are interested in. Therefore we can be sure that any other obstacle (different from G or D in Figure 2) either completely crosses the angular section E, or has no intersection with it. This ensures the correctness of the update process.

First we check if object C resides between the two points of tangency at time t. To do so, we just need to check whether g-hit-item (the bit-item of the right tangent of obstacle G) is the same as hit-item or not. Then, if C does not exist, else, C exists. Note that [g-hit-item] can be ∞. This information is enough to update the hit-item.

![Figure 2: Update of an LR event, the arrows point to the hit-item belonging to the consecutive visible items](image.png)

We then process the seven cases are processed in the very same way, by just changing the rules in Figure 3.

![Figure 3: Processing an LR event](image.png)

2.2 Complexity

We express the complexity of this kinetic data structure using terms proposed by Chazelle and Guibas [1]. Our data structure is optimal in the sense that it is linear in the size of the scene (the set of all obstacles). It is running, meaning that the cost of processing a certificate failure is small constant-time in our case. This JDKS is also using meaning that the number of certificates that involves a single object is O(1) in our case, with m certificates per obstacle.

However, it is not optimal since we may have to update many obstacles after the event. The next obvious step is to make our visibility polygon dynamic. Imagine a set s of small disks virtually aligned above a big disc, and the view point traversing the plane horizontally under the big disc. Using Chazelle and Guibas terminology, our JDKS is not optimal, since the total number of events processed may be of a higher order than the number of collisions in the VP. An optimal algorithm would update as many certificates as there are changes in the visibility polygon during the animation. Hall-Help and Rabinowicz [3] propose such an algorithm, but are limited to convex smooth obstacles. They do process only one certificate failure for each change in the VP, but the cost of processing one event is not constant in time. However, the overall cost of processing all events for a simple motion of the observer only is significantly better in their algorithm.

3. SIMPLE POLYGONAL OBSTACLES

We extend the above adaptation of the method to simple polygonal obstacles. A polygon can be concave, but none of its edges cross another. We consider that obstacles in the initial set are in general position, meaning that no pair of vertices is aligned with the observer V.

The basic idea is the same. For each vertex x, we keep track of the ray starting at V and passing through x, which makes, by a small abuse of language, our new visibility tangent. For each vertex v, we keep track of its hit-item and of the type of the vertex. Here, the type of a VT is a bit more complex: Figure 4 shows how we name the type of a vertex depending on the position of both its adjacent edges relative to the VT passing through it.

![Figure 4: Various types of vertices](image.png)

Note that the hit-item of a vertex can be irrelevant (and even wrong) if the part of its VT beyond passes through the interior of the adjacent polygon of x. It is yet correctly updated when the VT gets swee in free space.

The certificates that must stay true are the same as in the case of convex smooth obstacles we described a cross-

In the event when two consecutive vertices (which are kept sorted in counter-clockwise order around the view point) get aligned with the VP, the situation is that no other VT can lie between the two VTs. This is the case here, since two consecutive vertices (convex in the cyclic order and on a polygon boundary) can get aligned with V.

Thus, we have two kinds of update when a certificate fails. If the certificate concerns vertices concurrent on the border of a polygon, then we have to update their type, and possibly their hit-item and the VP, this again, is done in constant time. In such a case, the update is similar to those in Section 2, with some more cases because we have to take into account how the vertices, namely Up and Down.

Figure 5 shows how the type of a vertex changes when V crosses the supporting line of one of its adjacent edges. Edges of a polygon are oriented so that the inside of the polygon lies to the left of its edges. When the crossing occurs, one vertex is nearer to V than the other: it will be said near, and the other, far; one vertex is following the other on the polygon's boundary: it will be said next, and the other, prev.

This is the terminology used in Figure 5 to decide which translation we should target.

![Figure 5: Updating a vertex's type. The arrows represent transitions where hit-item must be updated](image.png)

The complexity of the structure for polygonal objects is the same as for convex smooth objects.

Hall-Help proposes a refinement of this method, which processes exactly as many events as there are changes in the VP. The cost of processing one event is large, but for the same motion of all obstacles (and the viewpoint), his algorithm is less costly in time because a reduction on the constraints on the shape of the scene decomposition. However, he designed his algorithm only for convex obstacles.

The algorithm proposed by Hall-Help can in fact be adapted to simple polygonal obstacles using a radial decomposition...
of the polygonal scene where each edge of a polygon is considered separated of the other, so that the radial segments would lie in the polygon as well, and not only in free space.

4. CONCLUSION AND FUTURE WORK

We have presented a simple kinetic data structure that maintains the visibility polygon of a moving point in a planar scene of moving the objects (convex or smooth or simply polygonal).

A change in the visibility polygon is processed in constant time. The size of the structure is optimal (linear in the size of the scene). However, it processes as many events among all the events that processed, hence can have no effect on the VP. However, the number of events processed remains optimal if the scene is sparse, because nearly all obstacles become visible.

These algorithms can perhaps be accelerated by representing polygons with various level of detail (perhaps even aggregating polygons that are close to each other and far away from the view plane), and using a sufficient criterion to increment or decrement the LOD for some groups of polygons.

We may also want to describe a 3-dimensional visibility polyhedron, and extend it to the KDS framework. This looks much more difficult than in the 2D case.

5. REFERENCES


A kinetic view of the shooter problem

The problem of a kinetic data structure that maintains the visibility polygon of a moving point in a planar scene of moving objects (convex or smooth or simply polygonal). A change in the visibility polygon is processed in constant time. The size of the structure is optimal (linear in the size of the scene). However, it processes as many events among all the events that processed, hence can have no effect on the VP. However, the number of events processed remains optimal if the scene is sparse, because nearly all obstacles become visible.

These algorithms can perhaps be accelerated by representing polygons with various level of detail (perhaps even aggregating polygons that are close to each other and far away from the view plane), and using a sufficient criterion to increment or decrement the LOD for some groups of polygons.

We may also want to describe a 3-dimensional visibility polyhedron, and extend it to the KDS framework. This looks much more difficult than in the 2D case.

5. REFERENCES


A kinetic view of the shooter problem

The problem of a kinetic data structure that maintains the visibility polygon of a moving point in a planar scene of moving objects (convex or smooth or simply polygonal). A change in the visibility polygon is processed in constant time. The size of the structure is optimal (linear in the size of the scene). However, it processes as many events among all the events that processed, hence can have no effect on the VP. However, the number of events processed remains optimal if the scene is sparse, because nearly all obstacles become visible.

These algorithms can perhaps be accelerated by representing polygons with various level of detail (perhaps even aggregating polygons that are close to each other and far away from the view plane), and using a sufficient criterion to increment or decrement the LOD for some groups of polygons.

We may also want to describe a 3-dimensional visibility polyhedron, and extend it to the KDS framework. This looks much more difficult than in the 2D case.

5. REFERENCES

Denote by $LQ(i)$ the (maximal) linear clique consisting of $i$ and all areas that contain $(i)$.

Furthermore, let $i$ be an arc in $F$ whose node is in a cycle in the core of $F$. Let $\alpha(F)$ denote the size of a minimum independent set in the circumscribed graph for $F$ and $GD(i) = \{i, (\alpha(i), r_i)\}$. Furthermore, the greedy clique cover $GD(F)$ is defined to be $[LQ(1), \ldots, LQ(|\alpha|)]$ if $\alpha(F) \leq i$ and $[LQ(1), \ldots, LQ(|\alpha|+1)]$ otherwise.

Hus and Toul [HT31] prove the following theorem that we present here with our notations.

**Theorem 2.1** [HT31]. Assume that the set of arcs $F$ on a circle is a linear clique. Let $i$ be any arc in any cycle in the c-core for $F$. Then $GD(i)$ is a MCC for $F$. Moreover, it can be found in an $O(n)$ time assuming that the circular arc graph with ordered areas is given.

The significance of the above theorem is in providing a connection between a cycle in the c-core and the size of the minimal (linear) clique cover. Translated into simple terms, in order to find the MCC, first we need to identify a sequence of areas whose chain of the next relation forms a cycle, that is, a sequence of the arcs that wraps around the circle once or more times. Then, in such a sequence we want to find the longest segment of independent areas, that is, a stop before the arc starts wrapping around. The size depends on the size of the last element in the sequence. This is essentially the algorithm of the theorem.

Note that $F$ is a non-linear clique in a very special case.

**Corollary 2.1.** The relation to the shooting problem for a given position $p$ of the shooter is equivalent to finding the MCC in the $A(S)$. 

3. SHOOTING ALGORITHM

In this section we shall show the main ingredients of the algorithm having details to the full version of the paper.

Let $A(S)$ be the arrangement induced by lines passing through the endpoints of the segments in $S$. The size of the MCC are the same for all points belonging to the same cell as the corresponding linear arc-graphs are isomorphic.

Hence, the general shooting problem is equivalent to finding the smallest MCC over the entire arrangement. Direct application of Corollary 2.1, as it was done in the previous papers on shooting, leads to an $O(e^2)$ solution. We present an efficient representation for rectangles that allows us to maintain the size of the MCC with a poly-logarithmic cost per event when moving from one cell to an adjacent cell in a continuous motion. When the point moves between adjacent cells the size of the MCC is known in the next function that leads in turn to two kinds of changes in the underlying c-core a subtree is matched from one node to another, and the nodes in the cycle (as well as the cycle size) change, see Figure 2. Modifying these changes and updating c-cores require efficient data structures. Changes in c-cores shall be maintained in an interval tree structure, and changes in the c-core will be maintained in a data structure developed for this paper and called a p-tree.

A p-tree is built on p-core and consists of two structures that are linked together:

1. Each node in the p-core is replaced with a B-tree of its children. This will allow us to change the logarithmic time pruning between children and their parent while preserving the path decomposition (see below).
2. A path decomposition of paths in the p-core. Each path is organized as a B-tree and links nodes into longest paths. It is constructed as follows: while moving upwards from the leaves to the root of the p-core, at each node we combine the longest path of its children and terminate all other paths reaching this node. This decomposition allows us to quickly find the length of the cycle in the c-core.

A p-tree is illustrated in Figure 4. The size of the p-core is $O(e)$. Although we are not using randomization it is useful to think that the paths in the p-core are organized similarly to skip lists [P90].

Our solution is based on the following facts that we will list below.

1. To find $\alpha(S)$ it is sufficient to visit the arrangement $A(S)$ induced by the lines passing through the endpoints of $S$ and maintain the current value of the MCC. The size of the arrangement is $O(e^2)$.
2. The size of the MCC can be found based on the c-core by analyzing the cycle in the c-core.
3. There is an efficient representation for c-cores, called a p-core that supports the following three operations (it helps to think about p-core as a rooted tree with the root pointing to some internal node):

The semantics of JumpToRunt($r$) is to find the distance in the c-core from $r$ to the root.

**Lemma 3.1.** The time-complexity of operations FindNode and the cost of Jump and JumpToRunt is $O(\log n)$.

The input lens will be used to modify the c-core after changes to the new value of some areas, to length or the length of the cycle, and to modify a node to be another. One of useful and quite tricky applications of Jump is to find a given node $x$ is located on the cycle in a given c-core. To this end, we store the length of the cycle, computed at the beginning with Hase-Touki linear time algorithm and then recomputed with the JumpToRunt operation. With JumpToRunt($x$) we can compute the distance from $x$ to the root. Then JumpToRunt($x$) will start and end in $x$ if it is on the cycle.

4. There is a dynamic data structure that maintains the areas on a circle and allows us to find the value of the new area for each area in $O(e)$.

Our structure, called Pox, is similar to the interval tree, and $\mathbf{Pox}^{(\mathbf{W})}$ for a set of segments and area can be deleted and inserted in $\mathbf{Pox}$ in $O(e)$ time. Pox will be used to find changes in the c-core. Each time when two endpoints on a segment swap positions a request of the shooter motion.

5. The value of the MCC changes by at most 1 between the cells in $A(S)$. Indeed, let $\{c_1, \ldots, c_n\}$ denote the size of the c-core.

**Lemma 3.2.** $H(c_1, c_2)$ are the adjacent cells of $\mathbf{A(S)}$ that do not share the boundary that is one of the input segments.

Then $\mathbf{A(S)} \cap \mathbf{A(c_1)} \cap \mathbf{A(c_2)} \cap \mathbf{A(c_3)} \subseteq \mathbf{A(c_1)}$. 

**Proof:** Moving from $c_1$ to $c_2$ corresponds to swapping...
two neighbor endpoints of some two arcs in $Arc(c_1)$. In terms of the circular-arc graph for $Arc(c_1)$ it corresponds to adding or removing one edge. This may change the number of cliques in the current minimal cover; all or removing an edge one of the cliques may break into two or, after adding an edge two cliques may become a larger clique in $Arc(c_1)$. If the number of cliques increases, we have a cover that shows that $moc(c_1) \leq moc(c_2) \leq moc(c_1) + 1$. If the number of cliques decreases then removing the newly added edge reduces the problem to the previous case and we have $moc(c_1) \leq moc(c_2) \leq moc(c_1) + 1$, which ends the proof.

The lemma is actually more important than it appears at first glance. As we remember, the size of the MCC is the longest independent set in a cycle in the case. Thus the lemma allows to avoid traversing the entire cycle to find where the sequence of next starts wrapping around the circle. Instead, it can be accomplished by jumping from arc $i$ to the node $j$ in the cycle that is at the distance of the old value of two minus $i$ and then checking the overlap between arc $i$ and the next of just three arcs, $j$, next$(j)$ and next(next$(j)$). The gain is in reducing the cost to $O(n^3)$.

To solve the generalized shooter problem we start with computing the arrangement $AS$ and then for an arbitrarily selected cell we initialize the pfb-tree and compute the values of the MCC using the linear time algorithm of Han and Tsai. Then, using a schedule (such as a spanning tree of the cells) we visit the cells in the arrangement. When the shooter crosses the boundary between two cells then two endpoints of some two arcs in the circular arc-graph change their position. Using $Pos$ structure, in $O(n^3)$ time we find the new values of next for the affected area. The changes are propagated in $O(n^3)$ time to the pfb-tree and the new value of the node is recorded. We can keep all the values in a priority queue represented with a binary heap.

After visiting all the cells we can read off the solution to the shooter problem from the heap.

The correctness of the algorithm follows from the following lemma.

**Lemma 3.2.** The algorithm correctly updates the pfb-tree representing the tree after changes to the next values.

**Proof:** The pfb is initialized correctly to represent the initial cover. By case of possible changes in positions between a pair of arc when the boundary between cells is crossed, and from the correctness of operations on pfb-tree we demonstrate the claim.

It leads to the following

**Theorem 3.1.** Algorithm Shooting is correct, i.e., it finds the minimum number of cliques necessary to shoot the $n$ given segments. The cost of the algorithm is $O(n^3)$.

**Proof:** From Lemma 3.2 we know that the pfb-tree correctly represents the next relation for each cell in the arrangement as we move from one cell to another and that the returned value is equal to the size of the MCC for the current circular arc-graph. This implies that the algorithm finds the solution to the shooting problem. The shooting number is found by analyzing the neighborhood of the previous shooting number in the cycle. By Lemma 3.2 only such a neighborhood needs to be analyzed.

The initialisation of the arrangement, the initialisation of the data structures and $O(n^3)$ updates to the pfb-tree can be accomplished in $O(n^3 \log n)$ time. Moreover, after all the data structures are initialised, the cost per event is $O(n \log n)$.

4. CONCLUSIONS

We have focused in this paper on a solution to the generalised shooter problem. By designing an efficient data representation for pfb-tree, we have been able to maintain the Minimum Next Cover for the circular arc-graph corresponding to the segments viewed from the perspective of a moving shooter. It results in an algorithm with the best deterministic performance up to date. The algorithm is close to the optimal for segment arrangements with a reach set of optimal positions for the shooter.

Although the presentation assumed that the segments in $S$ are pairwise disjoint, this can be relaxed, and in particular, we can allow the shooter to cross the input segments while moving between cells. Note that when a segment is crossed the corresponding arc in the circular arc-graph jumps to the other half-circle, which makes the problem slightly more difficult.

The same approach can be used for other graph-theoretic structures such as the Maximum Dominating Sets and the Minimum Dominating Sets for circular arc-graphs.

Other computational applications of our approach include the MCC problem for a dynamic set of union length arcs and the shooting problem for segments that are semi-transparent; selected segments in the set are invisible from one side and visible when viewed from the other.

Additionally, the pfb-tree data structure can be applied whenever there is a need to react to groups of changes between nodes and to quickly jump from a node to its predecessor that is a given number of generations above in the tree. This structure is found from pfb-tree and skip lists, as well as uses a technique that we call a path decomposition.

Possible further applications may involve placing sets and transversals for more complex objects: polygons are good candidates here.

5. REFERENCES


Balanced Partition of Minimum Spanning Trees∗

[Extended Abstract]

Mattias Andersson and Christos Levcopoulos
Department of Computer Science, Lund University
Box 118, 22100 Lund
christos@cs.lth.se

Joachim Gudmundsson†
Department of Computer Science, Utrecht University
PO Box 80.089, 3508 TB Utrecht, the Netherlands
joachim@cs.uu.nl

Giri Narasimhan
Florida International University
Box 299, Miami, Florida 33199, USA
giri@fiu.edu

Abstract

To test or handle situations where additional resources are available to carry out a task, many problems from the manufacturing industry involve "optimally" dividing a task into a constant k number of smaller tasks. We consider the problem of partitioning a given set of n points in the plane into k subsets, S1, . . . , Sk, such that max_{i ∈ [k]} |M[S_i]| is minimized. A variant of this problem arises in the shipbuilding industry [3]. Initially, we show that this problem, the k-BMPST problem, is NP-hard and we then continue by presenting two approximation algorithms. The first one, a straight-forward greedy algorithm, is k-approximation algorithm that runs in O(n log n k) time. The second algorithm, which runs in O(n log n) time, is a (3/2 + ε)-approximation algorithm for the case k = 2 and a (2 + ε)-approximation algorithm for the case k ≥ 3.

Categories and Subject Descriptors
G.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—combinatorial optimization; G.2.1 [Discrete Mathematics]: Combinatorial Optimization; G.2.2 [Discrete Mathematics]: Graph Theory—graph algorithms

General Terms
Algorithms, Theory

Keywords
ε-approximation algorithms, greedy algorithms, minimum spanning trees, partitioning algorithms

1. INTRODUCTION

In one interesting application from the shipbuilding industry, the task is to use a robot to cut out a set of preprocessed regions from a sheet of metal while minimizing the completion time. In another application, a salesman needs to visit some potential buyers. Each buyer specifies a region (see, a neighborhood) within which the most needed to be held. A natural optimization problem is to find a subset of such regions that visit all of the buyer's neighborhoods and finally returns to his initial departure point. Both these problems are related to the problem known in the literature as the Traveling Salesman Problem with Neighborhood (TSPN) and which has been extensively studied [2, 4, 6, 7, 8, 9]. The problem (TSPN) asks for the shortest tour that visits each of the points. This problem was shown to be APX-hard [10].

Creating generalizations of the TSPN problem arise when additional resources (k ≥ 1 robots in the cutting problem, or k ≥ 1 salesmen in the second application above) are available. The k-TSPN problem is a generalization of the problem where we are given k salesmen and the aim is to minimize the completion time, i.e., the distance traveled by the salesmen making the longest journey.

The need for partitioning the input set such that the optimal substructures are balanced gives rise to many interesting theoretical problems. If we restrict our inputs to sets of points instead of regions an interesting problem in the area called the k-TSP problem, which is equivalent to the k-TSPN problem but with the regions replaced by points. In this paper, however, we consider the problem of partitioning the input such that the sizes of the minimum spanning trees of the subsets are balanced. More formally, the Balanced Partitioning Minimum Spanning Tree (k-BMPST) is stated as follows:

**Problem 1.** Given a set of n points S in the plane, partition S into k subsets S1, . . . , Sk such that the weight of the

largest minimum spanning tree, W = \max_{i \in [k]} |M[S_i]|, is minimized. Here M[S] is the minimum spanning tree of the set S, and |M[S]| is the weight of the minimum spanning tree of S. Note that a k-approximation for the k-BMPST problem immediately yields a 2-approximation for the k-TSP problem, by traversing the produced MSTs.

The paper is organized as follows. First, we show that the problem is NP-hard. We then present an approximation algorithm with approximation factor 4/3 + ε for the case k = 2, and with an approximation factor 2 + ε for the case k ≥ 3. The algorithms run in time O(εk n log n).

2. NP-HARDNESS

In this section we show that the k-BMPST problem is NP-hard. In order to do this we need to state the recognition version of the k-BMPST problem:

**Problem 2.** Given a set of n points S in the plane, and a real number L, does there exist a partition of S into k subsets S1, . . . , Sk such that the weight of the largest minimum spanning tree

W = \max_{i \in [k]} |M[S_i]|, is ≤ L.

In a computational model in which we can handle square roots in polynomial time, such as the real-RAM model (which will be used for simplicity), this version of the problem is equivalent to the k-BMPST problem.

Note, however, that it may be impractical in more realistic models, such as the Turing machine model, where efficient handling of square roots may not be possible. The computation of roots is necessary to determine the lengths of edges between points, which, in turn, is needed in order to calculate the weight of a minimum spanning tree. So in a realistic computational model the hardest part may not be to partition the points optimally, but instead to calculate precisely the length of the MSTs. Thus, in these more realistic computational models we would like to restrict the problem to instances where the lengths of MSTs are easy to compute. For example, this can be done by modifying the instances created in the reduction below, adding some points so that the MSTs considered only contain vertical and horizontal edges.

The proof is done (considering the real-RAM model) by a reduction from the following recognition version of Partition:

**Problem 3.** (Gwwer problems) \(a_0 = \{a_1, \ldots, a_n\}\), the recognition version of the problem is: Does there exist a subset P \(\subseteq \{1, 2, \ldots, n\}\) such that

\[|P| \leq \frac{a_0}{a_1 + a_2 + \cdots + a_n}\]

We will denote \(\#P\) by \(b\), \(b = a_0/2\). This version of Partition is NP-hard [11].

**Lemma 1.** The k-BMPST problem is NP-hard.

**Proof.** The reduction is done as follows. Given a Partition instance we create a 2-BMPST instance, in polynomial time, such that it is a yes-instance if, and only if, the Partition-instance is a yes-instance. Obviously Partition then polynomially reduces to 2-BMPST. Given that the Partition-instance contains \(n\) integers \(a_0, a_1, \ldots, a_n\), we create the following 2-BMPST instance. A set of points \(S\) as shown in Figure 1(a) is created, with inter point distances as shown in Figure 1(b). A closer description of these points and some additional definitions is given below:

- \(a_i = (i, 0), i = 0, \ldots, n\), where \(a_0 = (0, 0)\).
- \(l = \{(i, k), i = 0, \ldots, n\}\), where \(L = (n + 1, 0)\),
- \(r_i = \{(i, k), i = 0, \ldots, n\}\), where \(r_i = (i + 1, 0)\),
- \(b = \{(i, k), i = 0, \ldots, n\}\), where \(b\) is the midpoint on the line between \(l_i\) and \(l_{i+1}\), and \(r_i\) as well.
- \(P = \{(i, k), i = 0, \ldots, n\}\), where \(P\) is the midpoint on the line between \(r_i\) and \(r_{i+1}\).

For any given partition \(P = \{(i, k), 1 \leq i \leq n\}\) we define \(a_i = (i, 0), i = 0, \ldots, n\). Further, \(\lambda = L = (n + 1, 0)\) and \(\delta = \sqrt{n + 1}/n\).

Since the number of points in \(S\) is polynomial in \(n\) it is clear that this instance can be created in polynomial time. Next we consider the "if" and the "only if" parts separately.

Hence if \(P\) exists and we have a yes-Partition instance it is clear that the corresponding 2-BMPST instance is also a yes-instance. This follows when the partition \(S = \{a_0 = \{0\}, \ldots, a_n = \{n\}\}\), (a class 1 partition, as defined below) is considered. The general appearance of \(M[S]\) (see Figure 1(c)) is determined as follows. The points \(l_i\) and \(r_i\) will be connected as illustrated in Figure 1(c), which follows from the fact that \(\delta = \sqrt{n}/n\).

Next consider the remaining points \(a_i\). Any points \(a_i\) will be connected to either \(l_i\) in \(M[S]\) or \(r_i\) in \(M[S]\), since \(r_i\) and \(l_i\) are the points located closest to \(a_i\), follows since \(\lambda = 2 + \sqrt{n}/n\).

Thus, \(|M[S]| = |M[S]_1| + |M[S]_2| + \cdots + |M[S]_n|\) and we have that the created instance is a yes-instance.

Only if we have that \(P\) does not exist and we therefore want to show that the created 2-BMPST is a no-instance. For this two classes of partitions will be examined.
3. Repeated Validation

3.1 Repeated Validation Algorithm

We now consider the approximation algorithm in the form of two steps.

Step 1: Compute $A(S)$ by applying the approximation algorithm to the problem $S$. This step involves applying the approximation algorithm to the original problem $S$. The approximation algorithm is designed to provide a solution that is close to the optimal solution, but it may not be exactly optimal.

Step 2: Repeat Step 1 for at least $r$ times. After each repetition, the approximation algorithm is applied to the same problem $S$.

This process allows us to refine the approximation and potentially improve the solution quality.

3.2 Repeated Validation Theorem

Theorem 3.1: For a problem $S$, the approximation algorithm $A(S)$ can be repeated for at least $r$ times, and the resulting solutions will be within $c$ of the optimal solution.

Proof: Let $A(S)$ be the approximation algorithm applied to the problem $S$. The approximation algorithm is known to be within $c$ of the optimal solution. By repeating the algorithm, we can iteratively improve the solution. After $r$ repetitions, the solution will be within $c$ of the optimal solution. This is because each repetition refines the approximation, leading to a closer estimate of the optimal solution.

3.3 Repeated Validation Analysis

Analysis 3.1: The repeated validation approach allows for a more refined solution, with the tightness of the approximation improving with each iteration. This approach is particularly useful when the initial approximation is not close enough to the optimal solution.

Conclusion: The repeated validation process enhances the accuracy of the approximation algorithm, making it a powerful tool for solving complex optimization problems.
Consider an optimal partition of $S$ into $k$ subsets $S_1, \ldots, S_k$. Merge all subsets that can be connected by edges of length at most $w$. From this we obtain the set $S_1, \ldots, S_{k'}$, where $k' \leq k$ (see Figure 3). Let $m_i$ denote the number of elements from $S_{i-1}, \ldots, S_k$ included in $S_i$. The purpose of studying those new sets is that every component created in Step 1 of CA belongs to exactly one component in $S_1, \ldots, S_k$. A direct consequence of this is that a combination into $k'$ groups equal to $S_1, \ldots, S_{k'}$ must have been tested in Step 2.

Step 4 guarantees that $M(S_1), \ldots, M(S_{k'})$ will be calculated, and we thus MST's will be divided in all possible ways. Thus, a partition will be made such that each $M(S_i)$ will be divided into exactly $m_i$ components. These partitions $S_1$, $\ldots$, $S_{k'}$ into $V_1, \ldots, V_{k'}$. Let $V_i$ be a set in $V_1, \ldots, V_{k'}$ such that $[M(V_i)] = \max_{S_i} |M(S_i)|$. We wish to restrict our attention to exactly one element of the set $S_{i-1}, \ldots, S_k$ to $S_i$. Note that $V_i$ is a subset of exactly one element $S_i$ in $S_{i-1}, \ldots, S_k$. Assume that $M(V_i)$ was created in Step 4 when $M(S_i)$ was divided into $m_i$ components using RVP. Then $M(V_i) \leq \frac{1}{m_i} |M(S_i)|$, according to Lemma 4. Since the partition $V_1, \ldots, V_{k'}$ will always be tested we have $|M(V_i)| \leq \frac{1}{m_i} |M(S_i)| = \frac{1}{m_i} \sum_{S_i} |M(S_i)| i \geq \frac{1}{m_i} \sum_{S_i} |M(S_i)| i = \frac{1}{m_i} |M(S_i)| = \kappa - 1 \omega (v) \geq (1 - \frac{1}{m_i})^{|M(S_i)|} - \frac{1}{m_i} |M(S_i)| i \geq \frac{1}{m_i} |M(S_i)|$.

To obtain a useful bound we need an upper bound on $w$. Consider the situation after Step 1 has been performed. We have $|M(S_i)| \leq \frac{1}{m_i} |M(S_i)| = \frac{1}{m_i} \sum_{S_i} |M(S_i)| i \geq \frac{1}{m_i} \sum_{S_i} |M(S_i)| i \geq \frac{1}{m_i} |M(S_i)| = \kappa - 1 \omega (v) \geq (1 - \frac{1}{m_i})^{|M(S_i)|} - \frac{1}{m_i} |M(S_i)| i \geq \frac{1}{m_i} |M(S_i)|$.

The results in this paper are shown in a geometric setting but can, under some minor restrictions, be shown in a metric setting as well.

4. CONCLUSION

In this paper it was first shown that the $k$-BPMST problem is NP-hard. After this was established, the next step was to design approximation algorithms for the problem. The algorithm is based on partitioning the point set into a constant number of smaller components and then trying all possible combinations of those small components. This approach revealed a $(2 + \epsilon)$ approximation in the case $k = 2$, and a $(2 + \epsilon)$ approximation in the case $k \geq 3$. The time complexity of the algorithm is $O(n \log n)$.

5. REFERENCES


The Eighteenth European Workshop on Computational Geometry April 10-12, 2002, Warsaw, Poland