Jin Akiyama
Mikio Kano
Xuehou Tan (Eds.)

Discrete and Computational Geometry

LNCS 3742

Japanese Conference, JCDCG 2004
Tokyo, Japan, October 2004
Revised Selected Papers

Springer

Lecture Notes in Computer Science 3742

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Jin Akiyama Mikio Kano Xuehou Tan (Eds.)

Discrete and Computational Geometry

Japanese Conference, JCDCG 2004
Tokyo, Japan, October 8-11, 2004
Revised Selected Papers

Springer

Volume Editors

Jin Akiyama
Tokai University, Research Institute of Educational Development
2-28-4 Tomigaya, Shibuya-ku, Tokyo, 151-0063, Japan
E-mail: fwjb5117@mb.infoweb.ne.jp

Mikio Kano
Ibaraki University, Department of Computer and Information Sciences
Nakanarusawa, Hitachi, Ibaraki, 316-8511, Japan
E-mail: kano@cis.ibaraki.ac.jp

Xuehou Tan
Tokai University, School of High-Technology for Human Welfare
317 Nishino, Numazu, Shizuoka 410-0395, Japan
E-mail: tan@wing.ncc.u-tokai.ac.jp

Library of Congress Control Number: 2005936697

CR Subject Classification (1998): I.3.5, G.2, F.2.2, E.1

ISSN 0302-9743
ISBN-10 3-540-30467-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-30467-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11589440 06/3142 5 4 3 2 1 0

Preface

This volume consists of the refereed proceedings of the Japan Conference on Discrete and Computational Geometry (JCDCG 2004) held at Tokai University in Tokyo, Japan, October, 8–11, 2004, to honor János Pach on his 50th year. János Pach has generously supported the efforts to promote research in discrete and computational geometry among mathematicians in Asia for many years. The conference was attended by close to 100 participants from 20 countries.

Since it was first organized in 1997, the annual JCDCG has attracted a growing international participation. The earlier conferences were held in Tokyo, followed by conferences in Manila, Philippines, and Bandung, Indonesia. The proceedings of JCDCG 1998, 2000, 2002 and IJCCGGT 2003 were published by Springer as part of the series *Lecture Notes in Computer Science* (LNCS) volumes 1763, 2098, 2866 and 3330, respectively, while the proceedings of JCDCG 2001 were also published by Springer as a special issue of the journal *Graphs and Combinatorics*, Vol. 18, No. 4, 2002.

The organizers of JCDCG 2004 gratefully acknowledge the sponsorship of Tokai University, the support of the conference secretariat and the participation of the principal speakers: Ferran Hurtado, Hiro Ito, Alberto Márquez, Jiří Matoušek, János Pach, Jonathan Shewchuk, William Steiger, Endre Szemerédi, Géza Tóth, Godfried Toussaint and Jorge Urrutia.

July 2005

Jin Akiyama
Mikio Kano
Xuehou Tan

Organization

The Organizing Committee

Co-chairs: Jin Akiyama and Mikio Kano

Members: Tetsuo Asano, David Avis, Vašek Chvátal, Hiroshi Imai, Hiro Ito, Naoki Katoh, Midori Kobayashi, Chie Nara, Toshinori Sakai, Kokichi Sugihara, Xuehou Tan, Takeshi Tokuyama, Masatsugu Urabe and Jorge Urrutia

Members of the Executive Committee: Takako Kodate, Yoichi Maeda, Haruhide Matsuda and Mari-Jo P. Ruiz

Table of Contents

Matching Points with Circles and Squares

Bernardo M. Ábrego[1], Esther M. Arkin[2], Silvia Fernández-Merchant[1], Ferran Hurtado[3], Mikio Kano[4], Joseph S.B. Mitchell[2], and Jorge Urrutia[5]

[1] Department of Mathematics, California State University, Northridge
[2] Department of Applied Mathematics and Statistics, State University of New York at Stony Brook
[3] Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya
[4] Ibaraki University
[5] Instituto de Matemáticas, Universidad Nacional Autónoma de México

Abstract. Given a class $\mathcal{C}$ of geometric objects and a point set P, a $\mathcal{C}$-*matching* of P is a set $M = \{C_1, \ldots, C_k\}$ of elements of $\mathcal{C}$ such that each C_i contains exactly two elements of P. If all of the elements of P belong to some C_i, M is called a *perfect matching*; if in addition all the elements of M are pairwise disjoint we say that this matching M is *strong*. In this paper we study the existence and properties of $\mathcal{C}$-matchings for point sets in the plane when $\mathcal{C}$ is the set of circles or the set of isothetic squares in the plane.

1 Introduction

Let $\mathcal{C}$ be a class of geometric objects and let P be a point set with n elements $p_1, \ldots, p_n$ in general position, with n even. A $\mathcal{C}$-*matching* of P is a set $M = \{C_1, \ldots, C_k\}$ of elements of $\mathcal{C}$, such that every C_i contains exactly two elements of P. If all of the elements of P belong to some C_i, M is called a *perfect matching*. If in addition all of the elements of M are pairwise disjoint we say that the matching M is *strong*.

Let $G_{\mathcal{C}}(P)$ be the graph whose vertices are the elements of P, two of which are adjacent if there is an element of $\mathcal{C}$ containing them and no other element from P. Then, a perfect matching in $G_{\mathcal{C}}(P)$ in the usual graph-theoretic sense corresponds naturally with our definition of $G_{\mathcal{C}}(P)$-matchings.

There are several natural classes $\mathcal{C}$ of geometric objects of interest, including line segments, isothetic rectangles, isothetic squares, and disks. For these cases, we refer to the corresponding matching M as a *segment-matching*, a *rectangle-matching*, a *square-matching*, or a *circle-matching*, respectively. Notice that these four classes of objects have in common the *shrinkability* property: if there is an object C' in the class that contains exactly two points p and q in P, then there is an object C'' in the class such that $C'' \subset C'$, p and q lie on the boundary of C'', and the relative interior of C'' is empty of points from P. In the case of rectangles we can even assume the points p and q to be opposite corners of C''.

It is easy to see that P always admits a strong segment-matching and a strong rectangle-matching, which in fact are respectively non-crossing matchings in the

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 1–15, 2005.

complete geometric graph induced by P (in the sense in which geometric graphs are defined in [4]) and in the rectangle of influence graph associated with P [3].

On the contrary, the situation is unclear for circles and squares, and gives rise to interesting problems, That is the topic of this paper, in which we study the existence of perfect and non-perfect, strong and non-strong matchings for point sets in the plane when $\mathcal{C}$ is the set of circles or the set of isothetic squares in the plane.

It is worth mentioning that our results on square-matchings prove, as a side effect, the fact that Delaunay triangulations for the L_1 and L_∞ metrics contain a Hamiltonian path, a question that, to best of our knowledge, has remained unsolved since it was posed in the conference version of [2].

2 Matching with Disks

In this section we study circle-matchings. We show that a perfect circle-matching is always possible, but that there are collections of points for which there is no perfect strong circle-matching. We give bounds on the size of the largest strong circle-matching that any set P of n points admits. We also consider the special case in which the point set P is in convex position.

2.1 Existence of Circle-Matchings

First, notice that the fact that two points from P can be covered by a disk that contains no other point in P is equivalent to fact that the two points are neighbors in the Delaunay triangulation of P, $DT(P)$. In other words, when $\mathcal{C}$ is the set of all circles on the plane, the graph $G_{\mathcal{C}}(P)$ is $DT(P)$. Thus, a point set admits a circle-matching if and only if the graph $DT(P)$ contains a perfect matching, which is the case if and only if P has an even number of points, as proved by Dillencourt in 1990 [2]. Therefore we get the following result, which is a direct consequence of Dillencourt's result:

Theorem 1. *Every point set with an even number of elements admits a circle-matching.*

Nevertheless, for even n, a perfect strong circle-matching is not always possible, as we show next. Consider a circle C with unit radius and a point set P with n elements $p_1, \ldots, p_n$, where $p_1 = a$ is the center of C and $p_2, \ldots, p_n$ are points evenly spaced on the boundary of C. The point a must be matched with some point $b \in \{p_2, \ldots, p_n\}$; this forces the rest of points to be matched consecutively (see Figure 1). In particular, the following and preceding neighbors of b on the boundary must be matched using "large" circles, with centers outside of C, and these circles are forced to overlap for n large enough. In fact, elementary trigonometric computations show that this happens exactly for $n \geq 74$.

We do not describe here the details of the preceding construction. The underlying idea of the construction, however, can be extended to yield an arbitrarily large set of points such that at most a certain fraction of the points can be strongly matched. More precisely, the following result holds:

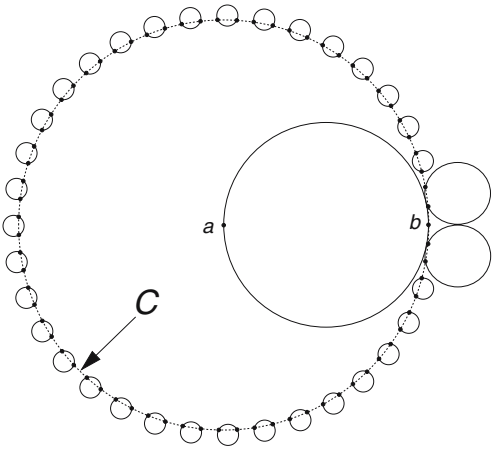


Fig. 1. The elements of a set S are $n-1$ points evenly distributed on C and the center of C. For $n \geq 74$ this point set does not admit a strong perfect circle-matching.

Theorem 2. *There is an n-element point set in the plane, where n can be arbitrarily large, such that at most a fraction $\frac{72}{73}n$ of its points can be strongly circle-matched.*

The proof of this result is omitted from this extended abstract, since it is very long and requires several technical lemmas.

2.2 Subsets That Can Be Strongly Matched

According to Theorem 2, not every point set P admits a strong circle-matching. In this subsection, we prove that for any point set P one can always find a linear number of disjoint disks each one covering exactly two points from P:

Theorem 3. *For any set P of n points in general position, there is a strong circle-matching using at least $2\lceil (n-1)/8 \rceil$ points of P.*

The proof utilizes several lemmas. Let M be a matching of $m = \lfloor n/2 \rfloor$ pairs of (distinct) points of P that minimizes the sum of the squared Euclidean distances between pairs of matched points. Let the m pairs in the matching M be $(p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m)$. We let $|p_i q_i|$ denote the Euclidean distance between p_i and q_i. We let $D_i = DD(p_i q_i)$ be the diametrical disk, with segment $p_i q_i$ as diameter, and let o_i denote the center of this disk. Let $\mathcal{C} = \{D_1, D_2, \ldots, D_m\}$ be the set of diametrical disks determined by the pairs (p_i, q_i) of the matching M.

Lemma 1. *If $DD(ab), DD(cd) \in \mathcal{C}$ then $\{c, d\} \nsubseteq DD(ab)$.*

Proof. Suppose that $c, d \in DD(ab)$. Note that $\angle dcb + \angle bdc < \pi$, so we may assume that $\angle dcb < \pi/2$. Thus $|bd|^2 < |cd|^2 + |bc|^2$, and since $c \in DD(ab)$, we know that $\angle bca \geq \pi/2$ and $bc^2 + ac^2 \leq ab^2$. Combining these inequalities we get $|bd|^2 + |ac|^2 < |ab|^2 + |cd|^2$, contradicting the optimality of M. □

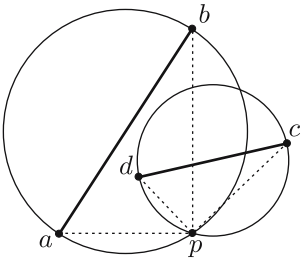


Fig. 2. Proof of Lemma 2

Lemma 2. *If $DD(ab), DD(cd) \in \mathcal{C}$ and p is in the intersection of the bounding circles of $DD(ab)$ and $DD(cd)$, then triangles $\triangle apb$ and $\triangle dpc$ do not overlap.*

Proof. Suppose that $\triangle apb$ and $\triangle dpc$ overlap. Assume $\overrightarrow{pd}$ is between $\overrightarrow{pb}$ and $\overrightarrow{pa}$ as in Figure 2. Since $\overline{ab}$ and $\overline{cd}$ are diameters of their respective circles, we know that $\angle dpc = \angle apb = \pi/2$, implying that $\angle apd < \pi/2$ and $\angle bpc < \pi/2$. Then $|ad|^2 < |pa|^2 + |pd|^2$, $|bc|^2 < |pb|^2 + |pc|^2$, and

$$|ad|^2 + |bc|^2 < |pa|^2 + |pb|^2 + |pc|^2 + |pd|^2 = |ab|^2 + |cd|^2,$$

which contradicts the optimality of M. □

Lemma 3. *No three disks in $\mathcal{C}$ have a common intersection.*

Proof. Suppose $I = DD(p_1q_1) \cap DD(p_2q_2) \cap DD(p_3q_3) \neq \varnothing$. By Lemma 1, the boundary of I must contain sections of at least two of the bounding circles of $DD(p_1q_1)$, $DD(p_2q_2)$ and $DD(p_3q_3)$. Thus, we may assume there is a point $p \in I$ such that p is in the intersection of the bounding circles of $DD(p_1q_1)$ and $DD(p_2q_2)$. By Lemma 2 the triangles $\triangle pp_1q_1$ and $\triangle pp_2q_2$ do not overlap. Now we consider three cases depending on the number of triangles that overlap with $\triangle pp_3q_3$.

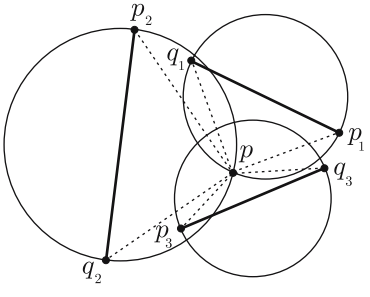


Fig. 3. Case 1: $\triangle pp_3q_3$ does not overlap with $\triangle pp_1q_1$ or with $\triangle pp_2q_2$

Case 1. $\triangle pp_3q_3$ does not overlap with $\triangle pp_1q_1$ or with $\triangle pp_2q_2$.

We may assume that the relative order of the triangles $\triangle pp_iq_i$ is as in Figure 3. Then, since $\angle q_1pp_1$, $\angle q_2pp_2$, $\angle q_3pp_3 \geq \pi/2$, we have that

$$\angle p_2pq_1 + \angle p_3pq_2 + \angle p_1pq_3 \leq \pi/2.$$

Thus, each of these angles is at most $\pi/2$, and at least two of them strictly acute (or zero). Thus,

$$|q_1p_2|^2 + |q_2p_3|^2 + |q_3p_1|^2 < |pq_1|^2 + |pp_2|^2 + |pq_2|^2 + |pp_3|^2 + |pq_3|^2 + |pp_1|^2.$$

Also, since none of the angles $\angle q_1pp_1, \angle q_2pp_2, \angle q_3pp_3$ is acute,

$$|pp_1|^2 + |pq_1|^2 + |pp_2|^2 + |pq_2|^2 + |pp_3|^2 + |pq_3|^2 \leq |p_1q_1|^2 + |p_2q_2|^2 + |p_3q_3|^2.$$

Thus, $|q_1p_2|^2 + |q_2p_3|^2 + |q_3p_1|^2 < |p_1q_1|^2 + |p_2q_2|^2 + |p_3q_3|^2$, which contradicts the optimality of M.

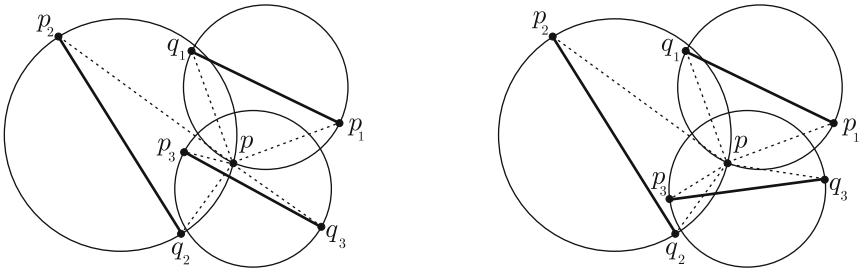


Fig. 4. Case 2: $\triangle pp_3q_3$ overlaps with $\triangle pp_2q_2$, but not with $\triangle pp_1q_1$

Case 2. $\triangle pp_3q_3$ overlaps with $\triangle pp_2q_2$, but not with $\triangle pp_1q_1$.

Assume $\overrightarrow{pp_3}$ is between $\overrightarrow{pp_2}$ and $\overrightarrow{pq_2}$. We may also assume that $\angle q_3pp_3 > \pi/2$; otherwise, p is in the bounding circle of $DD(p_3q_3)$ and then, by Lemma 2, $\triangle pp_3q_3$ and $\triangle pp_2q_2$ do not overlap. Since $\angle q_3pp_3 > \pi/2$, $|p_3q_3|^2 > |pp_3|^2 + |pq_3|^2$.

If $\angle q_3pq_2 \leq \pi/2$ (Figure 4a) then, as in the proof of Lemma 2, $|q_2q_3|^2 \leq |pq_2|^2 + |pq_3|^2$, $|p_2p_3|^2 \leq |pp_2|^2 + |pp_3|^2$, and thus

$$|q_2q_3|^2 + |p_2p_3|^2 \leq |pp_2|^2 + |pq_2|^2 + |pp_3|^2 + |pq_3|^2 < |p_2q_2|^2 + |p_3q_3|^2,$$

which contradicts the optimality of M.

If $\angle q_3pq_2 > \pi/2$ (Figure 4b), then $\angle p_1pq_3 + \angle p_2pq_1 < \pi/2$. Thus, $\angle p_1pq_3$, $\angle p_2pq_1 < \pi/2$, and thus $p_1q_3^2 < pp_1^2 + pq_3^2$ and $p_2q_1^2 < pp_2^2 + pq_1^2$. Also, since $\overrightarrow{pp_3}$ is between $\overrightarrow{pp_2}$ and $\overrightarrow{pq_2}$, $\angle q_2pq_3 < \pi/2$ and $q_2p_3^2 < pq_2^2 + pp_3^2$. Putting all of these inequalities together, we get

$$|p_1q_3|^2 + |p_2q_1|^2 + |q_2p_3|^2 < |pp_1|^2 + |pq_1|^2 + |pp_2|^2 + |pq_2|^2 + |pp_3|^2 + |pq_3|^2.$$

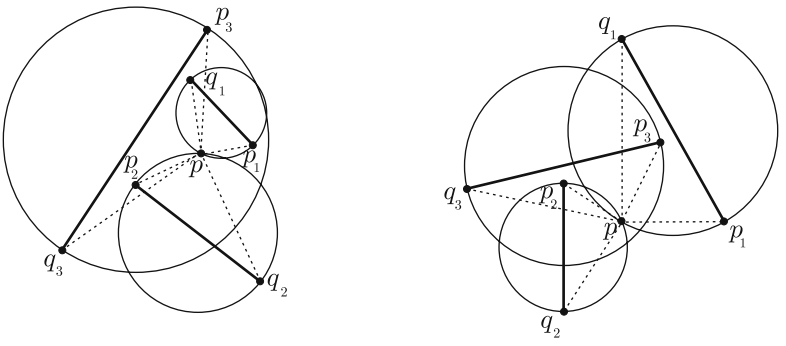


Fig. 5. Case 3: $\triangle pp_3q_3$ overlaps $\triangle pp_2q_2$ and $\triangle pp_1q_1$

Moreover, $|pp_1|^2+|pq_1|^2 = |p_1q_1|^2$, $|pp_2|^2+|pq_2|^2 = |p_2q_2|^2$, and $|pp_3|^2+|pq_3|^2 < |p_3q_3|^2$. Thus,

$$|p_1q_3|^2 + |p_2q_1|^2 + |q_2p_3|^2 < |p_1q_1|^2 + |p_2q_2|^2 + |p_3q_3|^2,$$

which contradicts the optimality of M.

Case 3. $\triangle pp_3q_3$ overlaps $\triangle pp_2q_2$ and $\triangle pp_1q_1$.

We may assume that $\overrightarrow{pp_3}$ (resp., $\overrightarrow{pq_3}$) is between $\overrightarrow{pp_1}$ and $\overrightarrow{pq_1}$ (resp., $\overrightarrow{pp_2}$ and $\overrightarrow{pq_2}$). Refer to Figure 5. Again, by Lemma 2, we may assume that $\angle q_3pp_3 > \pi/2$ and $|p_3q_3|^2 > |pp_3|^2 + |pq_3|^2$.

If $\angle p_1pq_2 \leq \pi/2$ (Figure 5a), then $|p_1q_2|^2 \leq |pp_1|^2+|pq_2|^2$. From the locations of p_3 and q_3, we have that $\angle q_1pp_3$, $\angle q_3pp_2 < \pi/2$, so $|p_3q_1|^2 < |pq_1|^2 + |pp_3|^2$ and $|p_2q_3|^2 < |pp_2|^2 + |pq_3|^2$. Thus,

$$|p_1q_2|^2 + |p_3q_1|^2 + |p_2q_3|^2 < |pp_1|^2 + |pq_1|^2 + |pp_2|^2 + |pq_2|^2 + |pp_3|^2 + |pq_3|^2.$$

As $|pp_i|^2 + |pq_i|^2 = |p_iq_i|^2$ for $i = 1, 2$ and $|pp_3|^2 + |pq_3|^2 < |p_3q_3|^2$, we get

$$|p_1q_2|^2 + |p_3q_1|^2 + |p_2q_3|^2 < |p_1q_1|^2 + |p_2q_2|^2 + |p_3q_3|^2.$$

If $\angle p_1pq_2 > \pi/2$ (Figure 5b), then, similarly, we get

$$|p_2q_1|^2 + |p_1p_3|^2 + |q_2q_3|^2 < |p_1q_1|^2 + |p_2q_2|^2 + |p_3q_3|^2.$$

In both cases we get a contradiction to the optimality of M. □

Lemma 4. *If $D_1, D_2, D_3, D_4 \in \mathcal{C}$ with $D_1 \cap D_2 \neq \varnothing$ and $D_3 \cap D_4 \neq \varnothing$ then the segments $\overline{o_1o_2}$ and $\overline{o_3o_4}$ do not intersect.*

Proof. Suppose $\overline{o_1o_2}$ and $\overline{o_3o_4}$ intersect at a point c, and let $p \in D_1 \cap D_2 \cap \overline{o_1o_2}$, and $q \in D_3 \cap D_4 \cap \overline{o_3o_4}$. Assume that $p \in \overline{co_2}$ and $q \in \overline{co_4}$. By the triangle inequality, $|o_1q| \leq |o_1c| + |cq|$ and $|o_3p| \leq |o_3c| + |cp|$; thus,

$$|o_1q| + |o_3p| \leq |o_1c| + |cp| + |o_3c| + |cq| = |o_1p| + |o_3q|.$$

Thus, either $|o_1q| \leq |o_1p|$ or $|o_3p| \leq |o_3q|$, which implies that either $q \in D_1$ or $p \in D_3$. This is a contradiction to Lemma 3, since either $q \in D_1 \cap D_3 \cap D_4$ or $p \in D_1 \cap D_2 \cap D_3$. □

Proof of Theorem 3. Let G be the graph with vertex set equal to the set of centers of the disks in $\mathcal{C}$ and with two vertices connected by an edge if and only it the corresponding disks intersect. By Lemma 4, G is a planar graph. Then, by the Four Color Theorem, the maximum independent set of G has at least $\lceil m/4 \rceil = \lceil \lfloor n/2 \rfloor /4 \rceil = \lceil (n-1)/8 \rceil$ vertices. Thus, the corresponding diametrical disks are pairwise disjoint. Therefore, P has a circle-matching using at least $2\lceil (n-1)/8 \rceil$ points. It may happen that these diametrical disks have points of P in their interior; however, it is always possible to find a circle inside one of these diametrical disks containing only two points of P.

2.3 Convex Position

For n points on a line, with n even, it is obvious that a strong perfect matching with disks is always possible, as we can simply take the diametrical circles defined by consecutive pairs. As a consequence a strong perfect matching is also always possible when we are given any set P of n points lying on a circle C: using an inversion with center at any point in $C \setminus P$ the images of all points from P become collinear and admit a matching, which, applying again the same inversion, gives the desired matching (because inversions are involutive and map circles that do not pass through the center of inversion to circles).

This may suggest that a similar result should hold for any set of points in convex position, but this is not the case, as we show next using the same kind of arguments.

Let Q be a point set consisting of the center a of a circle C, and 73 additional points evenly distributed on C; as remarked earlier (see Figure 1), Q does not admit a strong perfect circle-matching.

Let P be the point set obtained from Q by applying any inversion with center at some point $p \in C \setminus Q$; the point set P does not admit a strong perfect circle-matching. Note that all of the points in P with the exception of the image of a lie on a line. Applying an infinitesimal perturbation to the elements of P in such a way that they remain in convex position but no three are collinear produces a point set P' in convex position for which no strong perfect circle-matching exists, since the inverse set Q' is an infinitesimal perturbation of Q and therefore does not admit a strong perfect circle-matching. Therefore we have proved the following result:

Proposition 1. *There are point sets in convex position in the plane for which there is no strong perfect circle-matching.*

3 Isothetic Squares

In this section we consider the following variation of our geometric matching problem. Let P be a set of $2n$ points in general position in the plane. As in the

previous section, we define a graph $G(P)$ in which the points P are the vertices of $G(P)$ and two points are adjacent if and only if there is an isothetic square containing them that does not contain another element of P.

3.1 Existence of Square-Matchings

We show here that P always admits a square-matching. We need a result that is part of folklore, yet we prove here for the sake of completeness:

Lemma 5. *$G(P)$ is planar.*

Proof. For any two points $q_s, q_t \in P$, let us denote by $R_{s,t}$ the smallest isothetic rectangle containing them. Suppose now that q_i is adjacent to q_j, and q_k to q_l, and that the segments joining q_i to q_j and q_k to q_l intersect; it is straightforward to see that $R_{i,j}$ and $R_{k,l}$ have to cross each other. Let us denote by a' and a'' the base and height of $R_{i,j}$, and by b' and b'' the base and height of $R_{k,l}$. Then, if we assume without loss of generality that $b'' = \max\{a', a'', b', b''\}$, any square containing q_k and q_l contains q_i or q_j, a contradiction. □

Let C be a square that contains all of the elements of P in its interior, and let $P^{'}$ be the point set obtained by adding to P the vertices of C. Let G be the graph obtained from $G(P')$ by adding an extra point p_∞ adjacent to the vertices of C. We are going to see that G is 4-connected; first, we prove a technical lemma.

Lemma 6. *Let S be a point set containing the origin O and a point p from the first quadrant, such that all of the others points in S lie in the interior of the rectangle R with corners at O and p. Then there is path in $G(S)$ from O to p such that every two consecutive vertices can be covered by an isothetic square contained in R, empty of any other point from S.*

Proof. The proof is by induction on $|S|$. If $|S| = 2$ the result is obvious. If $|S| > 2$ we grow homothetically from O a square with bottom left corner at O until a first point q from S, different from O, is encountered. This square is contained in R and gives an edge in $G(S)$ between O and q; now we apply induction to the points from S covered by the rectangle with q and p as opposite corners. □

Obviously, the preceding result can be rephrased for any of the four quadrants to any point taken as origin. We are now ready to prove the following result:

Lemma 7. *G is 4-connected.*

Proof. We show that the graph G' resulting from the removal of any three vertices from G is connected.

Suppose first that none of the suppressed vertices is p_∞, and let us show that p_∞ can be reached from any vertex of G'. If a vertex $v \in G'$ is a corner of C, then it is adjacent to p_∞. If v is not such a corner, consider the four quadrants it defines. In at least one of them no vertex from G has been suppressed. Thus, we can apply Lemma 6 to this quadrant and obtain a path in G' from v to a surviving corner of C; from there we arrive to p_∞.

If we suppress from G two points from P and p_∞, then G' contains the 4-cycle given by the corners of C. From any vertex $v \in P$ in G' we can reach one of these corners (and therefore any of them), because in at least two of the quadrants relative to v no vertex has been removed.

The cases in which p_∞ and one or two corners of C are suppressed are handled similarly. □

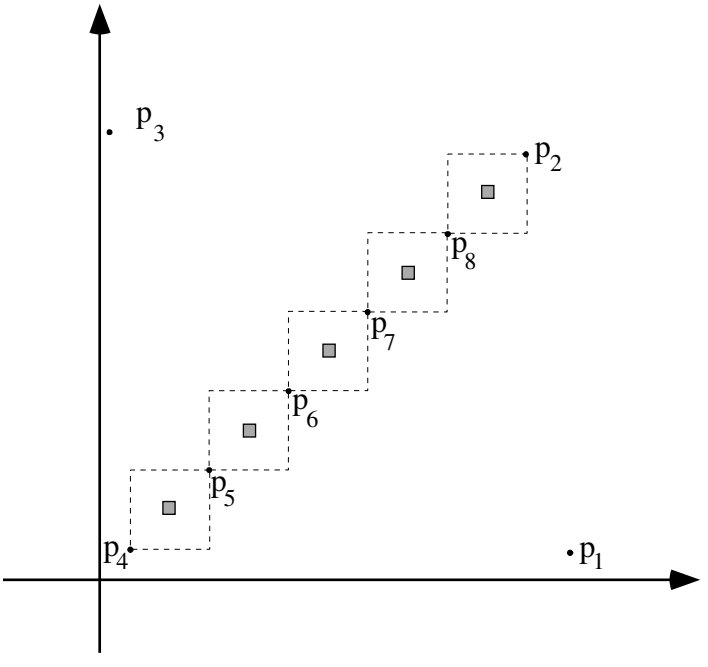


Fig. 6. Final step in showing the existence of square-matchings

Since it is clear that G is planar, it now follows using a classic result of Tutte [7] that G is Hamiltonian. This almost proves our result, since the removal of p_∞ from G results in a graph that has a Hamiltonian path. Using this path, we can now obtain a perfect matching in $G(P')$. A remaining issue is to find a matching in $G(P)$, which requires that we identify the elements of P to be matched to the corners of C.

We address this issue in a way similar to that used in [1]. Consider the five shaded squares and eight points $p_1, \ldots, p_8$ (represented by small circles) as shown in Figure 6. Within each of the shaded squares, place a copy of P, and let P'' be the point set containing the points of the five copies of P plus $p_1, \ldots, p_8$. Consider the graph $G(P'')$ and add to it a vertex p_∞ adjacent to p_1, p_2, p_3, p_4. The resulting graph $G(P'')$ is planar and 4-connected, so by Tutte's Theorem it is Hamiltonian. The removal of p_∞ gives a Hamiltonian path w in the resulting graph, with extremes in the set $\{p_1, p_2, p_3, p_4\}$. At least one of the five copies

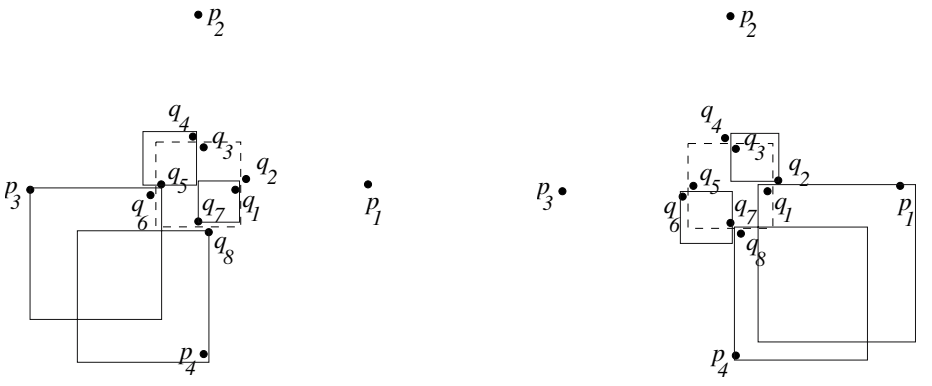


Fig. 7. Twelve points that do not admit a perfect strong square-matching

of P does not contain any neighbor of p_1 or p_3 in the path, because these two vertices have altogether at most four neighbors. Suppose, for example, that this is the case for the copy of P inside the box between p_5 and p_6; since the path has to arrive to the points inside the box and leave the set, and this can only be done through p_5 and p_6, the points inside the box have to be completely visited by the path w before leaving the box. Thus, we have obtained a Hamiltonian path in $G(P)$, which gives a perfect matching in $G(P)$, and thus we have proved:

Theorem 4. *P has a perfect square-matching.*

Remark. Under standard non-degeneracy assumptions the graph $G(P)$ is simply the Delaunay triangulation for the L_∞ metric (or the L_1 metric, after a 45-degree rotation). Therefore, our preceding considerations prove that these triangulations admit a Hamiltonian path, a question that to best of our knowledge has remained unsolved since it was posed in the conference version of [2]).

3.2 Subsets That Can Be Strongly Square-Matched

We show first a family of 12 points that allows no perfect strong square-matching. Consider the point set with twelve points shown in Figure 7: there are four extreme points, labelled $p_1, ..., p_4$ and eight more points that are very close to the midpoints of an auxiliary dashed square as shown in the same figure; four of them, q_1, q_3, q_5, q_7, are *internal*, while four of them, q_2, q_4, q_6, q_8, are *external*. The points can all be drawn in general position, with no two on a common vertical/horizontal line.

Notice that no pair can be matched inside $\{p_1, ..., p_4\}$; therefore, two pairs have to appear in any matching by picking points from $\{q_1, q_2, ..., q_8\}$. No two external points can be matched, and matching a close pair such as (q_1, q_2) would leave an extreme point (p_1 in the example) without a partner for a matching. Distributing the four internal points into two matched pairs produces always overlapping rectangles.

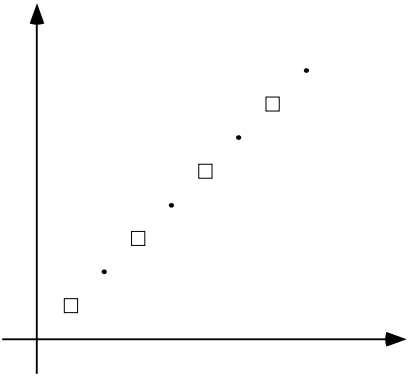


Fig. 8. Extending the 12-point example for strong square-matchings

This leaves only two possibilities for two pairs taken from $\{q_1, q_2, ..., q_8\}$: either using three internal points and one external point, or using two internal points and two external points. In the first case, the situation must be as in Figure 7, left, where q_1 is matched to q_7 and q_4 to q_5; this forces (p_3, q_6) and (p_4, q_8) to be matched pairs and causes overlap. In the second case, the situation must be as in Figure 7, right, where q_2 is matched to q_3 and q_6 to q_7; this forces (p_1, q_1) and (p_4, q_8) to be matched pairs and causes overlap. This concludes the proof.

We now prove that the preceding result can be used to construct arbitrarily large sets that do not admit perfect strong square-matchings:

Proposition 2. *There are sets with* $13m$ *points such that any strong square-matching of them contains at most* $6m$ *pairs of matched points.*

Proof. On the line $y = x$ consider the points with coordinates (i, i), $i = 1, ..., 2m$, with m even. For $j = 2i + 1$, $i = 0, ..., m - 1$, consider an ϵ-neighborhood around the point (j, j) and insert a copy of the 12-point configuration P_{12} (scaled down to fit within this ϵ-neighborhood). The remaining points (i, i), i even, remain as singletons. Let P be the point set containing all of these $12m + m$ points, and let M be a strong square-matching of P. See Figure 8

Observe that the 12-point set close to the point (1,1) cannot be matched within themselves. Thus, M matches at most 10 of these points. This leaves two points pending. One of these points can be matched with point $(2, 2)$. The remaining point cannot be square-matched with any point in P. Similarly, one of the points in the ϵ-neighborhood of $(2i + 1, 2i + 1)$ cannot be matched to any element of P. This leaves at least m elements of P unmatched in M. Our result follows. □

We determine next a lower bound on the number of points of a point set that can always be strongly square-matched.

Theorem 5. *For any set P of n points in general position, there is a strong square-matching using at least $2\lceil \frac{n}{5} \rceil$ points of P.*

In fact, we prove a slightly stronger result, from which the preceding theorem is immediately derived:

Lemma 8. *Let S be a square that contains a set P of $n \geq 2$ elements. Then it is always possible to find a strong square-matching of P with $\lceil \frac{n}{5} \rceil$ elements.*

Proof. The claim is obviously true for $n = 2$. Suppose that it is true for sets P of size $n-1$ ($n-1 \geq 2$); we now prove it is true for sets of size n. Observe first that if $n = 5k + i$, $i = 2, 3, 4, 5$ then $\lceil \frac{n}{5} \rceil = \lceil \frac{n-1}{5} \rceil$, and, by induction, we are done. Suppose then that $n = 5k + 1$ for some k.

Partition S into four squares, S_1, S_2, S_3, S_4, of equal size. Assume that square S_i contains r_i points. If each r_i is greater than 2, or equal to zero, we are done, since for any integers such that $r_1 + ... + r_4 = n$ we have

$$\left\lceil \frac{r_1}{5} \right\rceil + ... + \left\lceil \frac{r_4}{5} \right\rceil \geq \left\lceil \frac{n}{5} \right\rceil .$$

Suppose then that some of the r_i's are one. A case analysis follows.

Case 1. Three elements of the set $\{r_1, r_2, r_3, r_4\}$ are equal to one; say, $r_2 = r_3 = r_4 = 1$, and $r_1 = 5(k-1) + 3$.

Let S_1' be the smallest square, one of whose corners is p, containing all elements of P in S_1 except for one point, say p_1. Suppose, without loss of generality, that p_1 lies below the horizontal line through the bottom edge of S_1'. Then S_1' contains $5(k-1)+2$ points, and thus, by induction, we can find k disjoint squares in that square containing exactly two elements of P_n. It is easy to see that there is a square contained in $S - S_1'$ that contains p_1 and the element of P_n in S_3. This square contains a square that contains exactly two elements of P. See Figure 9.

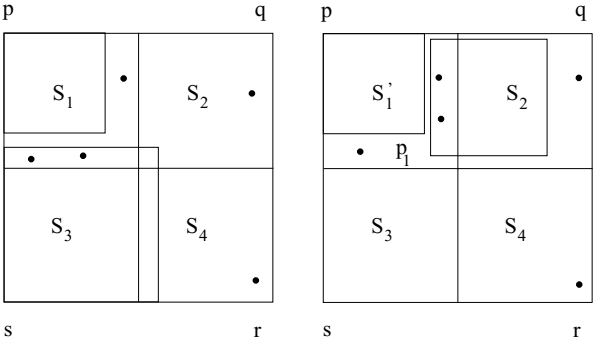


Fig. 9. Proof of Case 1 in Lemma 8

Case 2. Two elements of $\{r_1, r_2, r_3, r_4\}$ are equal to one.

Suppose that r_i and r_j are not 1. Observe that $r_i + r_j = 5k - 1$ and that $\lceil \frac{r_i}{5} \rceil + \lceil \frac{r_j}{5} \rceil \geq \lceil \frac{n-1}{5} \rceil$. If $\lceil \frac{r_i}{5} \rceil + \lceil \frac{r_j}{5} \rceil > \lceil \frac{n-1}{5} \rceil = k$, we are done. Suppose then that $\lceil \frac{r_i}{5} \rceil + \lceil \frac{r_j}{5} \rceil = \lceil \frac{n-1}{5} \rceil = k$; this happens only if one of them, say $r_i = 5r$ and the other element $r_j = 5s - 1$ for some r and s greater than or equal to zero.

Up to symmetry two subcases arise: (i). $r_1 = 5r$ and $r_3 = 5s - 1$; and (ii). $r_1 = 5r$ and $r_4 = 5s - 1$.

In case (i) let $S_1^{'}$ be the smallest square contained in S_1 that contains all but three of the elements, say p_1, p_2 and p_3, of P in S_1, such that p is a vertex of $S_1^{'}$. If two of these elements, say p_1 and p_2, are below the horizontal line through the lower horizontal edge of $S_1^{'}$, then there is a square $S_3^{'}$ contained in $S - S_1^{'}$ that contains all of the elements of P in S_3 and also contains p_1 and p_2; see Figure 10(a). Then, by induction, we can find in $S_1^{'}$ and $S_3^{'}$ $\lceil \frac{5r-3}{5} \rceil = r$ and $\lceil \frac{5s+1}{5} \rceil = s + 1$ disjoint squares, i.e. $r + s + 1 = k + 1$ disjoint squares contained in S each of which contains exactly two elements of P_n.

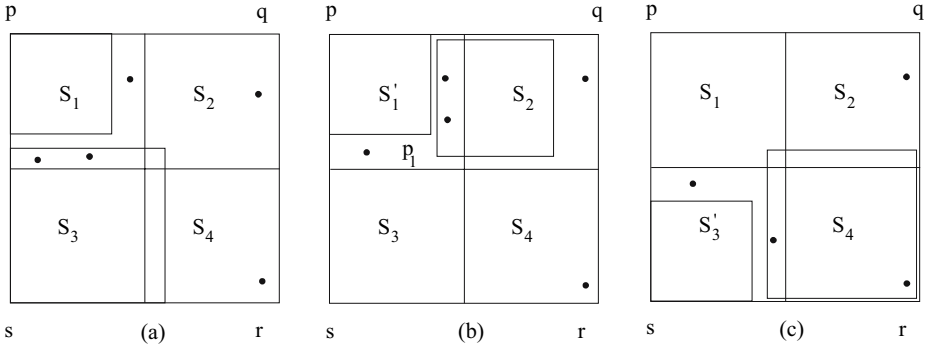


Fig. 10. Proof of Case 2 in Lemma 8

If no two elements of p_1, p_2 and p_3 lie below the horizontal line through the lower horizontal edge of $S_1^{'}$, then there is a square contained in $S_1 \cup S_2 - S_1^{'}$ that contains two of these elements. Applying induction to the elements of P in $S_1^{'}$, the elements of P in S_3 and the square we just obtained prove our result. See Figure 10(b).

If $r = 0$, and thus $s > 0$, choose $S_3^{'}$ such that it contains all but two points of P_n in S_3. If two points in S_3 lie above line containing the top edge of $S_3^{'}$ or to the right of the line L containing the rightmost vertical edge of $S_3^{'}$, an analysis similar to the one above follows. Suppose then that there is exactly one point in S_3 to the right of L. Then $S_3^{'}$ contains $5s - 3 \geq 2$ points, and there is a square contained in S containing the point of P_n in S_4. See Figure 10(c). By induction on the number of elements in $S_3^{'}$, and using the last square we obtained, our result follows.

Case (ii) can be solved similarly.

Case 3. The remaining case, when only one of $\{r_1, r_2, r_3, r_4\}$ is one, can be solved in a similar way to the previous cases. For example, the case when only $r_4 = 1$, (in which case r_1, r_2 and r_3 are multiples of 5) $r_1 \neq 0$, and $r_2 = 0$ is solved in almost the same way as case 2(i). We leave the details to the reader. □

3.3 A Perfect Strong Square-Matching for the Convex Case

When several points may have the same x-coordinate or the same y-coordinate, a perfect strong matching is not always possible., For example, in the point configuration of Figure 11 p cannot be matched (using a square) to any of the points on the horizontal line.

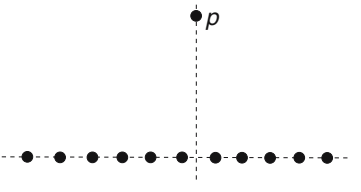


Fig. 11. Point p cannot be matched

Nevertheless, we can prove that for points in convex position having no repeated coordinates a perfect strong matching always exists:

Theorem 6. *Any even-cardinality set P of points in the plane in convex position, with no two points on a common vertical or horizontal line, admits a perfect strong square-matching.*

The proof of this result is omitted from this extended abstract, since it is long and requires several technical lemmas.

4 Open Problems

We have proved that (weak) matchings with circles and isothetic squares are always possible. It is natural to ask which other classes of convex objects enjoy the same property and to try to characterize them. On the algorithmic side, there are also decision and construction problems that are very interesting. These issues are the main lines of our ongoing research on this topic.

Remarks and Acknowledgments

It has been pointed out to us that the question of whether every sufficiently large even-cardinality set of points in the plane has a strong perfect circle-matching

was recently raised independently by the participants of a school mathematics club in Budapest run by Luis Posa, and that, as a consequence, Theorem 2 was found independently by Andras Lazar and János Pach.

The research in this paper was initiated at the International Workshop on Combinatorial Geometry at the Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Jun. 30–Jul. 4, 2003. The authors would like to thank the other workshop participants, namely, Gabriela Araujo, Elsa Omaña-Pulido, Eduardo Rivera-Campo and Pilar Valencia for helpful discussions.

E. Arkin and J. Mitchell acknowledge support from the National Science Foundation (CCR-0098172, CCF-0431030). In addition, J. Mitchell acknowledges support from the U.S.-Israel Binational Science Foundation (grant No. 2000160), NASA Ames Research (NAG2-1620), NSF grant ACI-0328930, and Metron Aviation. Ferran Hurtado is partially supported by Projects MEC-MCYT-FEDER BFM2003-00368 and Gen. Cat 2001SGR00224. Jorge Urrutia is partially supported by CONACYT of Mexico grant 37540-A.

References

1. J. Czyzowicz, E. Rivera-Campo, J. Urrutia and J. Zaks, *Guarding rectangular art galleries*, Discrete Mathematics, 50, 1994, 149-157.
2. M. Dillencourt, *Toughness and Delaunay Triangulations*, Discrete and Computational Geometry, 5(6), 1990, 575-601. Preliminary version in Proc. of the 3rd Annual Symposium on Computational Geometry, Waterloo, 1987, 186-194.
3. G. Liotta, A. Lubiw, H. Meijer and S. H. Whitesides, *The Rectangle of Influence Drawability Problem*, Discrete and Computational Geometry, 5(6), 1990, 575-601.
4. J. Pach, Editor, *Towards a Theory of Geometric Graphs*, Amer. Math. Soc., Contemp. Math. Series/342, 2004.
5. F. P. Preparata and M. I. Shamos, *Computational Geometry. An Introduction*, Springer-Verlag, 1995.
6. M. Sharir, *On k-Sets in Arrangements of Curves and Surfaces*, Discrete and Computational Geometry, 6, 1991, 593-613.
7. W. T. Tutte. *A theorem on planar graphs*, Trans. Amer. Math. Soc. 82, 1956, 99-116.

The Minimum Manhattan Network Problem: A Fast Factor-3 Approximation*

Marc Benkert, Alexander Wolff, and Florian Widmann

Faculty of Computer Science, Karlsruhe University, P.O. Box 6980,
D-76128 Karlsruhe, Germany
i11www.ira.uka.de/algo/group

Abstract. Given a set of nodes in the plane and a constant $t \geq 1$, a Euclidean *t-spanner* is a network in which, for any pair of nodes, the ratio of the network distance and the Euclidean distance of the two nodes is at most t. These networks have applications in transportation or communication network design and have been studied extensively.

In this paper we study 1-spanners under the Manhattan (or L_1-) metric. Such networks are called *Manhattan networks*. A Manhattan network for a set of nodes can be seen as a set of axis-parallel line segments whose union contains an x- and y-monotone path for each pair of nodes. It is not known whether it is NP-hard to compute minimum Manhattan networks, i.e. Manhattan networks of minimum total length. In this paper we present a factor-3 approximation algorithm for this problem. Given a set of n nodes, our algorithm takes $O(n \log n)$ time and linear space.

1 Introduction

For many applications it is desirable to connect the nodes of a transportation or communication network by short paths within the network. In the Euclidean plane this can be achieved by connecting all pairs of nodes by straight line segments. While the complete graph minimizes node-to-node travel time, it maximizes the network-construction costs. An interesting alternative are Euclidean *t-spanners*, i.e. networks in which the ratio of the network distance and the Euclidean distance between any pair of nodes is bounded by a constant $t \geq 1$. Euclidean spanners have been studied extensively, and researchers have tried to combine the spanner property with other desirable properties, such as small node degree, small total edge length, and small diameter. Euclidean spanners with one or more of these properties can be constructed in $O(n \log n)$ time [1], where n is the number of nodes.

Under the Euclidean metric, in a 1-spanner (which is the complete graph) the location of each edge is uniquely determined. This is not the case in the Manhattan (or L_1-) metric, where an edge $\{p, q\}$ of a 1-spanner is a *Manhattan p–q path*, i.e. an x- and y-monotone path between p and q. A 1-spanner under the Manhattan metric for a set $P \subset \mathbb{R}^2$ is called a *Manhattan network* (MMN)

* This work was supported by grant WO 758/4-1 of the German Science Foundation.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 16–28, 2005.

and can be seen as a set of axis-parallel line segments whose union contains a Manhattan p–q path for each pair $\{p, q\}$ of points in P.

In this paper we investigate how the extra degree of freedom in routing edges can be used to construct Manhattan networks of minimum total length, so-called *minimum Manhattan networks* (MMN). The MMN problem may have applications in city planning or VLSI layout. Lam et al. [5] also describe an interesting application in computational biology. Approximation algorithms for the MMN problem have been considered before. Gudmundsson et al. [3] have proposed an $O(n \log n)$-time factor-8 and an $O(n^3)$-time factor-4 approximation algorithm. Later, Kato et al. [4] have given an $O(n^3)$-time factor-2 approximation algorithm. However, their correctness proof is incomplete. It is not known whether the MMN problem is NP-hard.

In this paper we present an $O(n \log n)$-time factor-3 approximation algorithm. We use some of the ideas of [4], but our algorithm is simpler, faster and uses only linear (instead of quadratic) storage. The main novelty of our approach is that we partition the plane into two regions and compare the network computed by our algorithm to an MMN in each region separately.

This paper is structured as follows. In Section 2 we give some basic definitions and observations. In Section 3 we show how the backbone of our network is computed. We describe the algorithm in Section 4 and analyze it in Section 5.

2 Preliminaries

We use $|M|$ to denote the total length of a set M of line segments. For all such sets M we assume throughout the paper that each segment of M is inclusion-maximal with respect to $\bigcup M$. It is not hard to see that for every Manhattan network M there is a Manhattan network M' with $|M'| \leq |M|$ that is contained in the grid induced by the input points, i.e. M' is a subset of the union U of the horizontal and vertical lines through the input points. Therefore we will only consider networks contained in U. It is clear that any meaningful Manhattan network of a point set P is contained in the bounding box $\mathrm{BBox}(P)$ of P. Finding a Manhattan network for given P is trivial, e.g. the parts of U within $\mathrm{BBox}(P)$ yield a Manhattan network. However, this network can be n times longer than an MMN, as the point set $\{(1,1), \ldots, (n,n)\}$ shows.

We will use the notion of a *generating set* [4]. A generating set Z is a subset of $P \times P$ with the property that a network containing Manhattan paths for all pairs in Z is already a Manhattan network of P. The authors of [4] define a linear-size generating set Z. We use the same generating set Z, but more intuitive names for the subsets of Z. We define $Z = Z_{\mathrm{hor}} \cup Z_{\mathrm{ver}} \cup Z_{\mathrm{quad}}$. These subsets are defined below. Our algorithm will establish Manhattan paths for all point pairs of Z—first for those in $Z_{\mathrm{hor}} \cup Z_{\mathrm{ver}}$ and then for those in Z_{quad}.

Definition 1 (Z_{ver})**.** *Let $P = \{p_1, \ldots, p_n\}$ be the set of input points in lexicographical order, where $p_i = (x_i, y_i)$. Let $x^1 < \cdots < x^u$ be the sequence of x-coordinates of the points in P in ascending order. For $i = 1, \ldots, u$ let $P^i = \{p_{a(i)}, p_{a(i)+1}, \ldots, p_{b(i)}\}$ be the set of all $p \in P$ with x-coordinate x^i. Then*

$$\begin{aligned} Z_{\text{ver}} = \quad & \{(p_i, p_{i+1}) \mid x_i = x_{i+1} \text{ and } 1 \le i < n\} \\ & \cup \{(p_{a(i)}, p_{b(i+1)}) \mid y_{a(i)} > y_{b(i+1)} \text{ and } 1 \le i < u\} \\ & \cup \{(p_{b(i)}, p_{a(i+1)}) \mid y_{b(i)} < y_{a(i+1)} \text{ and } 1 \le i < u\}. \end{aligned}$$

In Figure 1 all pairs of Z_{ver} are connected by an edge. Note that Z_{ver} consists of at most $n-1$ point pairs. If no points have the same x-coordinate, then $Z_{\text{ver}} = \{(p_i, p_{i+1}) \mid 1 \le i < n\}$, i.e. Z_{ver} is the set of neighboring pairs in the lexicographical order. The definition of Z_{hor} is analogous to that of Z_{ver} with the roles of x and y exchanged. Figure 2 shows that $Z_{\text{hor}} \cup Z_{\text{ver}}$ is not always a generating set: Since $(p, h) \in Z_{\text{hor}}$ and $(p, v) \in Z_{\text{ver}}$, no network that consists only of Manhattan paths between pairs in $Z_{\text{hor}} \cup Z_{\text{ver}}$ contains a Manhattan p–q path. This shows the necessity of a third subset Z_{quad} of Z.

Definition 2 (Z_{quad}). *For a point $r \in \mathbb{R}^2$ denote its Cartesian coordinates by (x_r, y_r). Let $Q(r,1) = \{s \in \mathbb{R}^2 \mid x_r \le x_s \text{ and } y_r \le y_s\}$ be the first quadrant of the Cartesian coordinate system with origin r. Define $Q(r,2)$, $Q(r,3)$, $Q(r,4)$ analogously and in the usual order. Then Z_{quad} is the set of all ordered pairs $(p,q) \in P \times P$ with $q \in Q(p,t) \setminus \{p\}$ and $t \in \{1,2,3,4\}$ that fulfill*

(a) q is the point that has minimum y-distance from p among all points in $Q(p,t) \cap P$ with minimum x-distance from p, and
(b) there is no $q' \in Q(p,t) \cap P$ with (p,q') or (q',p) in $Z_{\text{hor}} \cup Z_{\text{ver}}$.

Obviously Z_{quad} consists of at most $4n$ point pairs.

For our analysis we need the following regions of the plane. Let $\mathcal{R}_{\text{hor}} = \{\text{BBox}(p,q) \mid \{p,q\} \in Z_{\text{hor}}\}$, where $\text{BBox}(p,q)$ is the smallest axis-parallel closed rectangle that contains p and q. Note that $\text{BBox}(p,q)$ is just the line segment $\text{Seg}[p,q]$ from p to q, if p and q lie on the same horizontal or vertical line. In this case we call $\text{BBox}(p,q)$ a *degenerate rectangle*. Define $\mathcal{R}_{\text{ver}}$ and $\mathcal{R}_{\text{quad}}$ analogously. Let $\mathcal{A}_{\text{hor}}$, $\mathcal{A}_{\text{ver}}$, and $\mathcal{A}_{\text{quad}}$ be the subsets of the plane that are defined by the union of the rectangles in $\mathcal{R}_{\text{hor}}$, $\mathcal{R}_{\text{ver}}$, and $\mathcal{R}_{\text{quad}}$, respectively.

Any Manhattan network has to bridge the vertical (horizontal) gap between the points of each pair in Z_{ver} (Z_{hor}). Of course this can be done such that at the same time the gaps of adjacent pairs are (partly) bridged. The corresponding minimization problem is defined as follows.

Definition 3 (Kato et al. [4]). *A set of vertical line segments $\mathcal{V}$ is a* cover *of (or* covers*) $\mathcal{R}_{\text{ver}}$, if any $R \in \mathcal{R}_{\text{ver}}$ is covered, i.e. for any horizontal line ℓ with $R \cap \ell \neq \emptyset$ there is a $V \in \mathcal{V}$ with $V \cap \ell \cap R \neq \emptyset$. We say that $\mathcal{V}$ is a* minimum vertical cover (MVC) *if $\mathcal{V}$ has minimum length among all covers of $\mathcal{R}_{\text{ver}}$. The definition of a* minimum horizontal cover (MHC) *is analogous.*

For an example, see Figure 3. Since any MMN covers $\mathcal{R}_{\text{ver}}$ and $\mathcal{R}_{\text{hor}}$, we have:

Lemma 1 (Kato et al. [4]). *The union of an* MVC *and an* MHC *has length bounded by the length of an* MMN.

To sketch our algorithm we need the following notation. Let N be a set of line segments. We say that N *satisfies* a set of point pairs S if N contains a Manhattan p–q path for each $\{p, q\} \in S$. We use $\bigcup N$ to denote the corresponding set of points, i.e. the union of the line segments in N. Let ∂M be the boundary of a set $M \subseteq \mathbb{R}^2$.

Our algorithm proceeds in four phases. In phase 0, we compute Z. In phase I, we construct a network N_1 that contains the union of a special MVC and a special MHC and satisfies $Z_{\text{ver}} \cup Z_{\text{hor}}$. In phase II, we identify a set $\mathcal{R}$ of open regions in $\mathcal{A}_{\text{quad}}$ that do not intersect N_1, but need to be bridged in order to satisfy Z_{quad}. The regions in $\mathcal{R}$ are staircase polygons. They give rise to two sets of segments, N_2 and N_3, which are needed to satisfy Z_{quad}. For each region $A \in \mathcal{R}$ we put the segments that form $\partial A \setminus \bigcup N_1$ into N_2, plus, if necessary, an extra segment to connect ∂A to N_1. Finally, in phase III, we bridge the regions in $\mathcal{R}$ by computing a set N_3 of segments in the interior of the regions. This yields a Manhattan network $N = N_1 \cup N_2 \cup N_3$.

The novelty of our analysis is that we partition the plane into two areas and compare N to an MMN in each area separately. The area $\mathcal{A}_3$ consists of the interiors of the regions $A \in \mathcal{R}$ and contains N_3. The other area $\mathcal{A}_{12}$ is the complement of $\mathcal{A}_3$ and contains $N_1 \cup N_2$. For a fixed MMN N_{opt} we show that $|N \cap \mathcal{A}_{12}| \leq 3|N_{\text{opt}} \cap \mathcal{A}_{12}|$ and $|N \cap \mathcal{A}_3| \leq 2|N_{\text{opt}} \cap \mathcal{A}_3|$, and thus $|N| \leq 3|N_{\text{opt}}|$. The details will be given in Section 4.

We now define vertical and horizontal neighbors of points in P. Knowing these neighbors helps to compute Z and $\mathcal{R}$.

Definition 4 (Neighbors). *For a point $p \in P$ and $t \in \{1, 2, 3, 4\}$ let* $p.\text{xnbor}[t] = \text{nil}$ *if $Q(p, t) \cap P = \{p\}$. Otherwise* $p.\text{xnbor}[t]$ *points to the point that has minimum y-distance from p among all points in $Q(p, t) \cap P \setminus \{p\}$ with minimum x-distance from p. The pointer* $p.\text{ynbor}[t]$ *is defined by exchanging x and y in the above definition.*

All pointers of types xnbor and ynbor can be computed by a simple plane sweep in $O(n \log n)$ time. Then we compute Z_{ver} by going through the points in lexicographical order and examining the pointers of type xnbor. This works analogously for Z_{hor}. Note that by Definition 1 each point $q \in P$ is incident to at most three rectangles of $\mathcal{R}_{\text{ver}}$, at most two of which can be (non-) degenerate. We refer to points $p \in P$ with $(p, q) \in Z_{\text{ver}}$ as *vertical predecessors* of q and to points $r \in P$ with $(q, r) \in Z_{\text{ver}}$ as *vertical successors* of q. We call a predecessor or successor of q *degenerate* if it has the same x-coordinate as q. Note that each point can have at most one degenerate vertical predecessor and successor, and at most one non-degenerate vertical predecessor and successor. Horizontal predecessors and successors are defined analogously with respect to Z_{hor}. For each $t \in \{1, 2, 3, 4\}$ the pair $(q, q.\text{xnbor}[t])$ lies in Z_{quad} if and only if $q.\text{xnbor}[t] \neq \text{nil}$ and no vertical or horizontal predecessor or successor of q lies in $Q(q, t)$. Thus:

Lemma 2. *All pointers of type* xnbor, ynbor *and the generating set Z can be computed in $O(n \log n)$ time.*

3 Minimum Covers

In general the union of an MVC and an MHC does not satisfy $Z_{\text{ver}} \cup Z_{\text{hor}}$. Additional segments must be added to achieve this. To ensure that the total length of these segments can be bounded, we need covers with a special property. We say that a cover is *nice* if each cover segment contains an input point.

Lemma 3. *For any nice* MVC $\mathcal{V}$ *and any nice* MHC $\mathcal{H}$ *there is a set* $\mathcal{S}$ *of line segments such that* $\mathcal{V} \cup \mathcal{H} \cup \mathcal{S}$ *satisfies* $Z_{\text{ver}} \cup Z_{\text{hor}}$ *and* $|\mathcal{S}| \leq W + H$*, where* W *and* H *denote width and height of* $\text{BBox}(P)$*, respectively. We can compute the set* $\mathcal{S}$ *in linear time if for each* $R \in \mathcal{R}_{\text{ver}}$ *(*$\mathcal{R}_{\text{hor}}$*) we have constant-time access to the segments in* $\mathcal{V}$ *(*$\mathcal{H}$*) that intersect* R*.*

Proof. We show that there is a set $\mathcal{S}_\mathcal{V}$ of horizontal segments with $|\mathcal{S}_\mathcal{V}| \leq W$ such that $\mathcal{V} \cup \mathcal{S}_\mathcal{V}$ satisfies Z_{ver}. Analogously it can be shown that there is a set $\mathcal{S}_\mathcal{H}$ of vertical segments with $|\mathcal{S}_\mathcal{H}| \leq H$ such that $\mathcal{H} \cup \mathcal{S}_\mathcal{H}$ satisfies Z_{hor}. This proves the lemma.

Let $(p, q) \in Z_{\text{ver}}$. If $R = \text{BBox}(p, q)$ is degenerate, then by the definition of a cover, there is a line segment $s \in \mathcal{V}$ with $R \subseteq s$, and thus $\mathcal{V}$ satisfies (p, q).

Otherwise R defines a non-empty vertical open strip $\sigma(p, q)$ bounded by p and q. Note that by the definition of Z_{ver}, R is the only rectangle in $\mathcal{R}_{\text{ver}}$ that intersects $\sigma(p, q)$. This yields that the widths of $\sigma(p, q)$ over all $(p, q) \in Z_{\text{ver}}$ sum up to at most W. Thus if we can show that there is a horizontal line segment h such that the length of h equals the width of $\sigma(p, q)$ and $\mathcal{V} \cup \{h\}$ satisfies (p, q), then we are done.

Now observe that no line segment in $\mathcal{V}$ intersects $\sigma(p, q)$ since $\mathcal{V}$ is nice and $\sigma(p, q) \cap P = \emptyset$. Hence the segments of $\mathcal{V}$ that intersect R in fact intersect only the vertical edges of R. We assume w.l.o.g. that $x_p < x_q$ and $y_p < y_q$ (otherwise rename and/or mirror P at the x-axis). This means that due to the definition of Z_{ver}, there is no input point vertically above p. Thus if there is a segment s_p in $\mathcal{V}$ that intersects the left edge of R, then s_p must contain p. Analoguously, a segment s_q in $\mathcal{V}$ that intersects the right edge of R must contain q. Since $\mathcal{V}$ covers R, s_p or s_q must exist. Let ℓ be the horizontal through the topmost point of s_p or the bottommost point of s_q. Then $h = \ell \cap R$ does the job, again due to the fact that $\mathcal{V}$ covers R. Clearly h can be determined in constant time. □

In order to see that every point set has in fact a nice MVC, we need the following definitions. We restrict ourselves to the vertical case.

For a horizontal line ℓ consider the graph $G_\ell(V_\ell, E_\ell)$, where V_ℓ is the intersection of ℓ with the vertical edges of rectangles in $\mathcal{R}_{\text{ver}}$, and there is an edge in E_ℓ if two intersection points belong to the same rectangle. We say that a point v in V_ℓ is *odd* if v is contained in a degenerate rectangle or if the number of points to the left of v that belong to the same connected component of G_ℓ is odd, otherwise we say that v is *even*. For a vertical line g let an *odd segment* be an inclusion-maximal connected set of odd points on g. Define *even segments* accordingly. For example, the segment s (drawn bold in Figure 4) is an even segment, while $f \setminus s$ is odd. We say that *parity changes* in points where two segments

of different parity touch. We refer to these points as *points of changing parity.* The MVC with the desired property will simply be the set of all odd segments. The next lemma characterizes odd segments. Strictly speaking we have to state the parity of segment endpoints, but since a closed segment has the same length as the corresponding open segment, we consider odd segments closed.

Lemma 4. *Let $g : x = x_g$ be a vertical line through some $p = (x_p, y_p) \in P$.*

(i) Let e be a vertical edge of a rectangle $R \in \mathcal{R}_{\text{ver}}$. Then either all points on e are even or the only inclusion-maximal connected set of odd points on e contains an input point.

(ii) Let $R_1, \ldots, R_d$ and $R'_1, \ldots, R'_{d'}$ be the degenerate and non-degenerate rectangles in $\mathcal{R}_{\text{ver}}$ that g intersects, respectively. Then $d = |g \cap P| - 1$ and $d' \leq 2$. If $d = 0$ then $d' > 0$ and each R'_i has a corner in p. Else, if $d > 0$, there are $p_1, p_2 \in P$ such that $g \cap (R_1 \cup \cdots \cup R_d) = \text{Seg}[p_1, p_2]$. Then each R'_i has a corner in either p_1 or p_2.

(iii) There are $b_g < t_g \in \mathbb{R}$ such that $g \cap \mathcal{A}_{\text{ver}} = \{x_g\} \times [b_g, t_g]$.

(iv) The line g contains at most two points of changing parity and at most one odd segment. For each point c of changing parity there is an input point with the same y-coordinate.

Proof. For (i) we assume without loss of generality that e is the right vertical edge of $R = \text{BBox}(p, q)$ and that q is the topmost point of e. If R is degenerate it is clear that all points on e (including p and q) are odd, and we are done. Thus we can assume that $x_p < x_q$. Let $p_0 = q, p_1 = p, p_2 \ldots, p_k$ be the input points in order of decreasing x-coordinate that span the rectangles in $\mathcal{R}_{\text{ver}}$ that are relevant for the parity of e. Let $p_i = (x_i, y_i)$. For $2 \leq i \leq k$ define recursively $\overline{y_i} = \min\{y_i, \overline{y_{i-2}}\}$ if i is even, and $\overline{y_i} = \max\{y_i, \overline{y_{i-2}}\}$ if i is odd. Let $\overline{p_i} = (x_i, \overline{y_i})$, and let $\overline{\mathcal{L}}$ be the polygonal chain through $p_0, p_1, \overline{p_2}, \overline{p_3}, \ldots, \overline{p_k}$, see Figure 4. Note that the parity of a point v on e is determined by the number of segments of $\overline{\mathcal{L}}$ that the horizontal h_v through v intersects. If h_v is below $\overline{p_k}$, then it intersects a descending segment for each ascending segment of $\overline{\mathcal{L}}$, hence v is even. If on the other hand h_v is above $\overline{p_k}$, then it intersects an ascending segment for each descending segment—plus $\overline{p_1 p_0}$, hence v is odd. In other words, if $\overline{y_k} = y_0$, all points of e are even, if $\overline{y_k} = y_1$, all points of e are odd, and otherwise parity changes only in $(x_0, \overline{y_k})$ and q is odd. This settles (i).

(ii) follows directly from the definition of Z_{ver}, and (iii) follows from (ii).

For (iv) we first assume $d = 0$. Then (ii) yields $d' \in \{1, 2\}$ and $g \cap P = \{p\}$. By (i) we know that the only inclusion-maximal connected set of odd points on each vertical rectangle edge on g contains an input point, i.e. p. Thus there are at most two points of changing parity and at most one odd segment on g. Also according to the above proof of (ii), parity can change only in points of type $(x_0, \overline{y_k})$, and $\overline{y_k}$ is the y-coordinate of some input point in the set $\{p_0, \ldots, p_k\}$.

Now if $d > 0$ note that all degenerate rectangles consist only of odd points. By (ii) we have that $g \cap (R_1 \cup \cdots \cup R_d) = \text{Seg}[p_1, p_2]$ and that each of the at most two non-degenerate rectangles has a corner in either p_1 or p_2. Thus again the statement holds. □

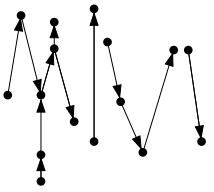


Fig. 1. Point pairs in Z_{ver}

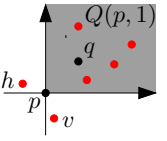


Fig. 2. The pair (p,q) is in Z_{quad}

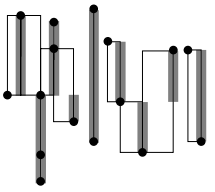


Fig. 3. The odd MVC

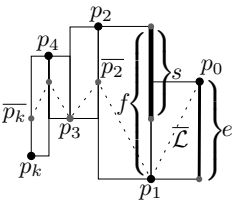


Fig. 4. Proof of Lemma 4

Lemma 5. *The set $\mathcal{V}$ of all odd segments is a nice* MVC*, the* odd MVC.

Proof. Clearly $\mathcal{V}$ covers $\mathcal{R}_{\text{ver}}$. Let ℓ be a horizontal line that intersects $\mathcal{A}_{\text{ver}}$. Consider a connected component C of G_ℓ and let k be the number of vertices in C. If k is even then any cover must contain at least $k/2$ vertices of C, and $\mathcal{V}$ contains exactly $k/2$. On the other hand, if $k > 1$ is odd then any cover must contain at least $(k-1)/2$ vertices of C, and $\mathcal{V}$ contains exactly $(k-1)/2$. If $k = 1$ any cover must contain the vertex, and so does $\mathcal{V}$ since the vertex belongs to a degenerate rectangle. Thus $\mathcal{V}$ is an MVC. Lemma 4 (i) shows that $\mathcal{V}$ is nice. □

Lemma 6. *The odd* MVC *can be computed in $O(n \log n)$ time using linear space.*

To compute the odd MVC we sweep the plane from bottom to top. For each point c of changing parity there is an input point p with $y_c = y_p$. Thus, the event-point queue can be implemented as a sorted list of the y-coordinates of the input points. The sweep-line status is a balanced binary tree in which each node corresponds to a connected components of G_ℓ, where ℓ is the current position of the horizontal sweep line. For details we refer to the full version of this paper [2].

4 An Approximation Algorithm

Our algorithm ApproxMMN proceeds in four phases, see Figure 5.

Phase 0. In phase 0 we compute all pointers of types xnbor and ynbor, and the set Z. From now on we will organize our data structures such that we have constant-time access to all relevant information such as xnbor, ynbor, vertical and horizontal predecessors and successors from each point $p \in P$.

Phase I. First we compute the nice odd MVC and the nice odd MHC, denoted by $\mathcal{C}_{\text{ver}}$ and $\mathcal{C}_{\text{hor}}$, respectively. Then we compute the set $\mathcal{S}$ of additional segments according to Lemma 3. We compute $\mathcal{C}_{\text{ver}}$, $\mathcal{C}_{\text{hor}}$ and $\mathcal{S}$ such that from each point $p \in P$ we have constant-time access to the at most two *cover segments* (i.e. segments in $\mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}}$) that contain p and to the at most four additional segments in rectangles incident to p.

Lemmas 1, 3, and 6 show that $N_1 = \mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}} \cup \mathcal{S}$ can be computed in $O(n \log n)$ time and that $|N_1| \leq |N_{\text{opt}}| + H + W$ holds. Recall that N_{opt} is a fixed MMN.

Phase II. In general N_1 does not satisfy Z_{quad}; further segments are needed. In order to be able to bound the length of these new segments, we partition the plane into two areas $\mathcal{A}_{12}$ and $\mathcal{A}_3$ as indicated in Section 2. We wanted to define $\mathcal{A}_3$ such that $|N_{\text{opt}} \cap \mathcal{A}_3|$ were large enough for us to bound the length of the new segments. However, we were not able to define $\mathcal{A}_3$ such that we could at the same time (a) satisfy Z_{quad} by adding new segments exclusively in $\mathcal{A}_3$ and (b) bound their length. Therefore we put the new segments into two disjoint sets, N_2 and N_3, such that $N_1 \cup N_2 \subseteq \mathcal{A}_{12}$ and $N_3 \subseteq \mathcal{A}_3$. This enabled us to bound $|N_1 \cup N_2|$ by $3|N_{\text{opt}} \cap \mathcal{A}_{12}|$ and $|N_3|$ by $2|N_{\text{opt}} \cap \mathcal{A}_3|$.

We now prepare our definition of $\mathcal{A}_3$. Recall that $Q(q,1), \ldots, Q(q,4)$ are the four quadrants of the Cartesian coordinate system with origin q. Let $P(q,t) = \{p \in P \cap Q(q,t) \mid (p,q) \in Z_{\text{quad}}\}$ for $t = 1,2,3,4$. For example, in Figure 7, $P(q,1) = \{p_1, \ldots, p_5\}$. Due to the definition of Z_{quad} we have $Q(p,t) \cap P(q,t) = \{p\}$ for each $p \in P(q,t)$. Thus the area $\mathcal{A}_{\text{quad}}(q,t) = \bigcup_{p \in P(q,t)} \text{BBox}(p,q)$ is a staircase polygon. The points in $P(q,t)$ are the "stairs" of the polygon and q is the corner opposite the stairs. In Figure 7, $\mathcal{A}_{\text{quad}}(q,1)$ is the union of the shaded areas. In order to arrive at a definition of the area $\mathcal{A}_3$, we will start from polygons of type $\mathcal{A}_{\text{quad}}(q,t)$ and then subtract areas that can contain segments of N_1 or are not needed to satisfy Z_{quad}.

Let $\Delta(q,t) = \text{int}\big(\mathcal{A}_{\text{quad}}(q,t) \setminus (\mathcal{A}_{\text{hor}} \cup \mathcal{A}_{\text{ver}})\big)$, where $\text{int}(M)$ denotes the interior of a set $M \subseteq \mathbb{R}^2$. In Figure 7, $\Delta(q,1)$ is the union of the three areas with dotted boundary. Let $\delta(q,t)$ be the union of those connected components A of $\Delta(q,t)$, such that $\partial A \cap P(q,t) \neq \emptyset$. In Figure 7, $\delta(q,1)$ is the union of the two dark shaded areas A and $\overline{A}$.

Due to the way we derived $\delta(q,t)$ from $\mathcal{A}_{\text{quad}}(q,t)$, it is clear that each connected component A of $\delta(q,t)$ is a staircase polygon, too. The stairs of A correspond to the input points on ∂A, i.e. $P(q,t) \cap \partial A$. Let q_A be the point on ∂A that is closest to q. This is the corner of A opposite the stairs. The next lemma, which is proved in [2], is crucial for estimating the length of our network within the $\delta(q,t)$ regions.

Lemma 7. *Areas of type $\delta(q,t)$ are pairwise disjoint.*

By Lemma 7 we are sure that we can treat each connected component A of $\delta(q,t)$ independently. Finally we define $\mathcal{A}_3 = \bigcup_{t \in \{1,2,3,4\}} \bigcup_{q \in P} \delta(q,t)$ and $\mathcal{A}_{12} = \mathbb{R}^2 \setminus \mathcal{A}_3$. This definition ensures that $N_1 \subset \mathcal{A}_{12}$ as desired. The set N_2 will be constructed as follows: for each connected component A of $\mathcal{A}_3$, we put $\partial A \setminus \bigcup N_1$ into N_2 and test whether N_1 contains a Manhattan path from q_A to q. If not, we add a further segment to N_2. This segment lies in $\mathcal{A}_{\text{hor}}$ and will be defined below. Since $\mathcal{A}_{\text{hor}}$ as well as ∂A are contained in $\mathcal{A}_{12}$, we have $N_2 \subset \mathcal{A}_{12}$. The set N_3 will be defined in phase III and will be arranged such that $N_3 \subset \mathcal{A}_3$.

We now describe how to compute $P(q,t)$ and how to find the connected components of $\delta(q,t)$. We compute all sets $P(q,t)$ by going through the input

points and checking their Z_{quad}-partners. This takes linear time since $|Z_{\text{quad}}| = O(n)$. We sort the points in each set $P(q,t)$ according to their x-distance from q. This takes $O(n \log n)$ total time. The remaining difficulty is to decide which points in $P(q,t)$ are incident to the same connected component of $\delta(q,t)$. In Figure 7, $\{p_1, p_2\} \subset \partial A$ and $\{p_3, p_4, p_5\} \subset \partial\overline{A}$. For our description how to figure this out we assume $t = 1$ and $P(q,1) = (p_1, \ldots, p_m)$. Note that each connected component of $\delta(q,1)$ corresponds to a sequence of consecutive points in $P(q,1)$. By definition, for each connected component A of $\delta(q,1)$ and all $p_i, p_j \in A$ we have $p_i.\text{ynbor}[3] = p_j.\text{ynbor}[3]$.

We detect these sequences by going through $p_1, \ldots, p_m$. Let p_i be the current point and let A be the current connected component. If and only if $p_i.\text{ynbor}[3] \neq p_{i+1}.\text{ynbor}[3]$ there is a rectangle $R_A \in \mathcal{R}_{\text{hor}}$ that separates A from the next connected component of $\delta(q,1)$. The rectangle R_A is defined by the point $v_A = p_i.\text{ynbor}[3]$ and its horizontal successor w_A, which in this case is unique, see Figure 7. It remains to specify the coordinates of the corner point q_A of A. Let p_0 be the (unique) vertical successor of q. Then $x_{q_A} = x_{p_0}$ and $y_{q_A} = y_{w_A}$.

At last, we want to make sure that $N_1 \cup N_2$ contains a Manhattan q–q_A path. The reason for this is that in phase III we will only compute Manhattan paths from each $p_i \in \partial A$ to q_A. Concatenating these paths with the q–q_A path yields Manhattan p_i–q paths since $q_A \in \text{BBox}(q, p_i)$. Note that segments in N_3 lie in $\mathcal{A}_3$ and thus cannot help to establish a q–q_A path within $\text{BBox}(q, q_A) \subset \mathcal{A}_{12}$.

The set N_1 contains a Manhattan q–p_0 path $\mathcal{P}_{\text{ver}}$ and a Manhattan v_A–w_A path $\mathcal{P}_{\text{hor}}$, since $(q, p_0) \in Z_{\text{ver}}$ and $(v_A, w_A) \in Z_{\text{hor}}$. If $q_A \in \mathcal{P}_{\text{ver}}$, then clearly N_1 contains a Manhattan q–q_A path. However, N_1 also contains a Manhattan q–q_A path if $q_A \in \mathcal{P}_{\text{hor}}$. This is due to the fact that $\mathcal{P}_{\text{ver}}$ and $\mathcal{P}_{\text{hor}}$ intersect. If $q_A \notin \mathcal{P}_{\text{ver}} \cup \mathcal{P}_{\text{hor}}$, then $\mathcal{P}_{\text{hor}}$ contains the point $c_A = (x_{q_A}, y_{v_A})$, which lies on the vertical through q_A on the opposite edge of R_A. Thus, to ensure a Manhattan q–q_A path in $N_1 \cup N_2$, it is enough to add the segment $s_A = \text{Seg}[q_A, c_A]$ to N_2. We refer to such segments as *connecting segments*.

The algorithm ApproxMMN does not compute $\mathcal{P}_{\text{ver}}$ and $\mathcal{P}_{\text{hor}}$ explicitly, but simply tests whether $q_A \notin \bigcup N_1$. This is equivalent to $q_A \notin \mathcal{P}_{\text{ver}} \cup \mathcal{P}_{\text{hor}}$ since our covers are minimum and the bounding boxes of $\mathcal{P}_{\text{ver}}$ and $\mathcal{P}_{\text{hor}}$ are the only rectangles in $\mathcal{R}_{\text{ver}} \cup \mathcal{R}_{\text{hor}}$ that contain s_A. Due to the same reason and to the fact that cover edges are always contained in (the union of) edges of rectangles in $\mathcal{R}_{\text{ver}} \cup \mathcal{R}_{\text{hor}}$, we have that $s_A \cap \bigcup N_1 = \{c_A\}$. This shows that connecting segments intersect N_1 at most in endpoints. The same holds for segments in N_2 that lie on $\partial\mathcal{A}_3$. We summarize:

Lemma 8. *In $O(n \log n)$ time we can compute the set N_2, which has the following properties: (i) $N_2 \subset \mathcal{A}_{12}$, (ii) a segment in N_1 and a segment in N_2 intersect at most in their endpoints, and (iii) for each region $\delta(q,t)$ and each connected component A of $\delta(q,t)$, $N_1 \cup N_2$ contains ∂A and a Manhattan q–q_A path.*

Proof. The properties of N_2 follow from the description above. The runtime is as follows. Let A be a connected component of $\mathcal{A}_3$ and $m_A = |P \cap \partial A|$. Note that $\sum m_A = O(n)$ since each point is adjacent to at most four connected components

ApproxMMN(P)

Phase 0: Neighbors and generat. set.
for each $p \in P$ **and** $t \in \{1,2,3,4\}$ **do**
  compute p.xnbor[t] and p.ynbor[t]
compute $Z = Z_{\text{ver}} \cup Z_{\text{hor}} \cup Z_{\text{quad}}$.

Phase I: Compute N_1.
compute odd MVC $\mathcal{C}_{\text{ver}}$ and MHC $\mathcal{C}_{\text{hor}}$
compute set $\mathcal{S}$ of additional segments
$N_1 \leftarrow \mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}} \cup \mathcal{S}$, $N_2 \leftarrow \emptyset$, $N_3 \leftarrow \emptyset$

Phase II: Compute N_2.
compute $\mathcal{A}_3$
for each connected comp. A of $\mathcal{A}_3$ **do**
  $N_2 \leftarrow N_2 \cup (\partial A \setminus \bigcup N_1)$
  if $q_A \notin \bigcup N_1$ **then**
    $N_2 \leftarrow N_2 \cup \{s_A\}$

Phase III: Compute N_3.
for each connected comp. A of $\mathcal{A}_3$ **do**
  $N_3 \leftarrow N_3 \cup$ Bridge(A)

return $N = N_1 \cup N_2 \cup N_3$

Fig. 5.

Bridge$\big(A = (q_A, p_1, \ldots, p_m)\big)$

for $i = 1$ **to** $m-1$ **do**
  compute α_i and β_i
return SubBridge$\big(1, m, 0, 0\big)$

SubBridge$\big(k, l, x_{\text{off}}, y_{\text{off}}\big)$

$A_{\text{curr}} = (q_A + (x_{\text{off}}, y_{\text{off}}), p_k, \ldots, p_l)$
if $l - k < 2$ **return** $\emptyset$
$\Lambda = \big\{j \in \{k, \ldots, l-1\} :$
  $\alpha_j - x_{\text{off}} \le \beta_j - y_{\text{off}}\big\}$
$i = \max \Lambda \cup \{k\}$
if $i < l-1$ **and** $\alpha_i - x_{\text{off}} \le \beta_{i+1} - y_{\text{off}}$
  then $i = i + 1$
$B = \emptyset$
if $i > 1$ **then** $B = B \cup \{a_{i-1} \cap A_{\text{curr}}\}$
if $i < l-1$ **then** $B = B \cup \{b_{i+1} \cap A_{\text{curr}}\}$
$x_{\text{new}} = x(p_{i+1}) - x(q_A)$
$y_{\text{new}} = y(p_i) - y(q_A)$
return $B \cup$
  $\cup$ SubBridge$(l, i-1, x_{\text{off}}, y_{\text{new}})$
  $\cup$ SubBridge$(i+2, l, x_{\text{new}}, y_{\text{off}})$

Fig. 6.

of $\mathcal{A}_3$, according to Lemma 7. After sorting $P(q,t)$ we can compute in $O(m)$ time for each A the segment s_A and the set $\partial A \setminus \bigcup N_1$. This is due to the fact that we have constant-time access to each of the $O(m)$ rectangles in $\mathcal{R}_{\text{hor}} \cup \mathcal{R}_{\text{ver}}$ that intersect ∂A and to the $O(m)$ segments of N_1 that lie in these rectangles. □

Phase III. Now we finally satisfy the pairs in Z_{quad}. Due to Lemma 8 it is enough to compute, for each connected component A of $\mathcal{A}_3$, a set of segments $B(A)$ such that the union of $B(A)$ and ∂A contains Manhattan paths from any input point on ∂A to q_A. We say that such a set $B(A)$ *bridges* A. The set N_3 will be the union over all sets of type $B(A)$. The algorithm Bridge that we use to compute $B(A)$ is similar to the "thickest-first" greedy algorithm for rectangulating staircase polygons, see [3]. However, we cannot use that algorithm since the segments that it computes do not lie entirely in $\mathcal{A}_3$.

For our description of algorithm Bridge, and also in the pseudocode in Figure 6, we assume that A lies in a region of type $\delta(q,1)$. Let again $(p_1, \ldots, p_m)$ denote the sorted sequence of points on ∂A. Note that ∂A already contains Manhattan paths that connect p_1 and p_m to q_A. Thus we are done if $m \le 2$. Otherwise let $p'_j = (x_{p_j}, y_{p_{j+1}})$, $a_j = \text{Seg}[(x_{q_A}, y_{p'_j}), p'_j]$ and $b_j = \text{Seg}[(x_{p'_j}, y_{q_A}), p'_j]$ for $j \in \{1, \ldots, m-1\}$, see Figure 8. We denote $|a_j|$ by α_j and $|b_j|$ by β_j. From now on we identify staircase polygon A with the tupel $(q_A, p_1, \ldots, p_m)$. Let B be the set of segments that algorithm Bridge computes. Initially $B = \emptyset$. The algorithm chooses an $i \in \{1, \ldots, m-1\}$ and adds—if they exist—a_{i-1} and b_{i+1} to B.

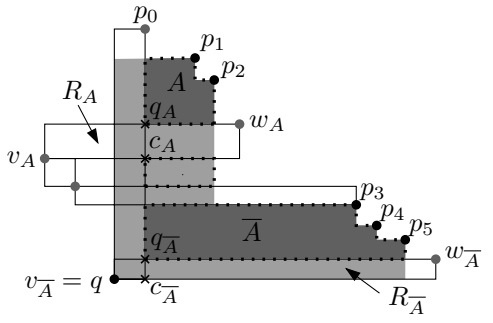


Fig. 7. Notation: $\mathcal{A}_{\text{quad}}(q,1)$ shaded, $\Delta(q,1)$ with dotted boundary, and $\delta(q,1) = A \cup A'$ dark shaded

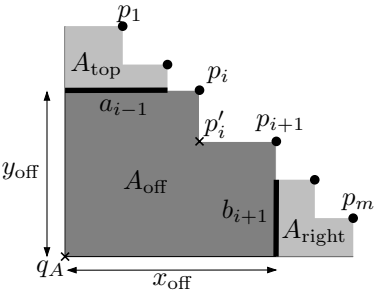


Fig. 8. Notation for algorithm Bridge

This satisfies $\{(p_i,q),(p_{i+1},q)\}$. In order to satisfy $\{(p_2,q),\ldots,(p_{i-1},q)\}$ and $\{(p_{i+2},q),\ldots,(p_{m-1},q)\}$, we solve the problem recursively for the two staircase polygons $\big((x_{q_A},y_{p_i}),p_1,\ldots,p_{i-1}\big)$ and $\big((x_{p_{i+1}},y_{q_A}),p_{i+2},\ldots,p_m\big)$.

Our choice of i is as follows. Note that $\alpha_1 < \cdots < \alpha_{m-1}$ and $\beta_1 > \cdots > \beta_{m-1}$. Let $\Lambda = \{j \in \{1,\ldots,m-1\} \mid \alpha_j \le \beta_j\}$. If $\Lambda = \emptyset$, we have $\alpha_1 > \beta_1$, i.e. A is flat and broad. In this case we choose $i = 1$, which means that only b_2 is put into B. Otherwise let $i' = \max \Lambda$. Now if $i' < m-1$ and $\alpha_{i'} \le \beta_{i'+1}$, we choose $i = i'+1$. In all other cases let $i = i'$. The idea behind this choice of i is that it yields a way to balance α_{i-1} and β_{i+1}, which in turn helps to compare $\alpha_{i-1}+\beta_{i+1}$ to $\min\{\alpha_i,\beta_i,\alpha_{i-1}+\beta_{i+1}\}$, i.e. the length of the segments needed by any Manhattan network to connect p_i and p_{i+1} to q.

To avoid expensive updates of the α- and β-values of the staircase polygons in the recursion, we introduce offset values x_{off} and y_{off} that denote the x- respectively y-distance from the corner of the current staircase polygon to the corner q_A of A. In order to find the index i in a recursion, we compare $\alpha_j - x_{\text{off}}$ to $\beta_j - y_{\text{off}}$ instead of α_j to β_j as in the definition of Λ above.

Running time and performance of algorithm Bridge are as follows:

Theorem 1. *Given a connected component A of $\mathcal{A}_3$ with $|P \cap \partial A| = m$, algorithm* Bridge *computes in $O(m \log m)$ time a set B of line segments with $|B| \le 2|N_{\text{opt}} \cap A|$ and $\bigcup B \subset A$ that bridges A.*

Our proof of Theorem 1 is similar to the analysis of the greedy algorithm for rectangulation, see Theorem 10 in [3]. The details can again be found in [2].

We conclude this section by analyzing the running time of ApproxMMN.

Theorem 2. ApproxMMN *runs in $O(n \log n)$ time and uses $O(n)$ space.*

Proof. Each of the four phases of our algorithm takes $O(n \log n)$ time: for phase 0 refer to Lemma 2, for phase I to Lemmas 3 and 6, for phase II to Lemma 8 and for phase III to Theorem 1. ApproxMMN outputs $O(n)$ line segments. □

5 The Approximation Factor

As desired we can now bound the length of N in $\mathcal{A}_{12}$ and $\mathcal{A}_3$ separately. Theorem 1 and Lemma 7 directly imply that $|N \cap \mathcal{A}_3| = |N_3| \leq 2|N_{\text{opt}} \cap \mathcal{A}_3|$. Note that by $|N_{\text{opt}} \cap \mathcal{A}_3|$ we actually mean $|\{s \cap \mathcal{A}_3 : s \in N_{\text{opt}}\}|$. It remains to show that $|N \cap \mathcal{A}_{12}| = |N_1 \cup N_2|$ is bounded by $3|N_{\text{opt}} \cap \mathcal{A}_{12}|$.

Recall that by Lemmas 1 and 3, $|N_1| \leq |N_{\text{opt}}| + H + W$. Since the segments of N_{opt} that were used to derive the estimation of Lemma 1 lie in $\mathcal{A}_{\text{ver}} \cup \mathcal{A}_{\text{hor}} \subset \mathcal{A}_{12}$, even the stronger bound $|N_1| \leq |N_{\text{opt}} \cap \mathcal{A}_{12}| + H + W$ holds. It remains to analyze the length of N_2 segments. Let N_2^{ver} (N_2^{hor}) denote the set of all vertical (horizontal) segments in N_2. We call segments of N_2 that lie on $\partial\mathcal{A}_3$ *boundary segments*. Due to Lemma 8, segments in N_2^{ver} and segments in $\mathcal{C}_{\text{ver}}$ intersect at most in segment endpoints. Thus, a horizontal line ℓ with $\ell \cap P = \emptyset$ does not contain any point that lies at the same time in $\bigcup \mathcal{C}_{\text{ver}}$ and in $\bigcup N_2^{\text{ver}}$. In the full version [2] we characterize the sequences that are obtained by the intersection of such a line ℓ with cover segments, boundary segments, and connecting segments. A counting argument then yields $\#N_2^{\text{ver}} \leq 2\#\mathcal{C}_{\text{ver}} - 1$ (and analogously $\#N_2^{\text{hor}} \leq 2\#\mathcal{C}_{\text{hor}} - 1$), where $\#N_2^{\text{ver}}$ and $\#\mathcal{C}_{\text{ver}}$ denote the number of segments in N_2^{ver} and $\mathcal{C}_{\text{ver}}$ intersected by ℓ, respectively. By integrating this inequality over all positions of ℓ, we obtain the following lemma.

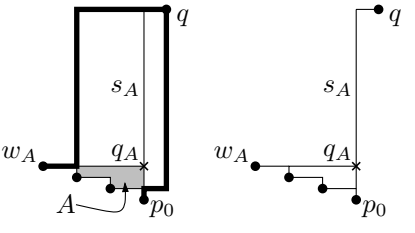


Fig. 9. Left: In the network N_1 neither the w_A–q nor the p_0–q path (thick) contain q_A. Thus $s_A \in N$. Right: MMN of the same point set.

Lemma 9. $|N_2^{\text{ver}}| \leq 2|\mathcal{C}_{\text{ver}}| - H$ *and* $|N_2^{\text{hor}}| \leq 2|\mathcal{C}_{\text{hor}}| - W$.

This finally settles the approximation factor of ApproxMMN.

Theorem 3. $|N| \leq 3|N_{\text{opt}}|$.

Proof. By Lemma 9 and $|\mathcal{C}_{\text{ver}} \cup \mathcal{C}_{\text{hor}}| \leq |N_{\text{opt}} \cap \mathcal{A}_{12}|$ we have $|N_2| \leq 2|N_{\text{opt}} \cap \mathcal{A}_{12}| - H - W$. Together with $|N_1| \leq |N_{\text{opt}}| + H + W$ this yields $|N_1 \cup N_2|/|N_{\text{opt}} \cap \mathcal{A}_{12}| \leq 3$. Theorem 1 and Lemma 7 show that $|N_3|/|N_{\text{opt}} \cap \mathcal{A}_3| \leq 2$. Then, $\mathcal{A}_{12} \cap \mathcal{A}_3 = \emptyset$ yields $|N|/|N_{\text{opt}}| \leq \max\{|N_1 \cup N_2|/|N_{\text{opt}} \cap \mathcal{A}_{12}|, |N_3|/|N_{\text{opt}} \cap \mathcal{A}_3|\} \leq 3$. □

Figure 9 shows that there are point sets for which $|N|/|N_{\text{opt}}|$ can be made arbitrarily close to 3. However, an experimental evaluation of ApproxMMN shows that it behaves much better on point sets under various random distributions. The average performance was around 1.2. Details can be found in [2]. Our algorithm can be tested under the URL `http://i11www.ira.uka.de/manhattan/`. We close with the obvious question: is it NP-hard to compute an MMN?

References

1. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: Short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput. (STOC'95)*, pages 489–498, Las Vegas, 29 May–1 June 1995.
2. M. Benkert, F. Widmann, and A. Wolff. The minimum Manhattan network problem: A fast factor-3 approximation. Technical Report 2004-16, Fakultät für Informatik, Universität Karlsruhe, 2004. Available at `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=/ira/2004/16`.
3. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Approximating a minimum Manhattan network. *Nordic J. Comput.*, 8:219–232, 2001.
4. R. Kato, K. Imai, and T. Asano. An improved algorithm for the minimum Manhattan network problem. In P. Bose and P. Morin, editors, *Proc. 13th Int. Symp. Alg. and Comp. (ISAAC'02)*, vol. 2518 of *LNCS*, pp. 344–356, 2002. Springer-Verlag.
5. F. Lam, M. Alexandersson, and L. Pachter. Picking alignments from (Steiner) trees. *Journal of Computational Biology*, 10:509–520, 2003.

Algorithms for the *d*-Dimensional Rigidity Matroid of Sparse Graphs

Sergey Bereg

Department of Computer Science, University of Texas at Dallas,
Box 830688, Richardson, TX 75083, USA
besp@utdallas.edu

Abstract. Let $\mathcal{R}_d(G)$ be the d-dimensional rigidity matroid for a graph $G = (V, E)$. Combinatorial characterization of generically rigid graphs is known only for the plane $d = 2$ [11]. Recently Jackson and Jordán [5] derived a min-max formula which determines the rank function in $\mathcal{R}_d(G)$ when G is *sparse*, i.e. has maximum degree at most $d + 2$ and minimum degree at most $d + 1$.

We present three efficient algorithms for sparse graphs G that

(i) detect if E is independent in the rigidity matroid for G, and
(ii) construct G using vertex insertions preserving if G is isostatic, and
(iii) compute the rank of $\mathcal{R}_d(G)$.

The algorithms have linear running time assuming that the dimension d is fixed.

1 Introduction

Techniques from Rigidity Theory [4,11] have been recently applied to problems such as collision free robot arm motion planning [1,8], molecular conformations [6,10] and sensor and network topologies [2]. We introduce some notation first, see [4,5,9,11] for more details.

A *framework* (G, p) in d-space is a graph $G = (V, E), n = |V|, m = |E|$ and an embedding $p : V \to \mathbb{R}^d$. Let $p(V) = \{p_1, \ldots, p_n\}$. The *rigidity matrix* of the framework is the $m \times dn$ matrix for the system of m equations

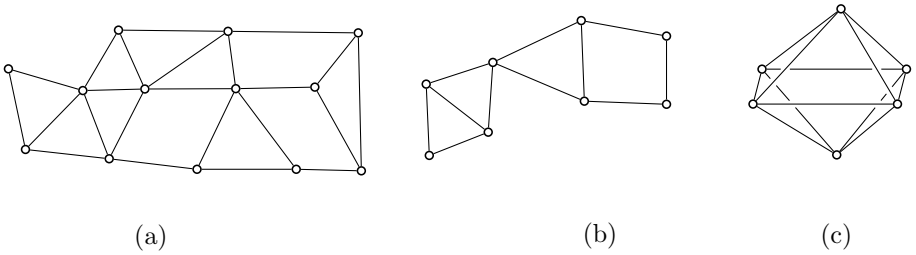
$$(p_i - p_j) \cdot (p'_i - p'_j) = 0, (p_i, p_j) = p(e), e \in E$$

in unknown velocities p'_i. The rigidity matrix of (G, p) defines the *rigidity matroid* of (G, p) on the ground set E by independence of rows of the rigidity matrix. A framework (G, p) is *generic* if the coordinates of the points $p(v), v \in V$ are algebraically independent over the rationals. Any two generic frameworks (G, p) and (G, p') have the same rigidity matroid called d-dimensional *rigidity matroid* $\mathcal{R}_d(G) = (E, r_d)$ of G. The rank of $\mathcal{R}_d(G)$ is denoted by $r_d(G)$.

Lemma 1. [9, Lemma 11.1.3] *For a graph G with n vertices, the rank $r_d(G) \leq S(n, d)$ where*

$$S(n, d) = \begin{cases} nd - \binom{d+1}{2} & \text{if } n \geq d + 1 \\ \binom{n}{2} & \text{if } n \leq d + 1. \end{cases}$$

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 29–36, 2005.



(a) (b) (c)

Fig. 1. (a) Rigid graph in the plane, (b) not rigid graph in the plane, and (c) rigid graph in $\mathbb{R}^3$

We say that a graph $G = (V, E)$ is *rigid* if $r_d(G) = S(n, d)$, see Fig. 1. We say that G is *M-independent*, *M-dependent*, or an *M-circuit* in $\mathbb{R}^d$ if E is independent, dependent, or a circuit, respectively, in $\mathcal{R}_d(G)$. A rigid graph is *minimally rigid* in $\mathbb{R}^d$ (or *generically d-isostatic*) if it is M-independent. The famous Laman Theorem [7] asserts that a graph G with n vertices and m edges is minimally rigid in $\mathbb{R}^2$ if and only if $m = 2n - 3$ and every subgraph induced by k vertices contains at most $2k - 3$ edges for any k.

A combinatorial characterization of rigid graphs is not known for dimensions $d \geq 3$. Recently Jackson and Jordán [5] generalized Laman Theorem to sparse graphs in higher dimensions. Let $G = (V, E)$ be a graph and $d \geq 3$ be a fixed integer. For $X \subseteq V$ let $G[X] = (V(X), E(X))$ be the subgraph of G induced by X. Let $i(X) = |E(X)|$. We say that a graph G is *Laman* if $i(X) \leq S(|X|, d)$ for all $X \subseteq V$. We denote the maximum and minimum degrees of G by $\Delta(G)$ and $\delta(G)$, respectively.

Theorem 1. [5, Theorem 3.5] *Let G be a connected graph with $\Delta(G) \leq d + 2$ and $\delta(G) \leq d + 1$. Then G is M-independent if and only if G is Laman.*

Jackson and Jordán [5] derived a min-max formula for the rank $r_d(G)$ of a sparse graph. A *cover* of G is a collection $\mathcal{X}$ of subsets of V, each of size at lest two, such that $\cup_{X \in \mathcal{X}} E(X) = E$. For $X \subseteq V$ let $f(X) = S(|X|, d)$ and $val(\mathcal{X}) = \sum_{X \in \mathcal{X}} f(X)$. A cover $\mathcal{X}$ is 1-*thin* if $|X \cap X'| \leq 1$ for all distinct $X, X' \in \mathcal{X}$.

Theorem 2. [5, Theorem 3.9] *Let G be a connected graph with $\Delta(G) \leq d + 2$ and $\delta(G) \leq d + 1$. Then $r_d(G) = min_{\mathcal{X}} val(\mathcal{X})$ where the minimum is taken over all 1-thin covers $\mathcal{X}$ of G.*

A direct computation of the rank $r_d(G)$ by Theorem 2 leads to an exponential algorithm since the number of 1-thin covers can be exponential. Thus, it would be interesting to design an efficient algorithm (with polynomial running time) for computing the rank $r_d(G)$.

Isostatic Graphs. By Theorem 60.1.2 [11], a graph $G = (V, E)$ is generically d-isostatic if and only if it is rigid and $|E| = S(|V|, d)$. Inductive constructions are useful for isostatic graphs.

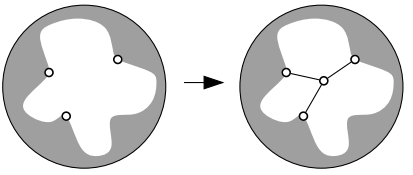


Fig. 2. Vertex addition for $d = 3$

Theorem 3. [11, Theorem 60.1.6] *Vertex Addition.*
Let G be a graph with a vertex v of degree d. Let G' denote the graph obtained by deleting v and the edges incident to it. Then G is generically d-isostatic if and only if G' is generically d-isostatic.

Theorem 4. [11, Theorem 60.1.7] *Edge Split.*
Let G be a graph with a vertex v of degree $d+1$. Let G' denote the graph obtained by deleting v and its $d+1$ incident edges. Then G is generically d-isostatic if and only if there is a pair u, w of vertices of G adjacent to v such that (u, w) is not an edge of G and the graph $G' + (u, w)$ is generically d-isostatic.

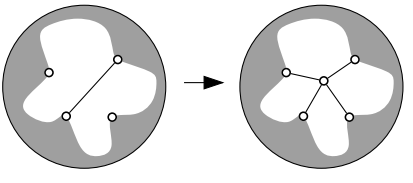


Fig. 3. Edge split for $d = 3$

1.1 Our Results

We present three efficient algorithms.

Theorem 5. *Let G be a graph with $\Delta(G) \leq d + 2$ and $\delta(G) \leq d + 1$. The following problems can be solved in linear time.*

(i) *Determine whether G is M-independent.*
(ii) *If G is generically d-isostatic then compute a sequence of vertex additions and edge splits that yield the graph G.*
(iii) *Compute the rank $r_d(G)$ and a basis of the rigidity matroid $\mathcal{R}_d(G)$.*

2 Detecting M-Independence of a Sparse Graph

Lemma 2. *Let G be a graph with $\Delta(G) \leq d + 2$ and $\delta(G) \leq d + 1$. Let $E' \subseteq E$ be a minimal M-dependent set and let X be the set of endvertices of the edges of E'. Then X contains at most M_d vertices where*

$$M_d = \left\lfloor \frac{(d-1)(d+1)}{d-2} \right\rfloor .$$

Proof. For every vertex $v \in X$, its degree in $G[X]$ is bounded by $d+2$, $d_X(v) \leq d+2$. Therefore

$$2i(X) = \sum_{v \in X} d_X(v) \leq (d+2)|X|.$$

The graph $G[X]$ is connected since E' is a minimal M-dependent set. By Theorem 1, $G[X]$ is not a Laman graph since $G[X]$ is M-dependent. Therefore $i(X) \geq S(|X|, d) + 1$. We assume that $|X| > d+1$ (the lemma follows otherwise). Therefore $S(|X|, d) = d|X| - \binom{d+1}{2}$ and

$$\begin{aligned} 2d|X| - d(d+1) + 2 \leq 2i(X) &\leq (d+2)|X|, \\ (d-2)|X| &\leq (d-1)(d+2), \\ |X| &\leq \frac{(d-1)(d+2)}{d-2}, \\ |X| &\leq M_d. \end{aligned}$$

Algorithm 1.
// Determine whether G is M-independent.
1. For each vertex v of G do
2. Compute $A = \{u \mid d(u,v) < M_d)\}$.
3. For each subset X of A such that $v \in X$ and $|X| \leq M_d$ and $G[X]$ is connected
4. Compute $i(X)$.
5. If $i(X) > S(|X|, d)$ then return "G is M-dependent"
6. return "G is M-independent"

□

Theorem 6. *Let G be a graph with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$. The above algorithm detects in linear time whether G is M-independent or not, in $O(n)$ time assuming that d is fixed.*

Proof. The algorithm checks all subsets of V of size at most M_d that induce connected graphs. By Lemma 2 the graph G is M-dependent if and only if at least one of these sets induces the connected and M-dependent graph. By Theorem 1 it is necessary and sufficient to test if the induced graph is Laman.

We analyze the running time in terms of both n and d, and show later that the dependence on n is linear. The degree of each vertex is bounded by $d+2$. Therefore the size of A is at most

$$|A| \leq 1 + (d+2) + (d+2)^2 + \cdots + (d+2)^{M_d - 1} = \frac{(d+2)^{M_d} - 1}{d+1}.$$

Let A_d be the number of the subsets of A of size at most M_d. Then

$$A_d = \binom{|A|}{1} + \binom{|A|}{2} + \cdots + \binom{|A|}{M_d} \leq |A|^{M_d}.$$

The running time is $O(A_d(d+2)^{M_d - 1}n)$ since we need $O((d+2)^{M_d-1})$ time to compute $i(X)$ for each subset. The theorem follows since d is a constant. □

3 Isostatic and M-Independent Graphs

A set $X \subseteq V$ is *critical* if $|X| \geq 2$ and $i(X) = S(|X|, d)$.

We show that M-independent graph with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$ can be constructed using (i) the operations of vertex addition and edge split as in Theorems 3 and 4, and (ii) addition of a vertex of degree less than d. We need the following bound on the size of a critical set.

Lemma 3. *Let G be a graph with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$. Any critical set in G contains at most N_d vertices where*

$$N_d = \left\lfloor \frac{d(d+1)}{d-2} \right\rfloor.$$

Proof. Let X be a critical set in G. For every vertex $v \in X$, its degree in $G[X]$ is bounded by $d+2$, $d_X(v) \leq d+2$. Therefore

$$2i(X) = \sum_{v \in X} d_X(v) \leq (d+2)|X|.$$

On the other hand, $i(X) = S(|X|, d)$ since X is critical. We assume that $|X| > d+1$ (the lemma follows otherwise). Therefore $S(|X|, d) = |X|d - \binom{d+1}{2}$ and

$$\begin{aligned} 2i(X) = 2d|X| - d(d+1) &\leq (d+2)|X|, \\ (d-2)|X| &\leq d(d+1), \\ |X| &\leq \frac{d(d+1)}{d-2}, \\ |X| &\leq N_d. \end{aligned}$$

Algorithm 2.
// Given a M-independent graph G with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$,
// find a sequence of vertex additions and edge splits that creates G.
1. Partition V into sets V_d, V_{d+1} and V_{d+2} of vertices of degree $\leq d, d$ and $d+2$, respectively.
2. **while** $E \neq \emptyset$ **do**
3. **if** $V_d \neq \emptyset$ **then**
4. Remove a vertex v from V_d.
5. Update E, V_d, V_{d+1} and V_{d+2}.
6. **else**
7. Let v be a vertex of V_{d+1}.
8. Compute $N(v) = \{u \mid (u, v) \in E\}$.
9. Compute $A = \{u \mid d(u, v) \leq N_d)\}$.
10. Compute the set $\mathcal{C}$ of all maximal critical sets $C \subseteq A$.
11. **for each** $u \in N(v)$
12. Find $C(u) \in \mathcal{C}$ such that $u \in C(u)$; if $C(u)$ does not exist then $C(u) = \{u\}$.
13. Find a pair $u, w \in N(v)$ such that $(u, w) \notin E$ and $C(u) \neq C(w)$.
14. Remove v from G and add the edge (u, w) to E.
15. Update E, V_d, V_{d+1} and V_{d+2}.

□

Theorem 7. *Let G be a M-independent graph in $\mathbb{R}^d$ with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$. The above algorithm computes in linear time a sequence of additions of vertices of degree at most $d+1$ and edge splits that yields the graph G.*

Proof. First, we prove the correctness of the algorithm. There are two updates of G in the algorithm: the removal of vertex v in the line 4 and the removal of vertex v with the insertion of edge (u, w) in the line 14. The degree of a vertex $u \neq v$ does not increase after either update. Therefore the graph G preserves the property $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$ after its modification.

The graph G remains M-independent after the deletion in the line 4 since the degree of v is at most d. We show that the update of G in the line 14 preserves M-independence of G. Arguing by contradiction we suppose that G is M-dependent after the update. Then there exists $E' \subseteq E$ that is dependent in $\mathcal{R}_d(G)$. Let V' denote the set of the vertices incident to an edge of E'. The graph (V', E') is M-circuit since $E' - \{(u, w)\}$ is independent. Therefore the graph $G' = (V', E' - \{(u, w)\})$ is critical. This contradicts the choice of (u, w).

The existence of the edge (u, w) (the line 13) follows from Lemma 4. The algorithm finds all critical sets in A since the size of a critical set is bounded by N_d by Lemma 3.

For analysis of the running of the algorithm time we assume that $d = O(1)$. The running time is linear since (i) $|E| = O(n)$, and (ii) $|N(v)| = O(1), |A| = O(1), |\mathcal{C}| = O(1)$, and (iii) the sets E, V_d, V_{d+1} and V_{d+2} can be updated in $O(1)$ time after each modification of G. □

Corollary 1. *Let G be a d-isostatic graph in $\mathbb{R}^d$ with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$. The above algorithm computes in linear time a sequence of additions of vertices of degree at most $d+1$ and edge splits that yields the graph G.*

Proof. The graph G has no vertices of degree less than d. By Theorems 3 and 4 the graph after removal of a vertex of degree d (the line 4) or degree $d+1$ (the line 14) is isostatic. Therefore the new graph does not contain a vertex of degree less than d. The corollary follows. □

Lemma 4. [5, Corollary 3.8] *Let G be a connected M-independent graph with $\Delta(G) = d+2$ and $\delta(G) = d+1$. Let X_1, X_2 be maximal critical subsets of V and suppose that $|X_i| \geq d+2$ for each $i \in \{1, 2\}$. Then $X_1 \cap X_2 = \emptyset$.*

4 Basis of the Rigidity Matroid

An independent set all of whose proper supersets are dependent is called a *basis*. We say that a set of vertices $X \subseteq V$ is *dependent* if the graph induced by X is M-dependent.

The algorithm for finding a basis of G maintains a graph $G' = (V, E')$ by inserting edges of G that are independent in G'. For a set $X \subset V$, we denote by $i'(X)$ the number of edges in the graph $G'[X]$ induced by X.

Algorithm 3.
// Given a graph G with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$,
// compute the rank r of $r_d(G)$ and a basis B of the rigidity matroid $\mathcal{R}_d(G)$.
1. Initialize $r = 0$ and $B = \emptyset$ and $G' = (V, \emptyset)$.
2. **for each** edge (u, v) of G **do**
3. flag=TRUE //boolean flag indicates whether (u, v) is independent
4. Compute $A = \{w \mid d(u, w) < N_d \text{ and } d(v, w) < N_d\}$.
5. **for each** subset X of A such that $u, v \in X$ and $|X| \leq N_d$ and X is connected
6. Compute $i'(X)$.
7. **if** $i'(X) = S(|X|, d)$ **then**
8. print "(u, v) is dependent" and set flag=FALSE
9. **if** flag **then**
10. Add (u, v) to G'.
11. $r = r + 1$ and $B = B \cup \{(u, v)\}$
12. **return** r and B

Theorem 8. *Let G be a graph with $\Delta(G) \leq d+2$ and $\delta(G) \leq d+1$. The above algorithm computes the rank $r_d(G)$ and a basis of the rigidity matroid $\mathcal{R}_d(G)$ in linear time.*

Proof. The graph G' has the property that $\Delta(G') \leq d+2$ and $\delta(G') \leq d+1$. By Theorem 1 and Lemma 3 the edges rejected for insertion to G' are dependent and the set B is M-independent. Therefore B is the basis of G and the rank is computed correctly.

The running time follows since $|E| = O(n)$ and the number of subsets of A is $O(1)$. □

5 Conclusion

We presented three efficient algorithms for sparse graphs for (i) detecting M-independent graphs, and (ii) constructing M-independent graphs, and (iii) computing the rank of a graph. All algorithms have linear running time assuming that d is fixed. The hidden constants are exponential in d. In the journal version we show that the algorithms can be improved so that the dependence of the running time on d is polynomial.

References

1. R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. In *Proc. 41th Annu. Sympos. on Found. of Computer Science*, pp. 432–442, 2000.
2. T. Eren, B. D. Anderson, W. Whiteley, A. S. Morse, and P. N. Belhumeur. Information structures to control formation splitting and merging. In *Proc. of the American Control Conference*, 2004. to appear.

3. T. Eren, W. Whiteley, A. S. Morse, P. N. Belhumeur, and B. D. Anderson. Sensor and network topologies of formations with direction, bearing and angle information between agents. In *Proc. of the 42nd IEEE Conference on Decision and Control*, pp. 3064–3069, 2003.
4. J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity*, volume 2. Amer. Math. Soc., Graduate Studies in Mathematics, 1993.
5. B. Jackson and T. Jordán. The d-dimensional rigidity matroid of sparse graphs. Technical Report TR-2003-06, EGRES Technical Report Series, 2003.
6. D. Jacobs, A. J. Rader, L. Kuhn, and M. Thorpe. Protein flexibility predictions using graph theory. *Proteins*, 44:150–165, 2001.
7. G. Laman. On graphs and rigidity of plane skeletal structures. *J. Engineering Math.*, 4:331–340, 1970.
8. I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proc. 41st Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 443–453, 2000.
9. W. Whiteley. Some matroids from discrete applied geometry. In J. E. Bonin, J. G. Oxley, and B. Servatius, editors, *Contemp. Mathematics*, 197, pp. 171–311. Amer. Math. Soc., Seattle, WA, 1997.
10. W. Whiteley. Rigidity of molecular structures: generic and geometric analysis. In M. F. Thorpe and P. M. Duxbury, editors, *Rigidity Theory and Applications*, pp. 21–46. Kluwer, 1999.
11. W. Whiteley. Rigidity and scene analysis. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 60, pp. 1327–1354. CRC Press LLC, Boca Raton, FL, 2004.

Sliding Disks in the Plane

Sergey Bereg[1], Adrian Dumitrescu[2], and János Pach[3,⋆]

[1] Computer Science, University of Texas at Dallas,
P.O. Box 830688, Richardson, TX 75083, USA
besp@utdallas.edu
[2] Computer Science, University of Wisconsin–Milwaukee,
3200 N. Cramer Street, Milwaukee, WI 53211, USA
ad@cs.uwm.edu
[3] Courant Institute of Mathematical Sciences,
251 Mercer Street, New York, NY 10012-1185, USA
pach@cims.nyu.edu

Abstract. Given a pair of start and target configurations, each consisting of n pairwise disjoint disks in the plane, what is the minimum number of moves that suffice for transforming the start configuration into the target configuration? In one move a disk slides in the plane without intersecting any other disk, so that its center moves along an arbitrary (open) continuous curve. We discuss efficient algorithms for this task and estimate their number of moves under different assumptions on disk radii and disk placements. For example, with n congruent disks, $\frac{3n}{2} + O(\sqrt{n \log n})$ moves always suffice for transforming the start configuration into the target configuration; on the other hand, $\left(1 + \frac{1}{15}\right) n - O(\sqrt{n})$ moves are sometimes necessary.

1 Introduction

Consider a set (system) of n pairwise disjoint objects in the plane that need to be brought from a given start (initial) configuration S into a desired goal (target) configuration T. The *motion planning* problem for such a system is that of computing a sequence of object motions (schedule) that achieves this task. Depending on the existence of such a sequence of motions, we say that the problem is *feasible* or respectively, *infeasible*. Here we restrict ourselves to systems of disks with pairwise disjoint interiors, as objects, and moves that slide a disk without intersecting any other disk throughout the motion. The disks are not labeled, therefore if there exist congruent disks in the system, occupying any of the target positions with a congruent disk is allowed.

It is easy to see that, for the class of disks, the problem is always feasible. More generally, it is also feasible for the class of all convex objects, using sliding moves (Theorem 1 below). This old result appears in the work of Fejes Tóth and Heppes [8], but it can be traced back to de Bruijn [5]; the algorithmic aspects

⋆ Supported by NSF grant CCR-00-98246, by an NSA grant, a PSC-CUNY Research Award, and grant OTKA-T-032-452 from the Hungarian Science Foundation.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 37–47, 2005.

of the problem have been studied by Guibas and Yao [9]. We refer to this set of motion rules (moves) as the *sliding model*. Other reconfiguration rules (models) for systems of disks have been examined recently, for example: in [6] moves are restricted so that a disk can only be placed in a position where it is adjacent to at least two other disks; in [1] moves are translations along a fixed direction at each step. Reconfiguration for modular systems acting in a grid-like environment, and where moves must maintain connectivity of the whole system has been recently addressed in [7].

Theorem 1. *Any set of n convex objects in the plane can be separated via translations all parallel to any given fixed direction, with each object moving once only. If the topmost and bottommost points of each object are given (or can be computed in $O(n \log n)$ time), an ordering of the moves can be computed in $O(n \log n)$ time.*

The following simple *universal* algorithm that can be adapted to any set of n convex objects performs $2n$ moves for reconfiguration of n disks. In the first step (n moves), in decreasing order of the x-coordinates of their centers, slide the disks initially along a horizontal direction, one by one to the far right. Note that no collisions can occur. In the second step (n moves), bring the disks "back" to target positions in increasing order of the x-coordinates of their centers. (General convex objects need rotations and translations in the second step). Already for the class of disks, one cannot do much better in terms of the number of moves (see Theorem 3). For the class of segments (as objects), it is easy to construct examples that require $2n-1$ moves for reconfiguration, even for congruent segments.

A move is a *target move* if it slides a disk to a final target position. Otherwise, it is a *non-target* move. Our lower bounds use the the following argument: if no target disk coincides with a start disk (so each disk must move), a schedule with x non-target moves consists of at least $n + x$ moves.

Our paper is organized as follows. In Section 2 (Theorem 2), we estimate the number of necessary moves for the reconfiguration of systems of congruent disks. In Section 3 (Theorem 3), we estimate the number of necessary moves for the reconfiguration of systems of disks of arbitrary radii.

2 Congruent Disks

We now consider reconfiguring sets of congruent disks in the plane. First, we prove the existence of a line bisecting the set of centers of the start disks such that the strip of width 6 around this line contains a small number of disks. A slightly weaker statement guaranteeing the existence of a bisecting line that cuts through few disks was given by by Alon *et. al* [2]. We have included our almost identical proof for completeness.

Lemma 1. *Let S be a set of n pairwise disjoint unit (radius) disks in the plane. Then there exists a line ℓ that bisects the centers of the disks such that the parallel*

strip of width 6 *around* ℓ *(that is,* ℓ *runs in the middle of this strip) contains entirely at most* $O(\sqrt{n \log n})$ *disks.*

Proof. Set $m = c_2\sqrt{n \log n}$ where $c_2 > 0$ is a suitable large constant to be chosen later. Assume for contradiction that the strip of width $w = 6$ around each line bisecting the set of centers of S contains at least m disks. Set $k = \lceil \sqrt{n/\log n} \rceil$ and consider the k bisecting lines that form angles $i\theta$ with the positive direction of the x-axis (in counterclockwise order), where $i = 0, \ldots, k-1$, and $\theta = \pi/k$.

Let A_i be the set of disks contained (entirely) in the i-strip of width $w = 6$ around the ith bisecting line, $i = 0, \ldots, k-1$. Clearly

$$n \geq |A_0 \cup \ldots A_{k-1}| \geq \sum_{i=0}^{k-1} |A_i| - \sum_{0 \leq i < j \leq k-1} |A_i \cap A_j| \tag{1}$$

by the inclusion-exclusion formula. By our assumption $\sum_{i=0}^{k-1} |A_i| \geq km$. The summand $|A_i \cap A_j|$ counts the number of disks contained in the intersection of the strips i and j. This intersection is a rhombus whose area is

$$F_{ij} = \frac{w^2}{\sin(j-i)\theta}.$$

Since the disks are pairwise disjoint,

$$|A_i \cap A_j| \leq \frac{F_{ij}}{\pi}.$$

We thus have

$$\sum_{0 \leq i < j \leq k-1} |A_i \cap A_j| = O\left(\sum_{0 \leq i < j \leq k-1} \frac{1}{\sin(j-i)\theta} \right).$$

The identity $\sin\alpha = \sin(\pi - \alpha)$ yields

$$\sum_{0 \leq i < j \leq k-1} \frac{1}{\sin(j-i)\theta} \leq k \sum_{i=1}^{\lfloor k/2 \rfloor} \frac{1}{\sin i\theta}.$$

For $1 \leq i \leq k/2$

$$\frac{1}{\sin i\theta} = \frac{1}{\sin \frac{i\pi}{k}} = O\left(\frac{k}{i}\right).$$

Consequently the second sum in Equation (1) is bounded as follows:

$$\sum_{0 \leq i < j \leq k-1} |A_i \cap A_j| = O\left(k^2 \sum_{i=1}^{\lfloor k/2 \rfloor} \frac{1}{i} \right) = O(k^2 \log k).$$

Let $c_1 > 0$ be an absolute constant such that $\sum_{0 \le i < j \le k-1} |A_i \cap A_j| \le c_1 \cdot k^2 \log k$. Since $\log k \le (\log n)/2$ for $n \ge 16$, and using the above estimates, Equation (1) can be rewritten as

$$n \ge mk - c_1 \cdot k^2 \log k \ge c_2 \sqrt{n \log n} \sqrt{\frac{n}{\log n}} - 2c_1 \frac{n}{\log n} \frac{\log n}{2} = (c_2 - c_1)n.$$

Take now $c_2 = c_1 + 2$, and obtain $n \ge 2n$ which is a contradiction. □

Theorem 2. *Given a pair of start and target configurations S and T, consisting of n congruent disks each, $\frac{3n}{2} + O(\sqrt{n \log n})$ moves always suffice for transforming the start configuration into the target configuration. The entire motion can be computed in $O(n^{3/2}(\log n)^{-1/2})$ time. On the other hand, there exist pairs of configurations that require $\left(1 + \frac{1}{15}\right) n - O(\sqrt{n})$ moves for this task.*

Proof. We start with the upper bound. Let S' and T' be the centers of the start disks and target disks, respectively, and let ℓ be the line guaranteed by Lemma 1. Without loss of generality we can assume that ℓ is vertical. Denote by $s_1 = \lfloor n/2 \rfloor$ and $s_2 = \lceil n/2 \rceil$ the number of centers of start disks to the left and to the right of ℓ. Let $m = O(\sqrt{n \log n})$ be the number of start disks contained in the vertical strip around ℓ. Denote by t_1 and t_2 the number of centers of target disks to the left and to the right of ℓ, respectively. By symmetry we can assume that $t_1 \le n/2 \le t_2$.

Let R be a region containing all start and target disks (e.g., the smallest axis-aligned rectangle that contains all disks). The algorithm has three steps. All moves in the region R are taken along horizontal lines, i.e., perpendicularly to the line ℓ.

Step 1. Slide to the far right all start disks whose centers are to the right of ℓ and the (other) start disks in the strip, one by one, in decreasing order of their x-coordinates (with ties broken arbitrarily). At this point all $t_2 \ge n/2$ target disks whose centers are right of ℓ are free.

Step 2. Using all the $s_1' \le n/2$ remaining disks whose centers are to the left of ℓ, in increasing order of their x-coordinates, we fill free target positions to the right of ℓ, in increasing order of their x-coordinates: each disk slides first to the left, then to the right on a wide arc and to the left again in the end. Note that $s_1' \le n/2 \le t_2$. Now all the target positions whose centers are to the left of ℓ are free.

Step 3. Move to place the far away disks: first continue to fill target positions whose centers are to the right of ℓ, in increasing order of their x-coordinates. When we are done, we fill target positions whose centers are to the left of ℓ, in decreasing order of their x-coordinates. Note that at this point all target positions to the left of ℓ are "free."

The only non-target moves are those done in Step 1 and their number is $n/2 + O(\sqrt{n \log n})$, so the total number of moves is $3n/2 + O(\sqrt{n \log n})$.

Algorithm. A trivial implementation of the algorithm examines all $k = \lceil \sqrt{n/\log n} \rceil$ strip directions each in $O(n)$ time, in order to find a suitable one, as described in the proof of Lemma 1. After that, $O(n \log n)$ time is

spent for this direction for sorting and performing the moves. The resulting time complexity is $O(n^{3/2}(\log n)^{-1/2})$.

Lower Bound. The target configuration consists of a set of n densely packed unit (radius) disks contained, for example, in a square of side length $\approx 2\sqrt{n}$. The disks in the start configuration enclose the target positions in a ring-like structure with long "legs." Its design is more complicated and uses "rigidity" considerations as described below.

A packing $\mathcal{C}$ of unit (radius) disks in the plane is said to be *stable* if each disk is kept fixed by its neighbors [4]. More precisely, $\mathcal{C}$ is stable if none of its elements can be translated by any small distance in any direction without colliding with the others. It is easy to see that any stable system of (unit) disks in the plane has infinitely many elements. K. Böröczky [3] showed that there exist stable systems of unit disks with arbitrarily small density.

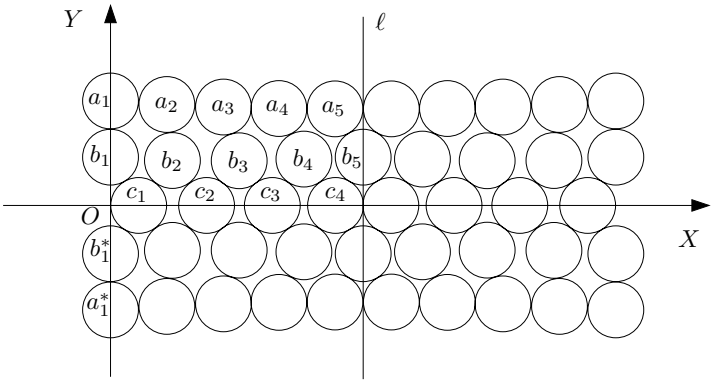


Fig. 1. A double bridge and its vertical line of symmetry ℓ. The part left of ℓ forms the initial section of a one-way infinite bridge.

The main building block used in Böröczky's construction was a one-way infinite "bridge" made up of disks, which can be defined as follows. In Fig. 1, the initial section of such a one-way infinite bridge appears on the left of the vertical line ℓ. Fix an x-y rectilinear coordinate system in the plane. Let us start with five unit disks centered at

$$a_1 = (0, 2+\sqrt{3}), b_1 = (0, \sqrt{3}), c_1 = (1, 0), b_1^* = -b_1, \quad a_1^* = -a_1,$$

that serve as an "abutment." The bridge will be symmetric about the x-axis, so it is sufficient to describe the part of the packing in the upper half-plane. The set of centers of the disks is denoted by C.

Take a strictly convex function $f(x)$ defined for all $x \geq 0$ such that $f(0) = 2+\sqrt{3}$ and $\lim_{x\to\infty} f(x) = 2\sqrt{3}$. Starting with a_1, choose a series of points $a_2, a_3, a_4, \ldots$ belonging to the graph of f such that the distance between any two consecutive points satisfies

$$|a_i - a_{i+1}| = 2 \quad (i = 1, 2, 3, \ldots).$$

All unit disks around these points belong to the packing, so that $a_i \in C$ for every i. These points will uniquely determine all other elements of C, according to the following rules.

Let b_2 be the point at distance 2 from both c_1 and a_2, which lies to the right of the line c_1a_2. Once b_2 is defined, let c_2 be the point on the x-axis, different from c_1, whose distance from b_2 is 2. In general, if b_i and c_i have already been defined, let b_{i+1} denote the point at distance 2 from both c_i and a_{i+1}, lying on the right-hand side of their connecting line, and let $c_{i+1} \neq c_i$ be the (other) point of the x-axis at distance 2 from b_{i+1}. Let C, the set of centers of the disks forming the bridge, consist of all points a_i, b_i, c_i $(i = 1, 2, 3, \ldots)$ and their reflections about the x-axis. Note that the points $c_i \in C$ lie on the x-axis, so they are identical with their reflections.

We need four properties of this construction, whose simple trigonometric proofs can be found in [3]:

1. the distance between any two points in C is at least 2;
2. all unit disks around a_i, b_i, c_i $(i = 2, 3, 4, \ldots)$ are kept fixed by their neighbors;
3. all points $b_2, b_3, b_4, \ldots$ lie strictly below the line $y = \sqrt{3}$;
4. the x-coordinate of c_i is smaller than that of a_{i+1} $(i = 1, 2, 3, \ldots)$.

It is not hard to see that the difference between the x-coordinates of c_i and a_{i+1} tends to zero as i tends to infinity.

Next, we slightly modify the above construction. Take a small positive ε and replace $f(x)$ by the strictly convex function

$$f_\varepsilon(x) := (1 + \varepsilon)f(x) - \varepsilon f(0)$$

whose asymptote is the line $y = 2\sqrt{3} - (2 - \sqrt{3})\varepsilon$. Clearly, $f_\varepsilon(0) = f(0)$. If we carry out the same construction as above, nothing changes before we first find a point a_i that lies below the line $y = 2\sqrt{3}$. However, if ε is sufficiently large, sooner or later we get stuck: the construction cannot be continued forever without violating any of the conditions listed above. Let k be the first integer for which such an event occurs, involving a_k, a_{k+1}, b_k, or c_k. By varying $\varepsilon > 0$, it can be shown by a simple case analysis that the construction can be realized up to level k so that the difference between the x-coordinates of b_k and a_k is 1. It follows that the disk around a_k is tangent to the vertical line ℓ passing through b_k. Remove the rightmost disk centered at c_k from the set. Thus from the above condition, by taking the union of the part of C built so far together with its reflection about ℓ, we obtain the following:

Lemma 2. *There exist arbitrarily long finite packings ("double-bridges") consisting of five rows of unit disks, symmetric about the coordinate axes, in which all but eight disks are kept fixed by their neighbors. These eight exceptional disks are at the two abutments of the double-bridge and their y-coordinates are* $\pm\sqrt{3}, \pm(2 + \sqrt{3})$.

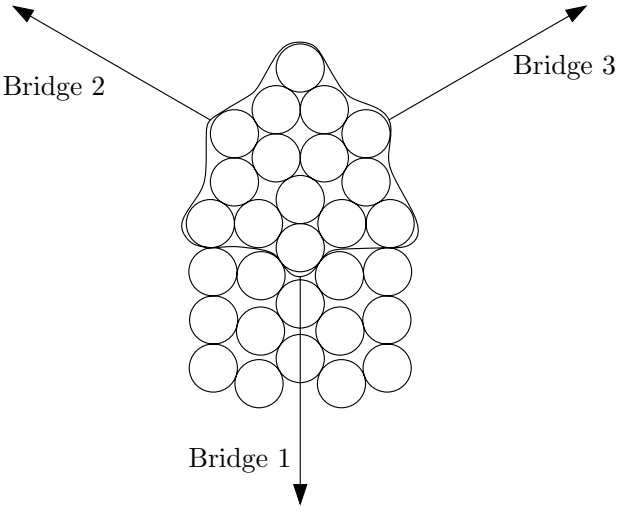


Fig. 2. Junction of type 1

Notice that three such bridges can be connected at a "junction" depicted in Fig. 2 so that the angles between their "long" half-axes of symmetries (corresponding to the positive x-axis) are $\frac{2\pi}{3}$. Consequently, using six double-bridges connected by six junctions one can enclose an arbitrarily large hexagonal region H. Let us attach a one-way infinite bridge to each of the unused sides of the junctions. As Böröczky pointed out, the resulting packing is stable.

Let us refer to the disks in the start (resp. target) configuration as white (resp. black) disks. Now fix a large n, and take n white disks. Use $O(\sqrt{n})$ of them to build six junctions connected by six double-bridges (as described above) to enclose a hexagonal region that can accommodate the n nonoverlapping black disks. See also Fig. 3. Divide the remaining white disks into six roughly equal groups, each of size $\frac{n}{6} - O(\sqrt{n})$, and rearrange each group to form the initial section of a one-way infinite bridge attached to the unused sides ("ports") of the junctions. Notice that the number of necessary moves is at least $\left(1 + \frac{1}{30}\right) n - O(\sqrt{n})$. To see this, it is enough to observe, that in order to fill the first target, we have to break up the hexagonal ring around the black disks. That is, we have to move at least one element of the six double-bridges enclosing H. However, with the exception of the at most $6 \times 5 = 30$ white disks at the far ends of the truncated one-way infinite bridges, every white disk is fixed by its neighbours. Each of these bridges consists of five rows of disks of "length" roughly $\frac{n}{30}$, where the length of a bridge is the number of disks along its side. Therefore, before we could move any element of the ring around H, we must start at a far end and move a sequence of roughly $\frac{n}{30}$ white adjacent disks.

Instead of enclosing the n black disks by a hexagon, we can construct a triangular ring T around them, consisting of three double-bridges (see Fig. 3). To achieve this, we have to build a junction of three sides establishing a connection between the abutments of three bridges such that the angles between their half-

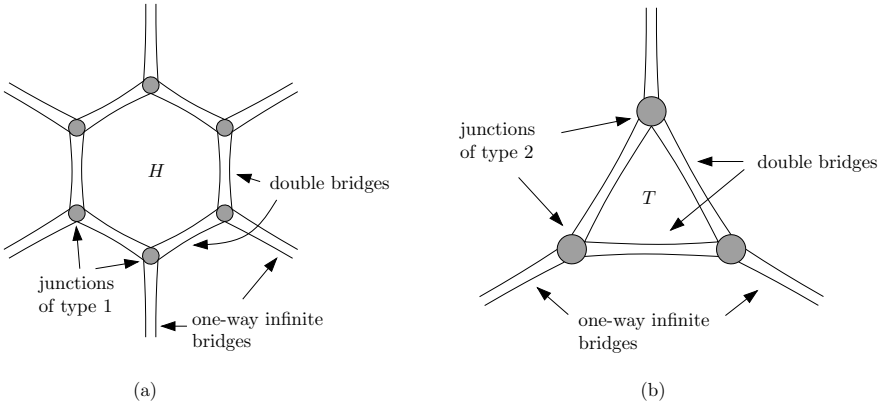


Fig. 3. Two start configurations based on hexagonal and triangular rings

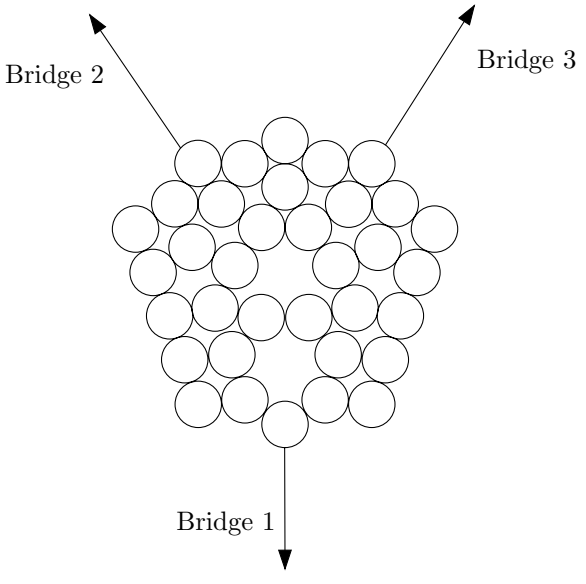


Fig. 4. Junction of type 2

axes of symmetry are $\frac{5\pi}{6}$, $\frac{5\pi}{6}$, and $\frac{\pi}{3}$. Such a junction is shown on Fig. 4. The convex hull of the disk centers (for the disks in the junction) is a pentagon symmetric with respect to a vertical line passing through the top vertex. Four out of the five centers along each of the three sides of the pentagon connected to bridges are collinear. The disk centers on the other two sides form two slightly concave chains. The number of necessary moves is at least $\left(1 + \frac{1}{15}\right) n - O(\sqrt{n})$ for this second construction. This completes the proof of Theorem 2. □

Remarks. We believe that our lower bound in Theorem 2 is closer to the truth. Closing the gap between the bounds remains an interesting problem which seems to require new ideas.

Note that moving out in STEP 1 only start disks whose centers are right of ℓ and those disks intersecting ℓ would not necessarily free all targets whose centers are right of ℓ. This is the reason for working with a strip of width 6 around ℓ; in fact imposing a bound on the number of disks contained in a strip of width 4, which extends three units to the left of ℓ and one unit to the right of ℓ would be enough.

3 Arbitrary Disks

For the reconfiguration of systems of arbitrary disks we obtain tight bounds (modulo lower order terms):

Theorem 3. *Given a pair of start and target configurations, consisting of n disks of arbitrary radii each, $2n$ moves always suffice for transforming the start configuration into the target configuration. The entire motion can be computed in $O(n \log n)$ time. On the other hand, there exist pairs of configurations that require $2n - o(n)$ moves for this task, for every sufficiently large n.*

Proof. The upper bound is immediate, using the universal reconfiguration algorithm described above. The recursive lower bound construction is depicted in Figure 6. The basic construction in Figure 5 (which will be repeated recursively)

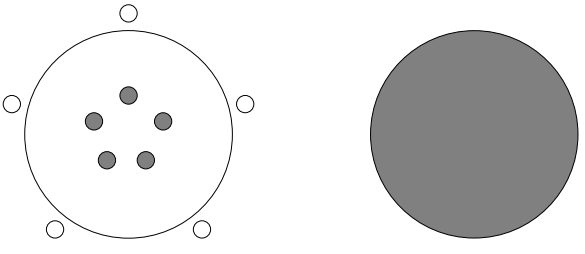


Fig. 5. A simple lower bound construction (basic step for the recursive construction) for sliding disks of arbitrary radii

gives a lower bound of $\approx 3n/2$: it consists of a large disk surrounded by $n-1$ small disks, whose centers form a regular polygon with $n-1$ vertices (let n be even). The target configuration has all small disks inside the original big disk and the large disk somewhere else. No small disk target can be filled before the large disk moves away, that is, before roughly half of the $n-1$ small disks move away. So about $3n/2$ moves in total are necessary.

The recursive construction is obtained by replacing the small disks around a big one by the "same" construction scaled (see Figure 6). To make it work we

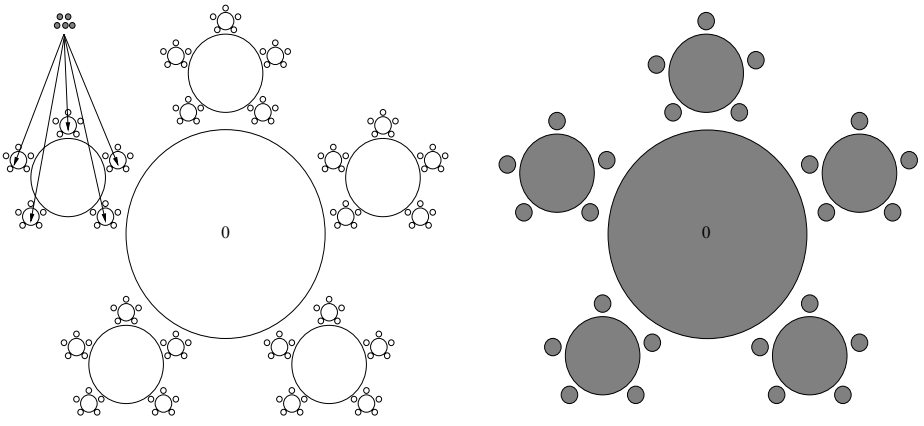


Fig. 6. Recursive lower bound construction for sliding disks of arbitrary radii: $m = 2$ and $k = 3$

choose: 1) all disks of distinct radii, and 2) the small disks on the last level or recursion have targets inside the big ones they surround (the other disks have targets somewhere else). Since all disks have distinct radii, one can think of them as being labeled. If there are k levels in the recursion, about $n/2+n/4+\cdots+n/2^k$ non-target moves are necessary. The precise calculation follows.

There is one large disk labeled 0, and $2m+1$ groups of smaller disks around it close to the vertices of a regular $(2m+1)$-gon ($m \geq 1$). Let m be fixed, and k be the number of levels in the recursion (m and k will be chosen later). Let $n = N(k)$ be the number of disks in the set, and $x = X(k)$ be the number of non-target moves performed (at level k). We have

$$N(0) = 1, \quad X(0) \geq 0, \quad N(1) = 2m + 2, \quad X(1) \geq m.$$

$N(k)$ and $X(k)$ satisfy the following recurrences:

$$N(k) = (2m+1)N(k-1) + 1,$$

$$X(k) \geq mN(k-1)+(m+1)mN(k-2)+\ldots+(m+1)^{k-2}mN(1)+(m+1)^{k-1}m.$$

The first recurrence gives

$$N(k) = (2m+1)^k + \ldots + (2m+1) + 1 = \frac{(2m+1)^{k+1}-1}{2m}.$$

Plugging this into the inequality for $X(k)$ yields

$$X(k) \geq m \sum_{i=0}^{k-1} \frac{(2m+1)^{k-i}-1}{2m}(m+1)^i = \frac{1}{2}\sum_{i=0}^{k-1}((2m+1)^{k-i}-1)(m+1)^i.$$

Using standard manipulations, the above inequality becomes

$$X(k) \geq \frac{(2m+1)^{k+1} - 2(m+1)^{k+1} + 1}{2m}.$$

This can be rewritten as

$$X(k) \geq \frac{(2m+1)^{k+1} - 1 - 2(m+1)^{k+1} + 2}{2m} = n - \frac{(m+1)^{k+1} - 1}{m}.$$

Put

$$z = \frac{(m+1)^{k+1} - 1}{m}.$$

Then

$$\frac{z}{n} = 2\frac{(m+1)^{k+1} - 1}{(2m+1)^{k+1} - 1} \leq 2\left(\frac{m+1}{2m+1}\right)^{k+1} \to 0, \quad \text{for} \quad k \to \infty.$$

Thus $n + x \geq 2n - z = 2n - o(n)$ and the lower bound follows for $n = N(k)$. The same result carries over for all sufficiently large n. In particular for $m = 1$, we get $n + x = 2n - O(n^{\log_3 2}) = 2n - O(n^{0.631})$. □

Acknowledgement. The authors thank Jan Siwanowicz for his valuable remarks and for many interesting conversations on the topic.

References

1. M. Abellanas, F. Hurtado, A. G. Olaverri, D. Rappaport, and J. Tejel, Moving coins. Short version in *Abstracts of Japan Conference on Discrete and Computational Geometry*, 2004. Full version submitted to LNCS Proceedings.
2. N. Alon, M. Katchalski, and W. R. Pulleyblank, Cutting disjoint disks by straight lines, *Discrete & Computational Geometry*, **4** (1989), 239–243.
3. K. Böröczky, Über stabile Kreis- und Kugelsysteme (in German), *Ann. Univ. Sci. Budapest. Eötvös Sect. Math.* **7** (1964), 79–82.
4. P. Brass, W. O. J. Moser, and J. Pach, *Research Problems in Discrete Geometry*, Springer–Verlag, 2005, to appear.
5. N. G. de Bruijn, Aufgaben 17 and 18 (in Dutch), Nieuw Archief voor Wiskunde **2** (1954), 67.
6. E. Demaine, M. Demaine, and H. Verrill, Coin-moving puzzles, in *More Games of No Chance*, edited by R. J. Nowakowski, pp. 405-431, Cambridge University Press, 2002.
7. A. Dumitrescu and J. Pach, Pushing squares around, *Proceedings of the 20-th Annual Symposium on Computational Geometry*, (SOCG'04), NY, June 2004, 166–123.
8. L. Fejes Tóth and A. Heppes, Über stabile Körpersysteme (in German), *Compositio Mathematica*, **15** (1963), 119–126.
9. L. Guibas and F. F. Yao, On translating a set of rectangles, in *Computational Geometry*, F. Preparata (ed.), pp. 61–67, Vol. 1 of *Advances in Computing Research*, JAI Press, London, 1983.

Weighted Ham-Sandwich Cuts

Prosenjit Bose[1,⋆] and Stefan Langerman[2,⋆⋆]

[1] School of Computer Science, Carleton University, Canada
jit@scs.carleton.ca
[2] Département d'Informatique, Université Libre de Bruxelles, Belgium
stefan.langerman@ulb.ac.be

Abstract. Let R and B be two sets of n points. A ham-sandwich cut is a line that simultaneously bisects R and B, and is known to always exist. This notion can be generalized to the case where each point $p \in R \cup B$ is associated with a weight w_p. A ham-sandwich cut can still be proved to exist, even if weights are allowed to be negative. In this paper, we present a $O(n \log n)$ algorithm to find a weighted ham-sandwich cut, but we show that deciding whether that cut is unique is 3-SUM hard.

1 Introduction

Let $R, B \subseteq \mathbb{R}^2$ be two finite point sets, with $|R| + |B| = n$. We call the elements of R the *red points* and the elements of B the *blue points*. A line L is said to *bisect* a set $S \subseteq \mathbb{R}^2$ if each of the two open halfplanes L^+ and L^- bounded by L contain no more than half of the points, i.e., if $||S \cap L^+| - |S \cap L^-|| \leq |S \cap L|$. The *ham-sandwich cut theorem* for point sets states that there always exists a line L that simultaneously bisect R and B. Such a line is called a *ham-sandwich cut*.

Megiddo [21] showed that, if the sets R and B are linearly separable (there exists a line that separates R from B) then a ham-sandwich cut can be found in $O(n)$ time. Edelsbrunner and Waupotitisch [10] modified Meggido's method and obtained an $O(n \log n)$ time algorithm for the general case. An optimal $O(n)$ time algorithm was found by Lo and Steiger [19]. In [7], it was shown that although a ham-sandwich cut can be computed in linear time, determining if that ham-sandwich cut is unique has a lower bound of $\Omega(n \log n)$. The best known algorithm for that task constructs and walks the median level of an arrangement of lines and so runs $O(n^{4/3})$ expected time [6].

The problem of computing ham-sandwich cuts in d dimensions, $d \geq 3$ has been considered by Lo et al [18]. Several generalizations of planar ham-sandwich cuts have also been proposed [1,2,3,4,5,8,15,16,23,24].

In this paper, we consider a generalization where every point p is associated with a weight $w_p \in \mathbb{R}$, positive or negative. Let $W(S) = \sum_{p \in S} w_p$. A line L bisects a set S of weighted points if

$$|W(S \cap L^+) - W(S \cap L^-)| \leq |W(S \cap L)|$$

⋆ Research supported in part by NSERC.
⋆⋆ Chercheur qualifié du FNRS.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 48–53, 2005.

and a ham-sandwich cut for R and B is a line that bisects both weighted sets simultaneously. The ham-sandwich cut theorem can be shown to hold in the weighted case: a careful reading of the proof using the Borsuk-Ulam theorem [20] reveals that the proof never requires the weights to be positive.

In the next section, we give a $O(n \log n)$ algorithm that finds such a cut, but we show in section 3 that deciding if the ham-sandwich cut is unique is 3-SUM hard, and so, that it is unlikely that a $o(n^2)$ algorithm exists for this problem. Determining uniqueness, or finding all ham-sandwich cuts, can be done in $O(n^2)$ time using topological sweep algorithm in the dual arrangement.

2 Algorithm

To simplify the algorithm, we assume that the points are in general position (no three points collinear, no two points on the same vertical). This assumption can be removed using known techniques [9]. We also add a formal infinitesimal to the weight of one of the red points and one of the blue points. This ensures that any line bisecting the blue set (respectively red set) is incident to at least one blue (red) point. Thus, the ham-sandwich cut L is incident to exactly one red point and one blue point, and L is also a valid ham-sandwich cut for the original weights.

Duality Transform. We use the standard duality transform that maps a the point $p = (a, b)$ to the line $T_p = \{(x, y)|y = ax + b\}$, and the line $L = \{(x, y)|y = mx + c\}$ to the point $(-m, c)$. This duality preserves incidences and above/below relations. The sets of points R and B then become sets of lines T_R and T_B. Because a bisecting line for R (respectively B) is incident to at least one point of R (respectively B), the bisectors of R (respectively B) correspond to a set of piecewise linear curves composed of edges of the arrangement T_R (respectively T_B). The curves are infinite or closed. A point at the intersection between a blue curve and a red curve is the dual of a ham-sandwich cut. It is easy to show, e.g. using the Borsuk-Ulam theorem, that the number of intersections between red and blue curves is odd.

Pruning Mechanism. We use the line partition pruning mechanism from [17]: given lines $L = \{\ell_1, \ldots, \ell_n\}$ in general position in R^2 (i.e. no three lines intersect in a common point and no line is vertical) and two other non-parallel lines ℓ_A and ℓ_B, not in L, nor parallel to any line in L, we write $C = \ell_A \cap \ell_B$. Every line in L crosses both ℓ_A and ℓ_B. The lines in L are partitionned into 4 sets, one for each quadrant ($I = +,+$, $II = +,-$, $III = -,-$, $IV = -,+$), depending on whether the line crosses ℓ_A above or below C, and whether it crosses ℓ_B above or below C (see Fig. 1). A line that belongs to one of these sets (quadrants) avoids the opposite quadrant. The lines ℓ_A and ℓ_B form an *α-partition* of L if each of the four groups contains at least αn lines. We use the following lemma from [17]:

Lemma 1. *Let $L = \{\ell_1, \ldots, \ell_n\}$ be a set of n lines in general position in R^2. There exists an $(1/4)$-partition of L and it can be found in time $O(n)$.*

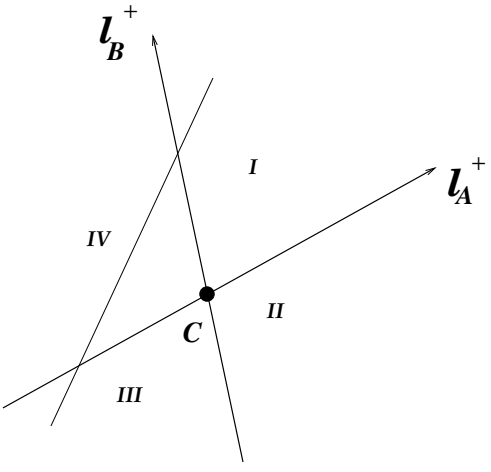


Fig. 1. Pruning Mechanism

The algorithm will start by constructing a (1/4)-partition of the set of lines $T_R \cup T_B$. Each of the 4 quadrants constructed will thus be intersected by at most $3n/4$ lines.

Because the total number of intersections between the red and blue curves is odd, at least one of the four quadrants contains an odd number of those intersections. After finding that quadrant, we prune the lines it does not intersect, and recurse restricting the search to that quadrant. As we prune lines, we keep track of the restricted region in which we concentrate the search, and maintain the total weight of the pruned red and blue lines above and below that region. After each pruning step, a quarter of the lines is eliminated.

Parity. We still have to explain how to compute the parity of the intersections inside a region. We use the following lemma. A d-dimensional version of this lemma was proved by Edgar Ramos [22].

Lemma 2. *Consider a jordan curve C, a set of red curves $\mathfrak{R}$ and a set of blue curves $\mathfrak{B}$, where all curves are infinite or closed. Let $r_1, \ldots, r_k$ be the intersections of the curves of $\mathfrak{R}$ and C, in clockwise order along C, and let x_i be the number of intersections of C with the curves from $\mathfrak{B}$ between r_i and r_{i+1}. Then the parity of the number of intersections between red and blue curves inside C is the parity of $x_2 + x_4 + x_6 + \ldots + x_k$.*

This implies that we can check in $O(n \log n)$ time if the number of points dual to ham-sandwich cuts inside a given triangle is odd. This completes the algorithm.

Theorem 1. *Given two sets $R, B \subseteq \mathbb{R}^2$ of weighted points, with $|R| + |B| = n$, a weighted ham-sandwich cut can be computed in $O(n \log n)$ time.*

3 Hardness

In this section, we show that determining whether or not a weighted ham-sandwich cut is unique is a 3SUM hard problem. The class of 3SUM hard problems was initially studied by Gajentaan and Overmars [14]. Many different problems are known to be 3SUM hard. Quadratic lower bounds for 3SUM hard problems have been shown in restricted models of computation [13,12,11]. Thus, showing that a problem is 3SUM hard provides some evidence that there may not exist a $o(n^2)$ time solution for the problem.

For the reduction, we use the following 3SUM hard problem [14]: Given three sets A, B, and C of n integers total, determine if there are three integers $a \in A, b \in B$ and $c \in C$ such that $a + b = 2c$.

Theorem 2. *Given two sets $R, B \subseteq \mathbb{R}^2$ of weighted points, with $|R| + |B| = n$, determining whether the weighted ham-sandwich cut is unique is 3SUM hard.*

Proof. Given the three sets A, B, and C of n integers total, we produce a set of $2n+4$ weighted blue and red points, placed on three horizontal lines, one line for each set. Points in A, B, and C are placed on horizontal lines $z = 3$, $z = 1$, and $z = 2$, respectively. The points on lines A and C are colored blue, while all points on B are colored red. Each integer s is represented by a pair of points on the line corresponding to its set. One point of weight $+1$ at x coordinate $s - \varepsilon$ and one point of weight -1 at x coordinate $s + \varepsilon$, for some small $0 < \varepsilon < 1/2$. We will refer to this pair of points as a *weighted pair*. Let S be the set of $2n$ weighted points created so far.

To S, we now add four points of weight 1 far on the right of the three lines, one red point on line A, and three blue points on line B (by "far on the right", we mean that the distance from the newly added points to any point in S is at least twice the diameter of S). Thus, the total weight of the blue set of points is 3, and the total weight of the red set of points is 1. Because the total weights of both sets is odd, any ham-sandwich cut must be incident to one blue point and one red point. See Fig. 2.

We claim that the rightmost vertical line is the unique ham-sandwich cut if and only if there are no three integers $a \in A$, $b \in B$, $c \in C$, such that $a + b = 2c$. Clearly, if three such integers exist, then there is a line ℓ incident to the three right points of the pair corresponding to these three elements, each having a weight of -1. The weight of the blue points is 2 to the left of ℓ, 3 to the right of ℓ and -2 on ℓ, while the weight of the red points is 1 to the left and to the

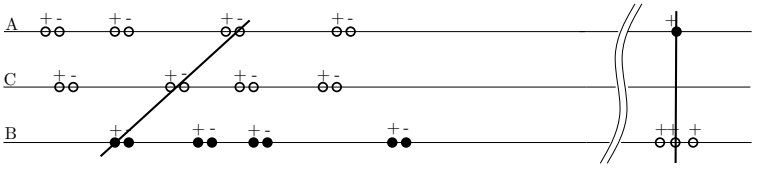


Fig. 2. Uniqueness of the weighted ham-sandwich cut is 3SUM hard

right of ℓ and -1 on ℓ, so ℓ is a weighted ham-sandwich cut and so the weighted ham-sandwich cut is not unique.

If no such three integers exist, then we have to show that no line other than the rightmost vertical one is a ham-sandwich cut. To show this, consider the red and blue point incident to a candidate cut line ℓ, and first assume that the incident points are not the rightmost extra points. Now, we claim that no line ℓ can separate a weighted pair on each of the three lines. This is because the existence of such a line implies the existence of three integers integers $a \in A$, $b \in B$, $c \in C$, such that $a + b = 2c$. Therefore, ℓ can separate at most two weighted pairs.

However, this implies that ℓ has blue weight 0 or 1 to the left of ℓ, $+1$ or -1 on ℓ and 2 or 3 to the right of ℓ, respectively, depending on whether the blue incident point is of weight $+1$ or -1. So ℓ cannot be a ham-sandwich cut.

If line ℓ goes through one of the blue points on the line of B, then it can only be incident to a red point by going through the red point on A (which is the claimed unique cut), or ℓ is the horizontal line corresponding to set B, which is not a ham-sandwich cut since the red weight is 0 below ℓ, 0 on ℓ, and 1 above ℓ.

If ℓ goes through the red point on A, then there are three cases: ℓ can be incident to one of the 3 points on B (which is again the claimed unique cut), or ℓ is the horizontal line corresponding to set A, (which is not a ham-sandwich cut since the blue weight is 3 below ℓ, 0 on ℓ and 0 above ℓ). In the third case, ℓ could be incident to some blue point on C. But then, the blue weight incident to ℓ is at most 1, the blue weight to the left of ℓ is 0 or 1 (if ℓ goes through a blue point of weight $+1$ or -1, respectively) and the blue weight to the right ℓ is 2 if ℓ is incident to a blue point of weight $+1$ or 3 otherwise. Since the difference is 2, ℓ is not a ham-sandwich cut. □

References

1. T. Abbott, E. D. Demaine, M. L. Demaine, D. Kane, S. Langerman, J. Nelson, and V. Yeung. Dynamic ham-sandwich cuts of convex polygons in the plane. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG 2005)*, Windsor, Ontario, Canada, to appear.
2. J. Akiyama, G. Nakamura, E. Rivera-Campo, and J. Urrutia. Perfect divisions of a cake. In *Proc. Canad. Conf. Comput. Geom. (CCCG 98)*, pages 114–115, 1998.
3. S. Bereg, P. Bose, and D. Kirkpatrick. Equitable subdivisions of polygonal regions. *Comput. Geom. Theory and Appl.*, to appear.
4. S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink. Generalizing ham sandwich cuts to equitable subdivisions. *Discrete Comput. Geom.*, 24:605–622, 2000.
5. P. Bose, E. D. Demaine, F. Hurtado, J. Iacono, S. Langerman, and P. Morin. Geodesic ham-sandwich cuts. In *Proc. of the 2004 ACM Symp. on Computational Geometry*, pages 1–9, 2004.
6. T. Chan. Remarks on k-level algorithms in the plane. Manuscript, 1999.
7. H. Chien and W. Steiger. Some geometric lower bounds. In *Proc. 6th Annu. Internat. Sympos. Algorithms Comput.*, volume 1004 of *LNCS*, pages 72–81. Springer-Verlag, 1995.

8. M. Díaz and J. O'Rourke. Ham-sandwich sectioning of polygons. In *Proc. Canad. Conf. Comput. Geom. (CCCG 90)*, pages 282–286, 1990.
9. H. Edelsbrunner and E. P. Mücke. Simulation of Simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.
10. H. Edelsbrunner and R. Waupotitisch. Computing a ham sandwich cut in two dimensions. *J. Symbolic Comput.*, 2:171–178, 1986.
11. J. Erickson. Lower bounds for linear satisfiability problems. *Chicago J. Theoret. Comp. Sci.*, (8), 1999.
12. J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM J. Comput.*, 28:1198–1214, 1999.
13. J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete Comput. Geom.*, 13:41–57, 1995.
14. A. Gajentaan and M. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory and Appl.*, 5:165–185, 1995.
15. H. Ito, H. Uehara, and M. Yokoyama. 2-dimension ham sandwich theorem for partitioning into three convex pieces. In *Proc. Japan. Conf. Discrete Comput. Geom. (JCDCG'98)*, volume 1763 of *Lecture Notes in Computer Science*, pages 129–157. Springer-Verlag, 1998.
16. H. Ito, H. Uehara, and M. Yokoyama. A generalization of 2-dimension ham sandwich theorem. *TIEICE: IEICE Trans. Comm./Elec./Inf./Sys.*, 2001.
17. S. Langerman and W. Steiger. Optimization in arrangements. In *Proceedings of the 20th International Symposium on Theoretical Aspects of Computer Science (STACS 2003)*, volume 2607 of *LNCS*, pages 50–61. Springer-Verlag, 2003.
18. C.-Y. Lo, J. Matoušek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, 11:433–452, 1994.
19. C.-Y. Lo and W. L. Steiger. An optimal time algorithm for ham-sandwich cuts in the plane. In *Proc. Canad. Conf. Comput. Geom. (CCCG 90)*, pages 5–9, 1990.
20. J. Matousek. *Using the Borsuk-Ulam theorem.* Springer, 2003.
21. N. Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, pages 430–433, 1985.
22. E. A. Ramos. Equipartition of mass distributions by hyperplanes. *Discrete Comput. Geom.*, 15(2):147–167, Feb. 1996.
23. I. Stojmenovic. Bisections and ham-sandwich cuts of convex polygons and polyhedra. *Inf. Process. Lett.*, 38(1):15–21, 1991.
24. A. H. Stone and J. W. Tukey. Generalized 'sandwich' theorems. *Duke Math. J.*, 9:356–359, 1942.

Towards Faster Linear-Sized Nets for Axis-Aligned Boxes in the Plane

Hervé Brönnimann*

Computer and Information Science, Polytechnic University,
Six Metrotech Center, Brooklyn, NY 11201, USA

Abstract. Let $\mathcal{B}$ be any set of n axis-aligned boxes in $\mathbb{R}^d$, $d \geq 1$. We call a subset $\mathcal{N} \subseteq \mathcal{B}$ a $(1/c)$-*net* for $\mathcal{B}$ if any $p \in \mathbb{R}^d$ contained in more than n/c boxes of $\mathcal{B}$ must be contained in a box of $\mathcal{N}$, or equivalently if a point not contained in any box in $\mathcal{N}$ can only stab at most n/c boxes of $\mathcal{B}$. General VC-dimension theory guarantees the existence of $(1/c)$-nets of size $O(c \log c)$ for any fixed d, the constant in the big-Oh depending on d, and Matoušek [8, 9] showed how to compute such a net in time $O(nc^{O(1)})$, or even $O(n \log c + c^{O(1)})$ which is $O(n \log c)$ if c is small enough. In this paper, we conjecture that axis-aligned boxes in $\mathbb{R}^2$ admit $(1/c)$-nets of size $O(c)$, and that we can even compute such a net in time $O(n \log c)$, for any c between 1 and n. We show this to be true for intervals on the real line, and for various special cases (quadrants and skylines, which are unbounded in two and one directions respectively). In a follow-up version, we also show this to be true with various fatness We also investigate generalizations to higher dimensions.

1 Introduction

Let $\mathcal{B}$ be any set of n axis-aligned boxes in $\mathbb{R}^d$, $d \geq 1$. For any point p, we define the subset $\mathcal{B}_p$ of $\mathcal{B}$ as $\mathcal{B}_p = \{B \in \mathcal{B} : p \in B\}$. A box B in $\mathcal{B}_p$ is said to be *stabbed* by p. A subset $\mathcal{N} \subseteq \mathcal{B}$ is a $(1/c)$-*net* for $\mathcal{B}$ if $\mathcal{N}_p \neq \emptyset$ for any $p \in \mathbb{R}^d$ such that $|\mathcal{B}_p| > n/c$, or equivalently, if a point not contained in any box in $\mathcal{N}$ can only stab $|\mathcal{B}_p| \leq n/c$ boxes of $\mathcal{B}$. In particular, $\emptyset$ is a 1-net, and $\mathcal{B}$ is a 0-net.

The shatter function $b(n)$ of boxes is the maximum possible number of distinct ways to stab $\mathcal{B}$, i.e. subsets of the form $\mathcal{B}_p$, over all sets $\mathcal{B}$ of n boxes. A set $\mathcal{B}$ of k boxes is *shattered* if there exist 2^k points $p_1, \ldots, p_{2^k}$ such that $\{\mathcal{B}_{p_i} \ : \ i = 1, \ldots, 2^k\} = 2^{\mathcal{B}}$, i.e. their arrangement gives rise to all possible combinations of boxes. The *VC-dimension* k of is the maximum size of a shattered set of boxes, or equivalently the largest integer k such that $b(k) = 2^k$. The set system described above has finite VC-dimension at most $2d$. A classical lemma, rediscovered independently by (Perles and) Shelah [15], Sauer [14], and Vapnik-Červonenkis [16], implies that the shatter function of a set system of VC-dimension k is bounded above by $\sum_{i=0}^{k} \binom{n}{i} = O(n^k)$. On the other hand, direct considerations imply that the number of distinct subsets $\mathcal{B}_p$ is at most

* Research of the author has been supported by NSF CAREER Grant CCR-0133599.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 54–61, 2005.

$(2n+1)^d$, since this is an upper bound to the number of cells in the arrangement of $\mathcal{B}$. This ensures that there always exists $(1/c)$-nets of size $O(dc\log(dc))$, and that they can be found in time $O_d(n)c^d$, using quite general machinery (see for example the books by Matoušek [6] or by Pach and Agarwal [12]).

In this paper, we investigate a fast, $O(n+n\log c)$-time construction of $(1/c)$-nets of size $O(c)$ for any value $1 \leq c \leq n$ and $d = 2$. The following unfortunately remains a conjecture:

Conjecture 1. Let $\mathcal{B}$ be a set of n axis-aligned boxes in $\mathbb{R}^2$ and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O(c)$.

We can prove a similar result only for special cases: segments on the real line (the one-dimensional case), quadrants of the form $(-\infty, x] \times (-\infty, y]$ in $\mathbb{R}^2$, and unbounded boxes of the form $[x_1, x_2] \times (-\infty, y]$ (which we call a *skyline*). The conjecture still stands for boxes. In those cases, however, we can also provide algorithms that run in time $O(n + n\log c)$ and return a $(1/c)$-net of size $O(c)$. There are very few systems that are known to admit $O(c)$-size $(1/c)$-nets [7], and so our conjecture, if it were true, would provide another interesting example. In addition, finding small nets efficiently has various applications, for instance for divide-and-conquer, or for computing the area of a union of rectangles (the Klee measure problem). One should note that recently, Ezra and Sharir [4] used a similar construction for an output-sensitive construction of the union of triangles in two phases: first cover the "deep" region of the union with a $(1/c)$-net, and then do extra work to find the "shallow" remainder of the union. Even more recently, Clarkson and Varadarajan [2] proved that unit cubes admit $O(c)$-size $(1/c)$-nets in any dimension, and give a randomized polynomial time algorithm to find such a net.

As in [7], our constructions do not generalize to higher dimensions although we can prove a bound of $O(c^{d-1})$ for the size of the resulting net in the special case of orthants. Although much higher than $O(dc\log(dc))$ for large values of c, the running time of $O(n+n\log c)$ might still make the deterministic construction attractive.

Our results are related to a technique developed by Nielsen for fast stabbing of boxes [10]. In his paper, he selects a small set of points that stabs all the boxes in $\mathcal{B}$, a problem which for congruent boxes is dual to covering a set of points by boxes. (For intervals, Katz et al. [11] apply similar techniques to maintain the piercing set under dynamic updates.) Interestingly, our problem can be phrased as selecting a small subset of the boxes that covers all the portion of depth at least n/c in the arrangement of the boxes.

2 Intervals on the Line

We first prove that it is easy to find small nets for intervals on the line, the one-dimensional case of the problem above.

Theorem 1. *Let $\mathcal{B}$ be a set of n intervals on the real line $\mathbb{R}$ and c be any parameter $1 \leq c \leq n$. There exists a subset $\mathcal{N}$ of at most $2\lceil c-1 \rceil$ boxes in $\mathcal{B}$ that is a $(1/c)$-net for $\mathcal{B}$. Such a set can be found in $O(n + n \log c)$ time.*

Proof. We give three constructions for such nets $\mathcal{N}$. The first one shows the intuition very clearly, but does not lead to a good runtime. The second has a good runtime, and is historically our first. The third builds on the first two and is the one that will best generalize for the sequel.

1. Consider the set $\mathcal{E}$ of all endpoints of the segments, be they left or right endpoints. The set $\mathcal{E}$ contains $2n$ real numbers. We may assume that they are all distinct and sorted. (In fact, the construction will yield only fewer segments if some segments share their endpoints.) Now select all the $\lfloor 2kn/c \rfloor$-th elements in $\mathcal{E}$, for $k = 1, \ldots, \lceil c-1 \rceil$, in order to form a set $\mathcal{E}'$. Between two consecutive elements in $\mathcal{E}'$, there can be at most $2n/c$ elements of $\mathcal{E}$. (See Figure 2.)

For each element x in $\mathcal{E}'$, compute the set of segments stabbed by x. Of all these segments, choose the one whose right endpoint is the rightmost. In a similar way, choose the interval stabbed by x whose left endpoint is the leftmost. This yields a set $\mathcal{N}$ of at most $2\lceil c-1 \rceil$ segments. (Some segments may be picked several times.)

Now let's see that it is a $(1/c)$-net. Indeed, if a point p does not stab any of the segments in $\mathcal{N}$, then any segment that it stabs must be entirely contained between two consecutive points of $\mathcal{E}'$. There are at most $2n/c$ endpoints between two consecutive points in $\mathcal{E}'$ and each of these segments takes two, so there can be at most n/c such segments.

2. Unfortunately, it is prohibitively expensive to compute all the segments stabbed by each point of $\mathcal{E}'$, because there can be $\Omega(nc)$ such pairs. Another construction in the spirit of Nielsen [10] is to compute the median endpoint p_m and split the segments into three groups, C consisting of all the segments containing p_m, L and R consisting of those segments entirely on the left or on the right of p_m. (Note that L and R have at most $n/2$ segments each.) We compute the leftmost and rightmost segments in C and add both to $\mathcal{N}$, and recurse on L and R unless 1. there are fewer than n/c segments left to recurse into, or 2. those segments are completely covered by the segments in $\mathcal{N}$ (this can easily be checked in $O(1)$ time by maintaining the union of the segments in $\mathcal{N}$ during the recursion). The depth of the recursion is at most $O(\log c)$ and there are at most c medians computed, hence the size of $\mathcal{N}$ is at most $2c$ and the time spent in all the median and splitting computations at a given level of recursion is $O(n)$ for a total of $O(n \log c)$.

3. Our final construction uses a set $\mathcal{E}''$ similar to $\mathcal{E}'$ but with only n/c endpoints between two consecutive positions. $\mathcal{E}''$ decomposes the line $\mathbb{R}$ into $2c$ elementary intervals, and we construct a complete binary tree on top of these where, to each internal node, corresponds an interval that is the union of the elementary intervals in its subtree (as with a segment tree, see [1]). Each segment in $\mathcal{B}$ is inserted in the tree by locating both endpoints, and is appended into the at

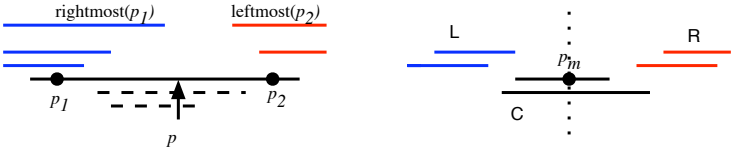


Fig. 1. For the proof of Theorem 1, construction 2: (left) The set of segments stabbed by a point p (drawn with dashes on the left) must stay within the two endpoints in $\mathcal{E}'$ enclosing p, or else it p must also stab rightmost(p_1) or leftmost(p_2). (right) The median endpoint p_m partitions the intervals into three groups, C (crossing) and L and R.

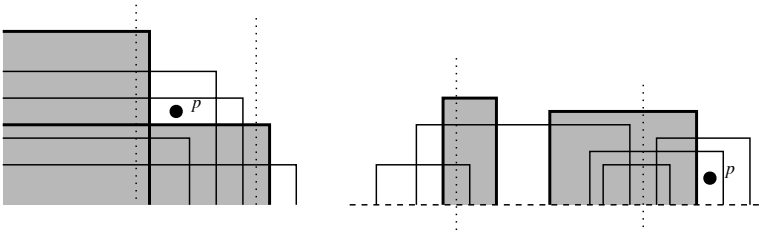


Fig. 2. On the left, the net for the quadrants is shown in bold; a point p that avoids $\mathcal{N}$ can only stab the quadrants between two consecutive positions of $\mathcal{E}'$ (shown in vertical dashed lines). On the right, a skyline is shown and its net is in bold; a point p that doesn't stab $\mathcal{N}$ cannot stab too many boxes.

most $2(\log c)$ nodes whose interval it maximally contains (see Figure 2). Finally, for every elementary interval (leaf of the tree), we pick any segment in $\mathcal{B}$ that completely spans it and collect those as $\mathcal{N}$: if such a segment exists, it is stored at one of the ancestors of the leaf. If none exists, any segment that intersects the elementary interval of the leaf must have one endpoint in it and so there can be only at most n/c such segments. Any point that does not stab a segment of $\mathcal{N}$ can only intersect those n/c segments for the elementary interval into which it falls. As described, the whole tree and list of segments at every node take $O(n \log c)$ storage and time to construct. With a bit of care, the construction can be realized without actually constructing the tree, in $O(n \log c)$ time and $O(n)$ space. □

3 Rectangles in the Plane

We generalize the method of the previous paragraph to the plane. We begin with the easier problem when all the boxes are south-west quadrants, i.e. they contain the point $(-\infty, -\infty)$.

Theorem 2. *Let $\mathcal{B}$ be a set of n quadrants with the same orientation in $\mathbb{R}^2$ (north/south-east/west) and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $\lceil c - 1 \rceil$. Such a net can be found in time $O(n + n \log c)$.*

Proof. Project the rightmost vertical boundaries on the x-axis, forming a set $\mathcal{E}$ of abscissae. As in the previous section, we may form a set $\mathcal{E}'$ by selecting all the $\lfloor kn/c \rfloor$-th elements in $\mathcal{E}$, for $k = 1, \ldots, \lceil c-1 \rceil$. The other elements may be sorted within $\mathcal{E}'$ by binary searching $\mathcal{E}'$, in total time $O(n + n \log c)$ again. So each quadrant is assigned the point of $\mathcal{E}'$ immediately to its left. The set $\mathcal{N}$ may be selected by sweeping over the positions in $\mathcal{E}'$ from the right to the left and maintaining the higher quadrant visited so far. Each time the sweep line sweeps over an abscissa x in $\mathcal{E}'$, the current higher quadrant B is updated with the quadrants assigned to x. This can all be performed in time $O(n)$.

The union of all these quadrants is a set $\mathcal{N}$ of size $\lceil c-1 \rceil$. Then $\mathcal{N}$ is a $(1/c)$-net because each point can only stab the quadrants between two consecutive elements of $\mathcal{E}'$. □

A *skyline* is a set of boxes that all intersect a common line. We are only interested in what happens on one side of that line, so we can consider unbounded boxes of the form $[x_1, x_2] \times (-\infty, y]$. We can extend the previous result to a skyline.

Theorem 3. *Let $\mathcal{B}$ be a set of n axis-aligned boxes, all unbounded in some common direction, and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size at most $\lceil 2c-1 \rceil$. Such a net can be found in time $O(n + n \log c)$.*

Proof. Without loss of generality, assume that boxes all intersect the x-axis and are unbounded south, i.e. unbounded boxes of the form $[x_1, x_2] \times (-\infty, y]$. As before, project the vertical boundaries on the x-axis and form a set $\mathcal{E}$ of abscissae, and select $\mathcal{E}'$ as the $\lfloor kn/c \rfloor$-th elements in $\mathcal{E}$, for $k = 1, \ldots, \lceil 2c-1 \rceil$. For each elementary interval $[x_i, x_{i+1}]$ between two consecutive elements of $\mathcal{E}'$, define h_i (if any) to be a box which spans the interval entirely and whose top side is the highest among all such boxes. The union of all the h_i's is $\mathcal{N}$ and has the $(1/c)$-net property. Indeed, a point p in $[x_i, x_{i+1}]$ that does not stab a box of $\mathcal{N}$ does not stab *any* of the boxes that contain the elementary interval, and thus can only stab a box which is entirely contained in the interval, or whose right or left endpoint falls in that interval. In any case, each box it stabs contributes one or two endpoints in that interval, and there are only n/c endpoints between two consecutive elements of $\mathcal{E}'$.

For the computation, reuse the segment tree method of Theorem 1 (construction 3). This time, the only variation is that instead of storing a segment in an internal node, we store the highest box that spans the interval corresponding to that node (the union of the elementary intervals at the leaves of that node's subtree). For finding h_i, we take the highest of those at the $O(\log c)$ ancestor nodes of the i-th leaf. The whole computation is done using $O(n)$ space and time $O(n \log c)$. □

4 Orthants in Higher Dimension

Quadrants and skylines easily generalize to higher dimensions, we call them (generalized) orthants and skylines. Unfortunately, we do not have a construction

for generalized skylines, but we now present how small nets we can find efficiently for orthants. The bound is far from optimal for dimensions greater than 2, as nets of size $O_d(c \log c)$ exist but take much longer to compute (see the introduction). We use the notation $O_d(\ldots)$ to indicate that the constant in the big oh may depend (often exponentially) on d but not on n or c.

Theorem 4. *Let $\mathcal{B}$ be a set of n orthants with the same orientation in $\mathbb{R}^d$ and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O_d(c^{d-1})$. Such a net can be found in time $O(n + n \log c)$.*

Proof. The proof goes by induction on d. Choosing a direction, say that of x_d, and projecting the x_d-boundaries of $\mathcal{B}$ on that direction yields a set $\mathcal{E}$ of x_d values. As in the previous section, we may form a set $\mathcal{E}'$ by selecting all the $\lfloor k(\alpha n/c) \rfloor$-th elements in $\mathcal{E}$, for $k = 1, \ldots, c/\alpha$. For now, we leave the choice of the constant $\alpha < 1$ undetermined. The other elements may be sorted within $\mathcal{E}'$ by binary searching $\mathcal{E}'$, in total time $O(n + n \log c)$ again. So each orthant is assigned the point of $\mathcal{E}'$ with the x_d value immediately smaller. Now for each x_d in $\mathcal{E}'$ and its assigned orthants, we select a $((1-\alpha)/c)$-net for the $(d-1)$-orthants in the plane x_d. Collecting the c/α nets yields a subset $\mathcal{N}$ of orthants.

Clearly, a point that does not stab a box of $\mathcal{N}$ can stab only the boxes assigned to its slice, or a fraction $((1-\alpha)/c)$ of the boxes of the slices above, for a total of at most n/c. Thus $\mathcal{N}$ is a $(1/c)$-net.

If the size of $\mathcal{N}$ is bounded by a function $f_d(c)$, we obtain the induction $f_d(c) \leq (c/\alpha) f_{d-1}(c/(1-\alpha))$, with $f_2(c) = 64c$. Hence

$$f_d(c) \leq \frac{64c^{d-1}}{\alpha^{d-1}(1-\alpha)^{(d-2)(d-1)/2}}.$$

Optimizing for α, we find that the size is $O((\sqrt{e}d/4)^{d-2}c^{d-1})$, which is $O_d(c^{d-1})$. □

Corollary 1. *Let $\mathcal{B}$ be a set of n boxes in $\mathbb{R}^d$ and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O_d(c^{d-1})$. Such a net can be found in time $O(n + n \log c)$.*

For points and halfspaces in $\mathbb{R}^d$, Matoušek, Seidel, and Welzl [7] have shown that there exist $(1/c)$-nets of size $O(c)$. They also show that it suffices to restrict to points in convex position, albeit by having nets bigger by a factor of d. We prove an analog result for orthants, without the blowup factor. The analogue of convex position for orthants is maximal position, as defined by Preparata and Shamos [13].

Theorem 5. *Suppose there exists a $(1/c)$-net of size $s(c)$ for any set of orthants in $\mathbb{R}^d$ in maximal position. Then there exists an $(1/c)$-net of size $s(c)$ for any set of orthants in $\mathbb{R}^d$.*

Proof. Consider any set $\mathcal{B}$ of n orthants, which we may assume to be in general position, and call $\max(\mathcal{B})$ the subset of orthants in $\mathcal{B}$ that are maximal. Suppose

that each orthant is unbounded towards negative coordinates, and call $\mathbf{v}$ the vector $(1, \ldots, 1)$.

We replace each orthant B not in $\max(\mathcal{B})$ by an orthant $\bar{B}$ which is maximal, using the following construction: we first find the orthant $\max(B)$, which is the last orthant in $\max(\mathcal{B})$ that is hit by the ray originating at the corner of B towards $\mathbf{v}$, and then put the orthant $\bar{B}$ with the corner slightly beyond the last hit point (in the direction of the face of $\max(B)$ that is hit). In this way, the orthant $\bar{B}$ is in maximal position. There is nothing special about $\mathbf{v}$, all we need is an orthant in $\mathcal{B}$ that completely dominates B. The construction is explained in Figure 3. If B *is* in $\max(\mathcal{B})$, then we let $\bar{B} = B$, of course. Doing this for every orthant in $\mathcal{B}$ yields a set $\bar{\mathcal{B}}$ of n orthants that are in maximal position.

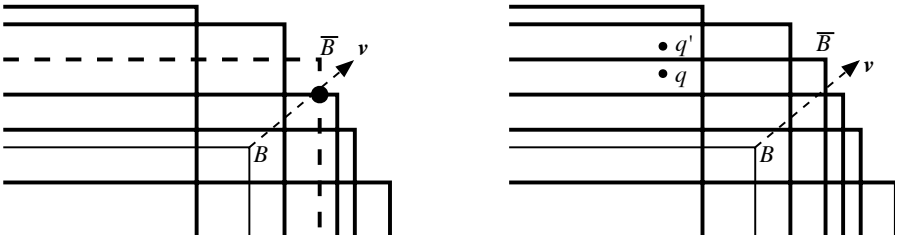


Fig. 3. On the left, how to build a set $\bar{\mathcal{B}}$ of orthants in maximal position. The orthants in $\max(\mathcal{B})$ are drawn in bold lines, the new orthant created for the box B is drawn in dashed line. On the right, the problematic case when q stabs an orthant $\bar{B}$ in $\bar{\mathcal{B}}$ which is not in $\max(B)$: then we replace q by a point q' just outside $\bar{B}$.

By assumption, this set admits a $(1/c)$-net $\bar{\mathcal{N}}$ of size $s(c)$. For each orthant X in $\bar{\mathcal{N}}$, if X is in $\max(\mathcal{B})$, then we put X into $\mathcal{N}$; otherwise, $X = \bar{B}$ for some non-maximal orthant B in $\mathcal{B}$, and we put $\max(B)$ into $\mathcal{N}$. Clearly, $\mathcal{N}$ has as many elements as $\bar{\mathcal{N}}$, or less.

Now why should $\mathcal{N}$ be a $(1/c)$-net for $\mathcal{B}$? Suppose a point q stabs more than n/c orthants in $\mathcal{B}$. Then q can only stab even more orthants in $\bar{\mathcal{B}}$ since we only move orthants upwards in any direction. So q must also stab an orthant X in $\bar{\mathcal{N}}$. If this orthant is in $\max(\mathcal{B})$, then we have proven that q stabs a orthant in $\mathcal{N}$. Otherwise, $X = \bar{B}$ for some orthant B in $\mathcal{B}$. If q stabs $\max(B)$, then again it does stab an orthant in $\mathcal{N}$, so a problem only occurs when q stabs an orthant $X = \bar{B}$ in $\bar{\mathcal{N}}$ but not its corresponding $\max(B)$. As shown in Figure 3, we might move q towards a point q' slightly above X. Then q' does not stab X any more, but it still stabs the same set of orthants in $\mathcal{B}$. Then we can do the same reasoning again with q'. Each time we move q into q', the number of orthants in $\bar{\mathcal{N}}$ that it stabs decreases, so the process must eventually stop, and the only way to stop is to find a orthant X in $\mathcal{N}$ that is stabbed by the final q'. Since the coordinates of q' only increase, that orthant in $\mathcal{N}$ must contain the original q. □

5 Conclusion

The main point of this research is to obtain small nets, fast. All the runtimes we give are $O(n \log c)$, for the whole range of values of c. This research hints at another very natural set system (axis-aligned boxes in the plane) where the general bound $O(c \log c)$ for a $(1/c)$-net can be improved to $O(c)$, and more efficient algorithms can be found. In this paper, we prove this is true for a number of special cases. Komlos, Pach and Woeginger [5] have shown that there exist set systems for which $(1/c)$-nets must have size $\Omega(c \log c)$.

This also poses the analog problem of finding good approximations, in the sense that not only does p hit few boxes if it misses $\mathcal{N}$, but the number of hits in $\mathcal{N}$ reflects the number of hits in $\mathcal{B}$ (scaled by $|\mathcal{N}|/|\mathcal{B}|$). The approach above seems to collapse because nothing guarantees the representativity of $\mathcal{N}$.

References

1. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications. Second Edition.* Springer-Verlag, Heidelberg, 2000.
2. K. Clarkson, K. Varadarajan. Improved Approximation Algorithms for Geometric Set Cover. To appear in *Proceedings of the Twenty First Annual Symposium on Computational Geometry*, 2005, Pisa, Italy.
3. T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms (2nd edition).* MIT Press, Cambridge, MA, 2002.
4. E. Ezra and M. Sharir. Output-sensitive construction of the union of triangles In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 420—429, 2004.
5. J. Komlòs, J. Pach, G.J. Woeginger. Almost Tight Bounds for epsilon-Nets. *Discrete & Computational Geometry* 7:163–173, 1992.
6. J. Matoušek. *Lectures on Discrete Geometry.* Springer, Berlin, 2002.
7. J. Matoušek, R. Seidel and E. Welzl. How to net a lot with little: small ε-nets for disks and halfspaces. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pp.16—22, June 7-9, 1990, Berkeley, California.
8. J. Matoušek. Approximations and optimal geometric divide-and-conquer. *J. Comput. Syst. Sci.* 50(2):203–208, 1995.
9. J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry* 8:315–334, 1992.
10. F. Nielsen, Fast Stabbing of Boxes in High Dimensions. *Theoretical Computer Science*, Elsevier Science, 246(1-2): , 2000.
11. M.J. Katz, F. Nielsen and M. Segal. Maintenance of a Piercing Set for Intervals with Applications *Algorithmica* 36(1):59–73, 2003.
12. J. Pach and P.K. Agarwal. *Combinatorial Geometry.* J. Wiley, New York, 1995.
13. F. Preparata and M.I. Shamos. *Computational Geometry.* Springer, New York, 1985.
14. N. Sauer. On the density of families of sets. *J. Combinatorial Theory* Ser. A, 13:145–147, 1972.
15. S. Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages, *Pacific J. Math.* 41:247-261, 1972.
16. V. Vapnik and A.Ya. Červonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.* 16:264-280, 1971.

Farthest-Point Queries with Geometric and Combinatorial Constraints

Ovidiu Daescu[1,⋆], Ningfang Mi[2], Chan-Su Shin[3,⋆⋆], and Alexander Wolff[4,⋆⋆⋆]

[1] Department of Computer Science,
University of Texas at Dallas, Richardson, TX 75083, USA
daescu@utdallas.edu
[2] Department of Computer Science, College of William and Mary,
P.O. Box 8795, Williamsburg, VA 23187-8795, USA
ningfang@cs.wm.edu
[3] School of Electronics and Information Engineering,
Hankuk University of Foreign Studies, Korea
cssin@hufs.ac.kr
[4] Department of Computer Science, Karlsruhe University,
P.O. Box 6980, D-76128 Karlsruhe, Germany
http://i11www.ira.uka.de/people/awolff

Abstract. In this paper we discuss farthest-point problems in which a set or sequence S of n points in the plane is given in advance and can be preprocessed to answer various queries efficiently. First, we give a data structure that can be used to compute the point farthest from a query line segment in $O(\log^2 n)$ time. Our data structure needs $O(n \log n)$ space and preprocessing time. To the best of our knowledge no solution to this problem has been suggested yet. Second, we show how to use this data structure to obtain an output-sensitive query-based algorithm for polygonal path simplification. Both results are based on a series of data structures for fundamental farthest-point queries that can be reduced to each other.

1 Introduction

Proximity problems are fundamental in computational geometry and have been studied intensively since Knuth [15] posed the post-office problem about three decades ago. In this paper we discuss farthest-point problems in which a set or sequence S of n points in the plane is given in advance and can be preprocessed to answer various queries efficiently. Our main results are the following.

First, we present a data structure that can be used to compute the point farthest from a query line segment in $O(\log^2 n)$ time. Our data structure needs $O(n \log n)$ space and preprocessing time. To the best of our knowledge no solution to this problem has been suggested yet.

⋆ Supported by NSF grant CCF-0430366.
⋆⋆ Supported by Hankuk University of Foreign Studies Research Fund of 2005.
⋆⋆⋆ Supported by grant WO 758/4-1 of the German Science Foundation (DFG).

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 62–75, 2005.

Second, we design a data structure that can be used to simplify polygonal paths in the following sense: given a path $P = (p_1, \ldots, p_n)$ and a real $\Delta > 0$ we want to find a subpath P' of P that goes from p_1 to p_n and consists exclusively of Δ-*approximating segments* according to the *tolerance-zone criterion*, i.e. a sequence of line segments $\overline{p_i p_k}$ with the property that each p_j with $i < j < k$ is at most Δ away from $\overline{p_i p_k}$. We are interested in a min-# subpath, i.e. a subpath with the minimum number of vertices. This is motivated by data reduction (e.g. in geographic information systems) and considered an important problem—finding a near-linear solution is listed as problem 24 in the *Open Problems Project* [17]. Our query-based algorithm finds a min-# subpath in $O(n^2 \log^3 n)$ worst-case running time. This is slightly worse than the quadratic running time of the best incremental algorithm [6], but much better in practice since, as we will see later, the running time of our algorithm is output sensitive. Our algorithm has the same structure as a query-based algorithm [11] for the weaker *infinite-beam criterion* which requires that a vertex p_j of P that is shortcut by an edge $\overline{p_i p_k}$ of P' must be within distance Δ from the *line* through p_i and p_k. The algorithm [11] outperformed an incremental algorithm similar to [6] in an experimental evaluation.

Both main results are based on our solution of the following basic problem:

FarthestVertexInHalfplane (FV-halfplane): Preprocess a convex n-gon C for queries of the following type. Given (q, l_q), where q is a point and l_q is a directed line through q, decide whether there is a vertex of C to the left of l_q. If yes, report the one farthest from q. (See Figure 1.)

Other than one might think at first glance, this problem cannot be solved simply by binary search on the vertices of C since the distance from the query point q is not unimodal on the boundary of C. Our data structure for FV-halfplane answers queries in $O(\log^2 n)$ time given $O(n \log n)$ space and preprocessing time.

Next we address a problem whose solution yields our first main result, an efficient data structure for finding points farthest from query line segments.

FarthestPointInHalfstrip (FP-halfstrip): Preprocess a set S of n points for queries of the following type. Given a triplet (q, l_q, Δ), where q is a point and l_q is a directed line through q such that all points in S are within distance Δ from l_q, decide whether there is a point $p \in S$ such that (i) $|qp| \geq \Delta$, and (ii) the projection of p on l_q lies before q. If yes, report the point farthest from q that fulfills conditions (i) and (ii). (See Figure 2.)

We prove that if there are points fulfilling conditions (i) and (ii), then among these the one farthest from q among them lies on the convex hull of S. Note that this statement does not hold if we drop condition (i): in Figure 3 the point p is farthest from q among all points in S that fulfill condition (ii), but p does not lie on the convex hull of S. Thanks to condition (i), our data structure for FV-halfplane in fact solves FP-halfstrip within the same asymptotic bounds. This in turn yields our first main result: we can preprocess a set S of n points

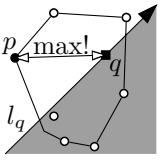


Fig. 1. FV-halfplane

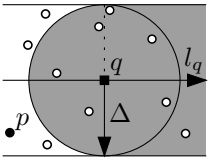


Fig. 2. FP-halfstrip

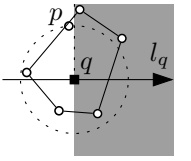


Fig. 3. Counterexample

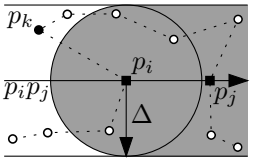


Fig. 4. FIP-halfstrip

in $O(n \log n)$ time and space such that the point in S farthest from a query line segment s can be reported in $O(\log^2 n)$ time.

For our second main result, which deals with polygonal path simplification, point order is important. Thus we consider an indexed version of FP-halfstrip:

FarthestIndexedPointInHalfstrip (FIP-halfstrip): Preprocess a sequence $S = (p_1, \ldots, p_n)$ of points for queries of the following type. Given a triplet (i, j, Δ) such that all points p_k with $i < k < j$ are within distance Δ from the line p_ip_j, decide whether there is a point p_k with $i < k < j$ such that (i) $|p_ip_k| \geq \Delta$, and (ii) the projection of p_k on p_ip_j lies before p_i. If yes, report the point p_k farthest from p_i that fulfills (i) and (ii). (See Figure 4.)

Our time and space bounds for FIP-halfstrip are a log-factor above those for FV-halfplane. The data structure for FIP-halfstrip yields an output-sensitive query-based algorithm for polygonal path simplification. Given a polygonal path $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^2$ and a real $\Delta > 0$, the algorithm computes a subpath of P with the minimum number m_{tz} of vertices among all subpaths satisfying the tolerance-zone criterion. The algorithm runs in $O(F_{\text{tz}}(m_{\text{tz}})\, n \log^3 n)$ time and uses $O(n \log^2 n)$ space, where $F_{\text{tz}}(m_{\text{tz}}) \leq n$ is the number of vertices that can be reached from p_1 with at most $(m_{\text{tz}} - 2)$ Δ-approximating segments.

Next we look at a batched version of an indexed farthest-point problem. Given a sequence S of points, we want to observe how the point farthest from a fixed point p changes over time while we insert the points of S one after the other. In each round we ignore all those points that lie in a halfplane determined by the newly inserted point. Our solution assumes knowledge of S before the observation starts.

BatchedFarthestIndexedPointInHalfplane (BFIP-halfplane): Given a sequence $S = (p_1, \ldots, p_n)$ of points and a point $p \notin S$, decide for each $i \in \{1, \ldots, n\}$ whether there is a point $p_f \in \{p_1, \ldots, p_i\}$ that lies on the same side as p with respect to the perpendicular bisector of p and p_i. If yes, report the point p_f farthest from p that has the above property.

Our algorithm for this problem takes $O(n \log^2 n)$ time and $O(n \log n)$ space.

Our paper is structured as follows. In Section 2 we briefly review related work. In Section 3 we first consider the problem FP-halfplane, a generalization of FV-halfplane where points do not have to be in convex position. In Section 4 we solve the convex case, i.e. FV-halfplane. In Section 5 we show that FP-halfstrip can be reduced to FV-halfplane and how this helps to solve the farthest-point-to-line-segment problem. In Section 6 we show how the data structure for FV-halfplane can be used to solve the indexed problem FIP-halfstrip. Section 7 settles the connection between FIP-halfstrip and polygonal path simplification. In Section 8 we address the batched problem BFIP-halfplane.

2 Previous Work

The problems we study are related to the nearest-point query problem [9, 18, 19] and to the all-pairs farthest- and closest-neighbors problem [21, 1, 3]. Cole and Yap [9] consider closest-point-to-line queries and present a data structure with $O(\log n)$ query time that needs $O(n^2)$ preprocessing time and space. A data structure with $O(n^{0.695})$ query time that needs $O(n \log n)$ preprocessing time and $O(n)$ space is presented by Mitra and Chaudhuri [18]. Using simplicial partitions, Mukhopadhyay [19] constructs in $O(n^{1+\varepsilon})$ time a data structure of size $O(n \log n)$ that finds a point *closest* to a query line in $O(n^{\frac{1}{2}+\varepsilon})$ time for arbitrary $\varepsilon > 0$. Finding a point *farthest* from a query line seems to be easier: it can be done by $O(\log n)$ time given $O(n \log n)$ preprocessing and $O(n)$ space, see Section 5. This data structure helps us to show how to find a point farthest from a query line segment in $O(\log^2 n)$ time given $O(n \log n)$ preprocessing and space. Bespamyatnikh and Snoeyink [5] show how to preprocess a set S of n points in $O(n \log n)$ time using $O(n)$ space such that the point closest to a query line segment *outside* the convex hull of S can be reported in $O(\log n)$ time. Using this data structure, Bespamyatnikh [4] shows how to solve in $O(n \log^2 n)$ time a batched problem where n points and n *disjoint* line segments are given and for each segment the *closest* point has to be determined. In contrast, our data structure for FP-halfstrip (see Section 5) answers *farthest*-point queries for an *arbitrary* line segment in $O(\log^2 n)$ time each, given $O(n \log n)$ space and preprocessing time.

While the all-pairs nearest neighbors of n points in a fixed dimension can be computed in optimal $O(n \log n)$ time [21], no algorithm is known to compute the all-pairs farthest neighbors of n points within the same time bound. Agarwal et al. [1] show that the all-pairs farthest neighbors in $\mathbb{R}^3$ can be computed in $O(n^{4/3} \log^{4/3} n)$ time. If the points are the vertices of a convex polygon in $\mathbb{R}^2$, the all-pairs farthest neighbors can be computed in linear time, even though the problem has a complexity of $\Omega(n \log n)$ for arbitrary points [3]. In $\mathbb{R}^3$ the convex case can be solved in $O(n \log^2 n)$ expected time [8].

Although the closest-point-to-line query problem and the all-pairs farthest neighbors problem are well understood, we are not aware of any published work on the farthest-point problems we consider.

3 Farthest Point in Halfplane

In this section, for completeness we address the following natural generalization of FV-halfplane.

FarthestPointInHalfplane (FP-halfplane): Preprocess a set S of n points for queries of the following type. Given (q, l_q), where q is a point and l_q is a directed line through q, decide whether there is a point in S to the left of l_q. If yes, report the one farthest from q.

We use the following structure: a *simplicial partition* for a set S of n points in the plane is a collection of pairs $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$, where the sets of type S_i partition S, and t_i is a triangle that contains S_i for $i = 1, \ldots, r$. An example of a point set S and a simplicial partition of S of size 4 are given in Figure 5. For a given simplicial partition $\Psi(S)$, the *crossing number* of a line l is the number of triangles of $\Psi(S)$ that l intersects. For example, the line l in Figure 5 has crossing number 3. The crossing number of $\Psi(S)$ is the maximum crossing number over all possible lines l. We say that a simplicial partition $\Psi(S)$ is *fine* if $|S_i| \leq 2n/r$, for every $1 \leq i \leq r$. Matoušek [16] showed the following important result on the construction of fine simplicial partitions with low crossing number:

Theorem 1 ([16]). *Let S be a set of n points in the plane, and let r be an integer with $1 \leq r \leq n/2$. Then a fine simplicial partition $\Psi(S)$ of size r with crossing number $O(\sqrt{r})$ exists. If r is constant, $\Psi(S)$ can be constructed in $O(n)$ time and space.*

Simplicial partitions are the basis of an efficient search data structure, called *partition tree*. The root of a partition tree of S has r children $v_1, \ldots, v_r$ that correspond one-to-one to the sets S_i in $\Psi(S)$. Each child v_i is the root of a recursively defined partition tree of S_i. The partition tree of n points can be computed in $O(n \log n)$ time and uses $O(n)$ space [16].

To solve the problem FP-halfplane we will take advantage of the *farthest-point Voronoi diagram*. Given a set of n sites in the plane, the farthest-point Voronoi diagram is a partition of the plane into cells, each of which is associated with a site and contains all the points in the plane that are *farther* from that site than from any other site. Unlike the *nearest-point* Voronoi diagram, in the farthest-point Voronoi diagram only the sites on the convex hull have a non-empty Voronoi region associated with them. In the plane, the farthest-point Voronoi diagram can be constructed in $O(n \log n)$ time if the sites are in general position [20]. When the sites are the vertices of a convex polygon, the diagram can be constructed [2] and preprocessed for planar point-location queries [12] in linear time.

We start by constructing the partition tree of S. Recall that for a node v_i of the tree, S_i is the subset of S stored at v_i, and t_i is the triangle of $\Psi(S)$ that contains S_i. Let $n_i = |S_i|$. For each node v_i we compute and store the

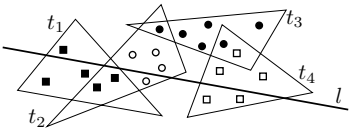


Fig. 5. A simplicial partition

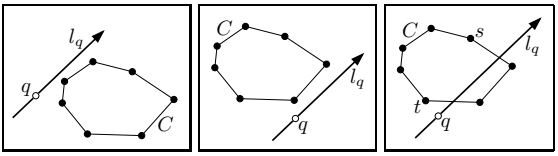


Fig. 6. Cases for the intersection of l_q with C

farthest-point Voronoi diagram of S_i and preprocess it for planar point-location queries. This takes $\tau(n_i) = O(n_i \log n_i)$ time and uses $\sigma(n_i) = O(n_i)$ space. Let $T(n)$ and $S(n)$ be the total construction time and space consumption of these secondary data structures, respectively. They satisfy the following recurrences: $T(n) \leq \tau(n) + \sum_{i=1}^{r} T(n_i)$ and $S(n) = \sigma(n) + r + \sum_{i=1}^{r} S(n_i)$ for $n > 1$, and $T(1) = S(1) = 1$. Since we have that $\sum_{i=1}^{r} n_i = n$, that $n_i \leq 2n/r$, and that r is a constant, the general version of the Master theorem [10] yields that $T(n) = O(n \log^2 n)$.

When we query the partition tree, we want to find the point in S farthest from the query point q that is left of the directed line l_q. We have to consider two different kinds of point sets S_i. First we consider the $O(\sqrt{r})$ point sets S_i with $t_i \cap l_q \neq \emptyset$. For each such point set S_i, we recursively search in its simplicial partition $\Psi(S_i)$. Second we have to consider those point sets S_i that lie left of the line l_q. For each of these at most $r - O(\sqrt{r})$ point sets, we locate the query point q in the farthest-point Voronoi diagram to find the point farthest from q. Point location takes time logarithmic in the size of the partition. Therefore, we get the following recurrence for the query time: $Q(1) = 1$ and for $n > 1$

$$Q(n) \quad \leq \quad r + \sum_{t_i \cap l_q = \emptyset} O(\log n_i) + \sum_{t_i \cap l_q \neq \emptyset} Q(n_i). \tag{1}$$

Let $c\sqrt{r} = O(\sqrt{r})$ be the crossing number of $\Psi(S)$. Given an arbitrary $\varepsilon > 0$, we can set $r = \lceil 2(c\sqrt{2})^{1/\varepsilon} \rceil$, which makes r a constant and yields $Q(n) = O(n^{1/2+\varepsilon})$ for n large enough, i.e. $n \geq 2r$. This can be seen by bounding the first sum in Inequality 1 by $O(r \log n)$ and the second sum by $c\sqrt{r} \cdot Q(2n/r)$. We sum up:

Theorem 2. *There is a data structure for FP-halfplane that answers queries in $O(n^{1/2+\varepsilon})$ time given $O(n \log^2 n)$ preprocessing time and $O(n \log n)$ space.*

4 Farthest Vertex in Halfplane

We now tackle FV-halfplane, the convex case of FP-halfplane. It is the basis of our solutions for the problems FP-halfstrip and FIP-halfstrip. The problem is to preprocess a convex n-gon C such that for a query pair (q, l_q), where q is a point and l_q is a directed line through q, one can efficiently decide whether there is a vertex of C left of l_q and if yes, report the one farthest from q.

Given a query pair (q, l_q), we first compute potential intersection points of l_q with the boundary ∂C of C. This can be done by binary search in $O(\log n)$ time since the distance from l_q is a unimodal function on ∂C. There are three possible cases, see Figure 6: (a) $l_q \cap C = \emptyset$ and C lies to the right of l_q; (b) $l_q \cap C = \emptyset$ and C lies to the left of l_q; (c) l_q has nonempty intersection with C. Knowing that $l_q \cap C = \emptyset$, case (a) can be handled in constant time. Case (b) reduces to finding the point on C farthest from l_q. This can be achieved in $O(\log n)$ time by locating the query point in the farthest-point Voronoi diagram of the vertices of C. In the remainder of this section we show how to handle case (c).

In the preprocessing phase, we construct a balanced binary tree T in $O(n \log n)$ time as follows. The vertices of the convex polygon C, in counter-clockwise order from the rightmost vertex, are associated with the leaves of T. At each internal node u, we compute and store the farthest-point Voronoi diagram V_u of the leaf descendants of u. This takes linear time for each level of T since all point sets are in convex position [2]. Within the same asymptotic time bound we then preprocess V_u for planar point-location queries [12]. Thus the computation of T takes $O(n \log n)$ time in total.

We query T as follows. Consider the edges of C intersected by l_q. If these edges are incident to the same vertex v of C to the left of l_q then we report v. Otherwise the edges have two different endpoints to the left of l_q. Let s be the first and t the second endpoint in counter-clockwise order on C, see Figure 6. We assume that the sequence of points on C that lie to the left of l_q does not contain both the rightmost vertex and its counter-clockwise predecessor. Otherwise the words left and right in the following description have to be exchanged.

We walk in T from s to t and collect a set $\mathcal{V}$ of $O(\log n)$ farthest-point Voronoi diagrams in two phases. In the ascending phase we go upwards from s until we reach the least common ancestor a of s and t. Whenever we get to a node $u \neq a$ from its *left* child, we add to $\mathcal{V}$ the Voronoi diagram stored at the right child of u. In the descending phase we go down from a towards t. Whenever we go to the *right* child of a node $u \neq a$, we add to $\mathcal{V}$ the Voronoi diagram stored at the left child of u. Clearly, all points associated with these Voronoi diagrams are to the left of l_q and thus the sought vertex is either s, t or one of these points. We locate q in $O(\log n)$ time in each farthest-point Voronoi diagram in $\mathcal{V}$ and keep track of the point farthest from q. This answers a query in $O(\log^2 n)$ time.

Theorem 3. *There is a data structure for FV-halfplane that answers queries in $O(\log^2 n)$ time given $O(n \log n)$ space and preprocessing time.*

5 Farthest Point in Halfstrip

In this section we want to preprocess a set S of n points for queries of the following type. Given a triplet (q, l_q, Δ), where q is a point and l_q is a directed line through q such that all points in S are within distance Δ from l_q, decide whether there is a point $p \in S$ such that (i) $|qp| \geq \Delta$, and (ii) the projection of p on l_q lies before q. If yes, report the point farthest from q that fulfills (i) and (ii). (See Figure 2.)

FP-halfstrip can be solved by the same approach as for FP-halfplane: construct a partition tree based on a fine simplicial partition in $O(n^{1+\varepsilon})$ time [16] and enhance it with a second-level data structure. For the points at each internal node of the partition tree, the second-level structure consists of the farthest-point Voronoi diagram preprocessed for planar point location.

We would prefer to use the faster solution for FV-halfplane, i.e. for the convex case. At first glance it seems that this is not possible, since among the points that fulfill condition (ii), the point p' farthest from the query point q may lie *inside* the convex hull C of S, see Figure 3. Condition (i), however, does in fact give us a way to use the data structure for FV-halfplane to solve FP-halfstrip. For a point q and a directed line l_q with $q \in l_q$ let l'_q be the directed line that results from turning l_q around q by $+90°$. Then the points whose projection on l_q lies before q are exactly the points to the left of l'_q.

Lemma 1. *Given a set $S \subset \mathbb{R}^2$ and a triplet (q, l_q, Δ), where q is a point and l_q is a directed line through q such that all points in S are within distance Δ from l_q, if there is a point $p \in S$ such that (i) $|qp| \geq \Delta$, and (ii) p lies to the left of l'_q, then among all points in S to the left of l'_q the point farthest from q is a vertex of the convex hull C of S.*

Proof. Let Σ be the closed strip that is bounded by the two lines at distance Δ from l_q and let H be the part of S to the right of l'_q. In Figure 7, Σ is the whole shaded area, H is the darker part. Let p be the point farthest from q to the left of l'_q, let D be a disk centered at q that touches p, and let $D' = D \cap \Sigma$. In Figure 7, the boundary of D is dotted, that of D' is bold solid. Finally let $U = D' \cup H$. Then p lies on the boundary of U. If $|pq| \geq \Delta$, U is convex. Thus for any (finite) set F with $p \in F \subset U$ it holds that p is a vertex of the convex hull of F. □

Since the convex hull of S can be computed in $O(n \log n)$ time, we have:

Theorem 4. *There is a data structure for FP-halfstrip that answers queries in $O(\log^2 n)$ time given $O(n \log n)$ space and preprocessing time.*

This yields our first main result, a data structure for finding the point farthest from a query segment.

Theorem 5. *Given a set S of n points, we can construct in $O(n \log n)$ space and preprocessing time a data structure that for any line segment s determines in $O(\log^2 n)$ time the point in S farthest from s.*

Proof. Let $s = \overline{uv}$ and let $\ell = uv$ be the line that is directed from u to v. There are two mutually exclusive cases. In the first case the point farthest from s is also the point farthest from ℓ. For this case we preprocess S by computing in $O(n \log n)$ time the convex hull C of S. Then this case can be solved by binary search in $O(\log n)$ time since the distance from ℓ is unimodal on C. Note that the point farthest from ℓ also gives us the smallest value Δ such that S lies within a Δ-strip around ℓ. For the second case, let S_w ($w \in \{u, v\}$) be the set of all

points in S that are separated from s by the line orthogonal to s in w. In this case the point farthest from s is the point in S_u farthest from u or the point in S_v farthest from v. These two points can be determined within the desired time and space bounds by querying a data structure for FP-halfstrip with the triplets (u, uv, Δ) and (v, vu, Δ). □

6 Farthest Indexed Point in Halfstrip

We solve FIP-halfstrip, the indexed version of FP-halfstrip, in a way similar to FV-halfplane. At the same time we use the data structure for FV-halfplane as a plug-in. Let the points in the input sequence S be denoted by $p_1, \ldots, p_n$. In the preprocessing phase we construct a balanced binary tree $\mathcal{T}$ of the same structure as for FV-halfplane. The i-th leaf of $\mathcal{T}$ is associated with the point $p_i \in S$. We build the tree $\mathcal{T}$ bottom-up. At each internal node v, we compute and store the convex hull C_v of the leaf descendants $p_{i(v)}, \ldots, p_{j(v)}$ of v. We also compute and store at v a secondary level data structure, namely the tree described in Section 4 that solves FV-halfplane (i.e. FP-halfstrip) for the vertices of C_v. The overall computation of $\mathcal{T}$ requires $O(n \log^2 n)$ time and space.

A query is also very similar to FV-halfplane: for a query (i, j, Δ), we follow the unique path from p_i to p_j in $\mathcal{T}$ collecting a set $\mathcal{C}$ of $O(\log n)$ convex hulls whose union contains all points p_k with $i < k < j$. This is done in the same way as with the set of farthest-point Voronoi diagrams in Section 4. For each convex hull $C_v \in \mathcal{C}$, we solve FP-halfstrip for the triplet $(p_i, p_i p_j, \Delta)$ using the secondary data structure stored at vertex v of $\mathcal{T}$. (Compare the situations in Figures 2 and 4!) Thus we can decide in $O(\log^2 |C_v|)$ time whether there is a k, $i(v) \leq k \leq j(v)$, such that the point p_k satisfies the two FIP-halfstrip conditions. Since the size of the set $\mathcal{C}$ is $O(\log n)$, the overall query time is $O(\log^3 n)$.

Theorem 6. *There is a data structure for FIP-halfstrip that answers queries in $O(\log^3 n)$ time given $O(n \log^2 n)$ space and preprocessing time.*

7 Polygonal Path Simplification and FIP-Halfstrip

In this section we use our solution of FIP-halfstrip to extend a recent result of Daescu and Mi [11] for the min-# version of the polygonal path simplification problem: Given a polygonal path $P = (p_1, p_2, \ldots, p_n)$, with n vertices, and an error tolerance Δ, find a subpath $P' = (p_{i_1} = p_1, p_{i_2}, \ldots, p_{i_m} = p_n)$ of P such that the vertices of P' are an ordered subset of the vertices of P, each line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P' is a Δ-approximation of the corresponding subpath $(p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P, and the number of vertices m of P' is minimized.

To decide whether a line segment of P' is a Δ-approximation of the corresponding subpath of P, two error criteria are commonly used: the *tolerance-zone* criterion and the *infinite-beam* criterion. The first criterion produces a compressed version that better captures the features of the original path, while the

second gives a better degree of compression. According to the tolerance-zone criterion, all vertices of the approximated subpath of P must be within distance Δ from the approximating line segment of P', while according to the infinite-beam criterion all vertices of the approximated subpath must be within distance Δ from the line supporting the approximating line segment.

In [11] an output sensitive, query based algorithm is presented for solving the min-# problem according to the infinite-beam criterion. There, it is also shown that the algorithm is very fast in practice and outperforms previous algorithms. Since a Δ-approximation segment according to the tolerance-zone criterion is also a Δ-approximation segment according to the infinite-beam criterion, extending the algorithm in [11] to the tolerance-zone criterion reduces to answering queries on indexed points, as formulated in problem FIP-halfstrip. More precisely, if the line segment $\overline{p_{i_j}p_{i_{j+1}}}$ of P', for $j \in \{1, \ldots, m_{\text{tz}} - 1\}$, approximates the subpath $P[i_j, i_{j+1}] = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P according to the infinite-beam criterion, then all the vertices of the subpath $P[i_j, i_{j+1}]$ are within distance Δ from the line $p_{i_j}p_{i_{j+1}}$. Let l_j and l'_j denote the lines orthogonal to $p_{i_j}p_{i_{j+1}}$ in p_{i_j} and $p_{i_{j+1}}$, respectively. If for each vertex p_k of P, $i_j \leq k \leq i_{j+1}$, with the property that l_j separates p_k and $p_{i_{j+1}}$ or l'_j separates p_k and p_{i_j}, we have that p_k is within distance Δ from p_{i_j} or $p_{i_{j+1}}$, respectively, then every vertex on the subpath $P[i_j, i_{j+1}]$ is within distance Δ from the line segment $\overline{p_{i_j}p_{i_{j+1}}}$. Clearly, this reduces to solving FIP-halfstrip, see Section 6. The result is an output sensitive, query-based algorithm for solving the min-# problem under the tolerance-zone criterion.

Theorem 7. *Given a polygonal path $P = (p_1, p_2, \ldots, p_n)$ in the plane, the min-# problem under the tolerance-zone criterion can be solved in $O(F_{\text{tz}}(m_{\text{tz}})\ n \log^3 n)$ time using $O(n \log^2 n)$ space, where $F_{\text{tz}}(m_{\text{tz}}) \leq n$ is the number of vertices that can be reached from p_1 with at most $(m_{\text{tz}} - 2)$ Δ-approximating segments, and m_{tz} is the number of vertices on an optimal approximating path.*

Proof. The algorithm is similar to the query-based algorithm in [11], except that now each query takes $O(\log n + \log^3 n)$ time instead of $O(\log n)$ time: as in [11] we first spend $O(\log n)$ time to decide whether some segment $\overline{p_ip_j}$ is a Δ-approximating segment according to the infinite-beam criterion. If the answer is positive, we now use Theorem 6 and spend additional $O(\log^3 n)$ time to decide whether $\overline{p_ip_j}$ is a Δ-approximating segment according to the tolerance-zone criterion. □

8 Batched Farthest Indexed Point in Halfplane

In this section we consider the problem BFIP-halfplane: given a sequence $S = (p_1, \ldots, p_n)$ of points and a point $p \notin S$, decide for each $i \in \{1, \ldots, n\}$ whether there is a point $p_f \in \{p_1, \ldots, p_i\}$ that lies on the same side as p with respect to the perpendicular bisector of p and p_i. If yes, report the point p_f farthest from p that has the above property.

A version of BFIP-halfplane without the index restriction has been considered in [14]. There the problem of computing the minimum-sum dipolar spanning tree (MSST) is considered. The MSST of a point set S is a tree with vertex set S and two non-leaf nodes $x, y \in S$ that minimizes $|xy| + \max\{r_x, r_y\}$, where r_x and r_y are the radii of two disks centered at x and y whose union covers S. In the computation of the MSST the following subproblem shows up: report for each point $p_i \in S$ a point farthest from the fixed point $p \notin S$ that lies on the same side as p with respect to the perpendicular bisector of p and p_i. In [14] the problem is reduced to the problem of finding for each $p_i \in S$ the first disk in a sequence of disks that does *not* contain p_i. This problem has been addressed in [7] under the name *off-line ball exclusion search (OLBES)*. The authors set up a tree data structure with a space requirement of $O(n \log n)$ and then query this structure with each point in S. This results in a total running time of $O(n \log n)$ for OLBES in the plane. In [14], a version of OLBES where all disks intersect a common point is solved in $O(n \log n)$ time and $O(n)$ space by sweeping an arrangement of circular arcs. The same problem is solved in [13] in $O(n \log n)$ time and space using a tree data structure and fractional cascading.

We are interested in the problem BFIP-halfplane since it adds a time component to the pure query problem FIP-halfplane. We want to keep track on how the point farthest from the fixed point p changes over time while we insert the points of S one after the other and ignore all those points that lie in a halfplane determined by the newly inserted point. We solve this problem by setting up a tree data structure similar to that in the proof of Lemma 2 in [13]. Here, however, we must solve a different OLBES problem in each query and thus need to modify our tree successively.

We need some notation. Let $D(x, p)$ be the open disk centered at x that touches p and let $h(p, q)$ be the closed halfplane that contains p and whose boundary is the perpendicular bisector of p and q. Note that $x \in h(p, q)$ is equivalent to $|xp| \leq |xq|$, which in turn is equivalent to $q \notin D(x, p)$.

Now we give our algorithm for BFIP-halfplane. Let $p_1, \ldots, p_n$ be the sequence of input points. We first sort the n disks $D(p_i, p)$ in order of non-increasing radius. Let $D_1, \ldots, D_n$ be the resulting sequence. Thus, a disk D_k in this sequence corresponds to some disk $D(p_i, p)$ and in general $k \neq i$. Let $D_{n+1} = \emptyset$. We build a binary tree T as follows. The leaves of T correspond to the sequence $D_1, \ldots, D_{n+1}$, from left to right. Each inner node v stores the intersection I_v of the leaf descendants of v (excluding D_{n+1}). We label each node v with a pair $[a_v, b_v]$ encoding the set $S_v = \{a_v, \ldots, b_v\}$ of consecutive indices that correspond to the disks associated with the leaves of the subtree of T rooted at v. In Figure 8 a tree with $n = 13$ is depicted. We build T in a bottom-up fashion. Each inner node has two children in the previous level, except possibly a level's rightmost node, which can have a right child in an earlier level, see the node with label $[9, 13]$ in Figure 8. We query T with the points in S, and the answer of a query will correspond to the index of the first disk in the sequence $D_1, \ldots, D_{n+1}$ that does *not* contain the query point.

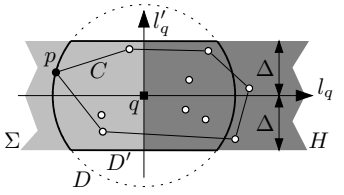


Fig. 7. Proof of Lemma 1

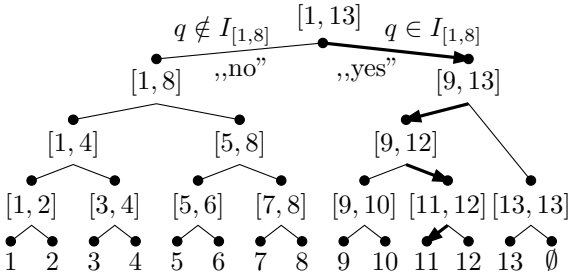


Fig. 8. The tree T for $n = 13$. Bold arrows indicate the search path for a point $q \in (D_1 \cap \cdots \cap D_{10}) \setminus D_{11}$

Unlike [7, 13] we start with an empty skeleton of T, i.e. all inner nodes v are labeled by $[a_v, b_v]$, but all leaves and all intersections I_v are set to $\mathbb{R}^2$. The order in which we query becomes crucial. We go through the points $p_1, \ldots, p_n \in S$ in order of increasing index. When we query with the point p_i, only the disks $D(p_1, p), \ldots, D(p_{i-1}, p)$ have been inserted in T. Before querying T with p_i we update T by adding the new disk $D_k = D(p_i, p)$ (recall that usually $k \neq i$) to the intersection I_v for each node v on the path from the root to the leaf that corresponds to D_k. Querying T with p_i amounts to following a path from the root to a leaf. In each inner node v with left child ℓ, the test $p_i \in I_\ell$ is performed. If $p_i \in I_\ell$, the query continues with the right, otherwise with the left child of v, see Figure 8. The leaf at the end of the query path π determines what our algorithm reports. Let D_j be the disk corresponding to that leaf. If $j \leq n$, then we report that p_j is the point farthest from p in $\{p_1, \ldots, p_i\} \cap h(p, p_i)$. Otherwise (i.e. if $j = n + 1$) we report that $\{p_1, \ldots, p_i\} \cap h(p, p_i)$ is empty. This algorithm yields the following.

Theorem 8. *Given a sequence S of n points and a point $p \notin S$, BFIP-halfplane can be solved in $O(n \log^2 n)$ time and $O(n \log n)$ space.*

Proof. We first show the correctness of the above algorithm. Depending on the index of the disk D_j we consider two cases. The first case is that $j = n + 1$. Then π is the rightmost root–leaf path. Consider the left children of the nodes on π. The sets S_ℓ that belong to these left children partition $\{1, \ldots, n\}$. In other words, the intersection of I_ℓ over these children is $D_1 \cap \cdots \cap D_n$. Since π is the rightmost root–leaf path, the containment queries in all nodes on π were answered positively. Thus p_i is contained in all disks currently in T, i.e. $p_i \in D(p_1, p) \cap \cdots \cap D(p_i, p)$. This means that none of $p_1, \ldots, p_i$ lies in $h(p, p_i)$. Otherwise Lemma 1 in [14] would guarantee that $p_i \notin D(p_k, p)$ for the point $p_k \in \{p_1, \ldots, p_i\}$ farthest from p in $h(p, p_i)$.

The second case is $j \leq n$. Again we consider the left children of the nodes on the query path π of p_i. The sets S_ℓ partition $\{1, \ldots, j-1\}$ if we take only those left children ℓ into account that do not themselves lie on π. Similarly to above, the intersection of I_ℓ over these children is $D_1 \cap \cdots \cap D_{j-1}$. Thus, p_i

is contained in all D_k with $k < j$ that are currently in T. On the other hand, since π is not the rightmost root–leaf path, π contains at least one node that is a left child of a node on π. The last such left child v is the root of the subtree whose rightmost leaf corresponds to D_j. Thus v is associated with some set $S_v = \{i_v, \ldots, j\}$, where $1 \leq i_v \leq j$. Since we have already observed that p_i is contained in all D_k with $k < j$ that are currently in T, but π came to v via a "no"-branch ($p_i \notin I_v$), we now know that $p_i \notin D_j$. Let m be such that $D_j = D(p_m, p)$. Note that $p_i \notin D(p_m, p)$ means that $D(p_m, p)$ was inserted in T before querying with p_i, and thus $m \leq i$. Since $p_i \notin D(p_m, p)$, and $p_i \in D(p_r, p)$ for all $r \leq i$ with $|pp_r| > |pp_m|$, Lemma 1 in [14] yields that p_m is farthest from p in $\{p_1, \ldots, p_i\} \cap h(p, p_i)$.

The total running time is $O(n \log^2 n)$ since both querying and updating T take $O(\log^2 n)$ time for each p_i. The space consumption is $O(n \log n)$ since each disk contributes at most one arc to each intersection stored on the path from the root to "its" leaf. For details refer to the full version of this paper. □

The definition of BFIP-halfplane and Theorem 8 can be generalized without much effort as follows. Instead of insisting that the separator of p and p_i splits $\overline{pp_i}$ in a ratio of 1:1, any other ratio can be used as long as the split is orthogonal.

9 Conclusions

We have presented solutions to some very basic farthest-point problems and have shown how they can be used to solve other, more complex problems efficiently, such as simplifying polygonal paths or determining the point farthest from a query segment. Both these problems can be solved more efficiently if the solution of the underlying problem FV-halfplane can be improved. It is possible to reduce the query time to $O(\log n)$, e.g. by storing the farthest-point Voronoi diagrams of all $\Theta(n^2)$ subsets of vertices that are consecutive on the given convex polygon. However, it seems hard to achieve the same improvement under the condition that space consumption and preprocessing time remain in $O(n \log n)$.

Acknowledgments

We thank Raimund Seidel for pointing us to the work of Edelsbrunner et al. [12], which helped us to reduce the preprocessing time of our data structure for FV-halfplane. We thank Raghavan Dhandapani for pointing out the alternative solution for FV-halfplane with $O(\log n)$ query time and $O(n^3)$ preprocessing time. We finally wish to thank the DIMACS center where this research was started.

References

[1] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1(4):189–201, 1992.

[2] A. Aggarwal, L. J. Guibas, J. B. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, 1989.
[3] A. Aggarwal and D. Kravets. A linear time algorithm for finding all farthest neighbors in a convex polygon. *Information Processing letters*, 31(1):17–20, 1989.
[4] S. Bespamyatnikh. Computing closest points for segments. *Int. J. Comput. Geom. Appl.*, 13(5):419–438, 2003.
[5] S. Bespamyatnikh and J. Snoeyink. Queries with segments in Voronoi diagrams. *Comput. Geom. Theory Appl.*, 16(1):23–33, 2000.
[6] D. Z. Chen and O. Daescu. Space-efficient algorithms for approximating polygonal curves in two dimensional space. *International Journal of Computational Geometry and Applications*, 13(2):95–112, 2003.
[7] D. Z. Chen, O. Daescu, J. Hershberger, P. M. Kogge, and J. Snoeyink. Polygonal path approximation with angle constraints. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'01)*, pp. 342–343, 2001.
[8] O. Cheong, C.-S. Shin, and A. Vigneron. Computing farthest neighbors on a convex polytope. *Theoretical Computer Science*, 296:47–58, 2003.
[9] R. Cole and C.-K. Yap. Geometric retrieval problems. In *Proc. 24th Ann. IEEE Symposium on Foundations of Computer Science (FOCS'83)*, pp. 112–121, 1983.
[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.
[11] O. Daescu and N. Mi. Polygonal path approximation: A query based approach. *Computational Geometry: Theory and Applications*, 30(1):41–58, 2005.
[12] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.
[13] J. Gudmundsson, H. Haverkort, S.-M. Park, C.-S. Shin, and A. Wolff. Facility location and the geometric minimum-diameter spanning tree. In *Proc. 5th Int. Workshop on Approx. Algorithms for Combinatorial Optimization (APPROX'02)*, vol. 2462 of *Lecture Notes Comput. Sci.*, pp. 146–160. Springer-Verlag, 2002.
[14] J. Gudmundsson, H. Haverkort, S.-M. Park, C.-S. Shin, and A. Wolff. Facility location and the geometric minimum-diameter spanning tree. *Computational Geometry: Theory and Applications*, 27(1):87–106, 2004.
[15] D. E. Knuth. *The Art of Computer Programming*, volume 3, chapter Sorting and Searching. Addison-Wesley, Reading, MA, 1973.
[16] J. Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8:315–334, 1992.
[17] J. S. B. Mitchell and J. O'Rourke. Computational geometry column 42. *SIGACT News*, 32(3):63–72, 2001.
[18] P. Mitra and B. Chaudhuri. Efficiently computing the closest point to a query line. *Pattern Recognition Letters*, 19(11):1027–1035, 1998.
[19] A. Mukhopadhyay. Using simplicial partitions to determine a closest point to a query line. In *Proc. Canadian Conf. Comp. Geom. (CCCG'02)*, pp. 10–12, 2002.
[20] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1990.
[21] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete and Computational Geometry*, 4:101–115, 1989.

Grid Vertex-Unfolding Orthostacks

Erik D. Demaine[1,⋆], John Iacono[2,⋆⋆], and Stefan Langerman[3,⋆⋆⋆]

[1] MIT Computer Science and Artificial Intelligence Laboratory,
32 Vassar St., Cambridge, MA 02139, USA
edemaine@mit.edu
[2] Department of Computer and Information Science, Polytechnic University,
5 MetroTech Center, Brooklyn, NY 11201, USA
http://john.poly.edu
[3] Département d'informatique, Université Libre de Bruxelles,
ULB CP212, 1050 Brussels, Belgium
Stefan.Langerman@ulb.ac.be

Abstract. An algorithm was presented in [BDD+98] for unfolding orthostacks into one piece without overlap by using arbitrary cuts along the surface. It was conjectured that orthostacks could be unfolded using cuts that lie in a plane orthogonal to a coordinate axis and containing a vertex of the orthostack. We prove the existence of a vertex-unfolding using only such cuts.

1 Introduction

A long-standing open question is whether every convex polyhedron can be *edge unfolded*—cut along some of its edges and unfolded into a single planar piece without overlap [She75, O'R98, Dem00, DO05]. A related open question asks whether every polyhedron without boundary[1] (not necessarily convex but forming a closed surface) can be *generally unfolded*—cut along its surface (not just along edges) and unfolded into a single planar piece without overlap. Biedl et al. [BDD+98] made partial progress on both of these problems in the context of *orthostacks*. An orthostack is an orthogonal polyhedron for which every horizontal planar slice is connected, and for which the interior of the polyhedron is a connected solid. Thus, every horizontal planar slice of an orthostacks's interior is a simple polygon. Biedl et al. showed that not all orthostacks can be edge unfolded (see Figure 1), but that all orthostacks can be generally unfolded. In

⋆ Research supported in part by NSF grants CCF-0347776, OISE-0334653, and CCF-0430849, and by DOE grant DE-FG02-04ER25647.
⋆⋆ Research supported in part by NSF grants OISE-0334653 and CCF-0430849.
⋆⋆⋆ Chercheur qualifié du FNRS.

[1] For the purposes of this problem, a *polyhedron without boundary* is an abstract polyhedral complex without boundary, i.e., a set of polygons and a definition of incidence between polygons such that every edge is incident to exactly two polygons and every two polygons meet at either a common vertex, a common edge, or not at all. Note that a polyhedron is treated as a surface throughout this paper.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 76–82, 2005.

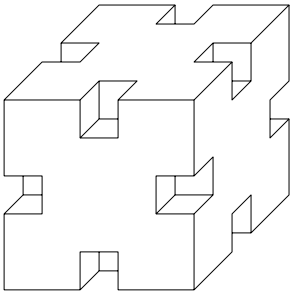


Fig. 1. This orthostack is not edge-unfoldable [BDD+98]

their general unfoldings, all cuts are parallel to coordinate axes, but many of the cuts do not lie in coordinate planes that contain polyhedron vertices. Given the lack of pure edge unfoldings, the closest analog we can hope for with (nonconvex) orthostacks is to find *grid unfoldings* in which every cut is in a coordinate plane that contains a polyhedron vertex. In other words, a grid unfolding is an edge unfolding of the refined ("gridded") polyhedron in which we slice along every coordinate plane containing a polyhedron vertex. Biedl et al. [BDD+98] asked whether all orthostacks can be grid unfolded.

We make partial progress on this problem by showing that every orthostack can be *grid vertex-unfolded*, i.e., cut along some of the grid lines and unfolded into a vertex-connected planar piece without overlap. Vertex-unfoldings were introduced in [DEE+02, DEE+03]; the difference from edge unfoldings is that faces can remain connected along single points (vertices) instead of having to be connected along whole edges.

2 A Grid Vertex Unfolding

Given an orthostack K, let $z_0 < z_1 < \cdots < z_n$ be the distinct z coordinates of vertices of K. Subdivide the faces of K by cutting along every plane perpendicular to a coordinate axis that passes through a vertex of K. This subdivision *rectangulates* K. We use the term *rectangle* to refer to one element of this facial subdivision, while *face* refers to a maximal connected set of coplanar rectangles. We use *up* and *down* to refer to the z dimension, and use *left* and *right* to refer to the x dimension.

2.1 Rectangle Categorization

We partition the rectangles of K into several categories. After this categorization, the description of the unfolding layout is not difficult.

For $i = 0, 1, \ldots, n-1$, define the *i-band* to be the set of vertical rectangles (i.e., that lie in an xz plane or in a yz plane) whose z coordinates are between z_i and z_{i+1}. Each i-band is connected, and all of the rectangles of an i-band have the same extent in the z dimension, namely, $[z_i, z_{i+1}]$.

For $i = 0, 1, \ldots, n$, we define the *i-faces* to be the faces of K in the horizontal plane $z = z_i$. As we have defined them, an i-face has several properties. It may have the interior of K above or below it (but not both). The perimeter of the i-face has a nonempty intersection with the $(i-1)$-band, provided $i > 0$, and with the i-band, provided $i < n$. (If an i-face does not meet the $(i-1)$-band, it must be the bottom face of the polyhedron, and if it did not meet the i-band, it must be the top face of the polyhedron.) By the definition of an orthostack, the intersection between an i-face and each incident band is connected. That is, the perimeter of the i-face can be cut into two contiguous intervals such that each interval intersects solely the $(i-1)$-band or the i-band.

Also needed are the notions of the "begin rectangle" and "end rectangle" of the i-band. Choose the 0-*band begin rectangle* to be an arbitrary rectangle of the 0-band. For $i \geq 0$, define the *i-band end rectangle* to be the rectangle of the i-band that is adjacent to the i-band begin rectangle in the counter-clockwise direction as viewed from $+z$. (Thus, the begin and end rectangles of the i-band are adjacent.) For $i \geq 1$, define the *i-connecting face* to be the i-face that shares an edge with the $(i-1)$-band end rectangle, if such a face exists. For $i \geq 1$, define the *i-band begin rectangle* to be one of the rectangles of the i-band that shares an edge with the i-connecting face, if it exists, or else the rectangle of the i-band that shares an edge with the $(i-1)$-band end rectangle. The *i-band interior rectangles* are rectangles of the i-band that are neither the begin rectangle nor the end rectangle.

Define the *i-connecting sequence* to be an edge-connected sequence of rectangles in the i-connecting face, if it exists, starting at the rectangle that shares an edge with the $(i-1)$-band end rectangle and ending at the rectangle that shares an edge with the i-band begin rectangle. This sequence is chosen to contain the fewest rectangles possible (a shortest path in the dual graph on the rectangles in the i-connecting face), in order to prevent the path from looping around an island and thereby isolating interior portions of the i-face. If the i-connecting face does not exist, the i-connecting sequence is the empty sequence. The rectangles in the i-connecting sequence are called *i-connecting rectangles*; all other rectangles of the i-face are called *normal rectangles*.

We now merge all normal rectangles with their normal neighbors in the x dimension. Call the resultant rectangular regions *über-rectangles*. Thus i-faces are partitioned into the i-connecting rectangles and the i-über-rectangles. All i-über-rectangles are connected to the perimeter of the i-face, and thus are edge-connected to the $(i-1)$-band or the i-band. Define an *i-up-über-rectangle* to be an über-rectangle that is incident to the i-band and an *i-down-über-rectangle* to be an über-rectangle that is incident to the $(i-1)$-band. If an über-rectangle is incident to both, we classify it arbitrarily.

Thus we have partitioned K into i-band begin rectangles, i-band end rectangles, i-band interior rectangles, i-up-über-rectangles, i-down-über-rectangles, and i-connecting rectangles. We now proceed to a description of the unfolding.

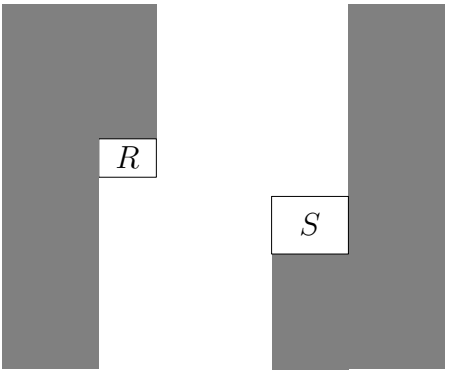


Fig. 2. If R and S are anchors of an anchored component, the component may be unfolded only in the unshaded region

2.2 Unfolding Algorithm

Our unfolding of an orthostack consists of several components strung together at distinguished rectangles called *anchors*. Specifically, there are two types of components, i-main components and i-connecting components, both of which are anchored at two rectangles, a begin rectangle and an end rectangle. The *i-main component* consists of the entire i-band (the i-band begin rectangle, the i-band end rectangle, and the i-band interior rectangles), the $(i+1)$-down-über-rectangles, and the i-up-über-rectangles. The *i-connecting component* consists of the $(i-1)$-band end rectangle, the i-connecting rectangles (if any), and the i-band begin rectangle. It serves to connect the $(i-1)$-main component and the i-main component (at the $(i-1)$-band end rectangle and the i-band begin rectangle, respectively).

To ensure that components do not overlap each other, we enforce that the components are *anchored* in the following sense. A component is anchored at anchor rectangles R and S if, in the unfolded layout of the component, no rectangles are in the shaded region of Figure 2. More precisely, every rectangle is strictly right of R and left of S, or directly below R, or directly above S.

We can combine two anchored components with a common anchor while avoiding overlap. More precisely, given a component C anchored at anchors R and S, and another component C' anchored at S and T, we can combine the two unfolded layouts by rigidly moving C' so that the two copies of S coincide (with matching orientations). The conditions on the rectangles in the two components C and C' guarantees nonoverlap of the combined unfolded layout.

We unfold the orthostack in such a way that the positive z direction of every vertical (i-band) rectangle is placed in the positive y direction in the planar unfolding.

We edge-unfold the i-main component by leaving one edge attached between the über-rectangles of the component (arbitrarily, if there is a choice), and cutting along all of the other edges of the über-rectangles. As shown in Figure 3, the layout induced by this edge unfolding consists of a central horizontal rect-

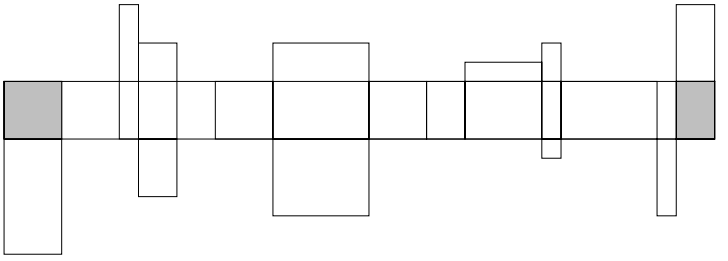


Fig. 3. An example of an unfolded i-main component. The shaded rectangles are the i-begin rectangle (right) and i-end rectangle (left). They are connected by the remainder of the i-band. Above the i-band are the $(i+1)$-down-über-rectangles and below are the i-up-über-rectangles.

angular strip, which contains all i-band rectangles, and has the $(i+1)$-down-über-rectangles connected to the top of this strip, and the i-up-über-rectangles connected to the bottom of this strip. The rightmost rectangle of this strip is the i-band begin rectangle, and the leftmost rectangle of the strip is the i-band end rectangle. There is nothing above the leftmost rectangle or below the rightmost rectangle because these vacant locations are where the connecting rectangles are attached, by definition, and we know that connecting rectangles are not über-rectangles. Therefore the edge unfolding of the i-main component is anchored at the i-band begin and end rectangles.

We vertex-unfold the i-connecting component by an incremental algorithm, by performing a sequence of vertex unfoldings, beginning with the $(i-1)$-band end rectangle. This unfolding proceeds in phases: the middle phase and the end phase.

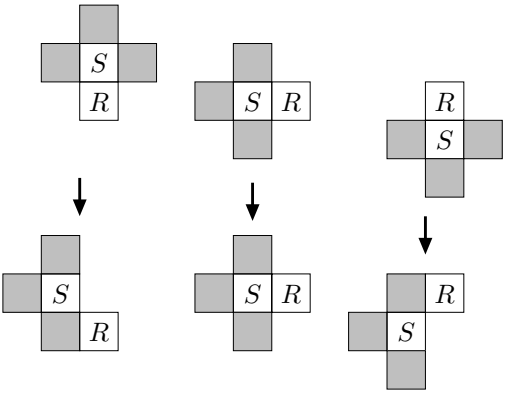


Fig. 4. How a path of rectangles can be vertex-unfolded so that each rectangle is to the left of the previous rectangle. In this diagram, R represents the current rectangle, S represents the next rectangle, and the grey shaded area represent the three possible positions of the next next rectangle. There are three cases of the possible location of S in relation to R. Note that the illustrated unfoldings work no matter what the sizes of the rectangles.

The middle phase of the vertex unfolding does all unfoldings except for the last (i.e., it is used for all unfoldings that do not involve the i-band begin rectangle). See Figure 4. This phase has the property that every rectangle is to the left of the previous rectangle. Suppose we are unfolding R and S, and the next item to be unfolded is T. Our algorithm requires as a precondition that S not be to the right of R and as a postcondition ensures that T is not to the right of S. The unfolding begins with the i-band end rectangle and the first rectangle of the i-connecting sequence. In order to place the i-band begin rectangle with the proper orientation, the first rectangle of the i-connecting sequence must be adjacent to its top edge. Thus, it satisfies the precondition for this construction. The construction has three cases, depending on whether S is above, below, or to the left of R. In the first two cases, we vertex-unfold 90° about the leftmost point of the edge conneting R and S so that S is to the left of R, while in the third case we do nothing.

The end phase is trickier because the i-begin rectangle must be oriented properly. If the i-connecting face does not exist, the i-begin rectangle is connected to the top edge of the $(i-1)$-end rectangle, and we are done. Otherwise, because of the construction in the middle phase, the i-begin rectangle may be connected to the top, left, or bottom edge of the last rectangle in the i-connecting sequence. The i-begin rectangle must be oriented so that the edge that connected it to the last i-connecting rectangle is the bottom edge when unfolded. There are three cases, illustrated in Figure 5.

Thus, the i-connecting component can always be vertex-unfolded into an anchored unfolding. Because the main component can also be vertex-unfolded into an anchored component, we conclude

Theorem 1. *Every orthostack can be grid vertex-unfolded.*

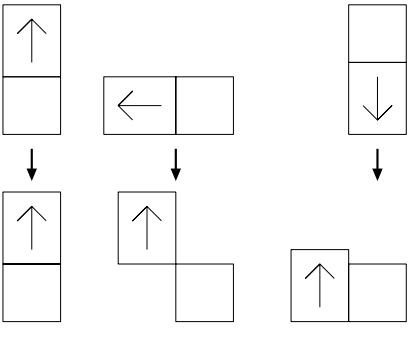


Fig. 5. How the end phase unfolds the connecting component. The empty rectangle represents the last i-connecting rectangle, and the rectangle with the arrow represents the i-begin rectangle. There are three cases for their configuration before unfolding; after unfolding the arrow must point up in order for this rectangle to be in the same orientation as in the main component. This figure shows how this is done. Note that the illustrated unfoldings work no matter what the sizes of the rectangles.

Acknowledgments

This work was initated while the authors visited McGill University's Computational Geometry Lab. We thank Mirela Damian and Joseph O'Rourke for helpful discussions. We also thank Koichi Hirata for helpful comments on the paper.

References

[BDD+98] Therese Biedl, Erik Demaine, Martin Demaine, Anna Lubiw, Mark Overmars, Joseph O'Rourke, Steve Robbins, and Sue Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, Montréal, Canada, August 1998. `http://cgm.cs.mcgill.ca/cccg98/proceedings/cccg98-biedl-unfolding.ps.gz`.

[DEE+02] Erik D. Demaine, David Eppstein, Jeff Erickson, George W. Hart, and Joseph O'Rourke. Vertex-unfolding of simplicial manifolds. In *Proceedings of the 18th Annual ACM Symposium on Computational Geometry*, pages 237–243, Barcelona, Spain, June 2002.

[DEE+03] Erik D. Demaine, David Eppstein, Jeff Erickson, George W. Hart, and Joseph O'Rourke. Vertex-unfolding of simplicial manifolds. In *Discrete Geometry: In Honor of W. Kuperberg's 60th Birthday*, pages 215–228. Marcer Dekker Inc., 2003.

[Dem00] Erik D. Demaine. Folding and unfolding linkages, paper, and polyhedra. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, volume 2098 of *Lecture Notes in Computer Science*, pages 113–124, Tokyo, Japan, November 2000.

[DO05] Erik D. Demaine and Joseph O'Rourke. A survey of folding and unfolding in computational geometry. In Jacob E. Goodman, János Pach, and Emo Welzl, editors, *Discrete and Computational Geometry*, Mathematical Sciences Research Institute Publications. Cambridge University Press, 2005. To appear.

[O'R98] Joseph O'Rourke. Folding and unfolding in computational geometry. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, volume 1763 of *Lecture Notes in Computer Science*, pages 258–266, Tokyo, Japan, December 1998.

[She75] G. C. Shephard. Convex polytopes with convex nets. *Mathematical Proceedings of the Cambridge Philosophical Society*, 78:389–403, 1975.

A Fixed Parameter Algorithm for the Minimum Number Convex Partition Problem

Magdalene Grantson and Christos Levcopoulos

Department of Computer Science,
Lund University, Box 118, 221 Lund, Sweden
{magdalene, christos}@cs.lth.se

Abstract. Given an input consisting of an n-vertex convex polygon with k hole vertices or an n-vertex planar straight line graph (PSLG) with k holes and/or reflex vertices inside the convex hull, the parameterized minimum number convex partition (MNCP) problem asks for a partition into a minimum number of convex pieces. We give a fixed-parameter tractable algorithm for this problem that runs in the following time complexities:

- linear time if k is constant,
- time polynomial in n if $k = O(\frac{\log n}{\log \log n})$, or, to be exact, in $O(n \cdot k^{6k-5} \cdot 2^{16k})$ time.

1 Introduction

A convex partition of a planar straight line graph (PSLG) G is a planar subdivision of the interior of the convex hull of G into convex polygons. A minimum number convex partition (MNCP) of G is a convex partition of G such that the number of convex polygons is minimized. It is known that the MNCP of G is NP-hard [7,5], either allowing or disallowing Steiner points. We give a fixed-parameter tractable algorithm for this problem with respect to an n-vertex convex polygon with k hole vertices [3], disallowing Steiner points. Our algorithm solves the problem in the following time complexities:

- For any constant k, it is solved in $O(n)$ time.
- For $k = O(\frac{\log n}{\log \log n})$ it is solved in time polynomial in n, or, to be exact, in $O(n \cdot k^{6k-5} \cdot 2^{16k})$ time.

Our results hold also for the more general case where the input is a PSLG and k is the total number of holes and/or reflex vertices inside the convex hull.

Known results for related problems include the following: With respect to n-vertex polygons without holes, disallowing Steiner points, Greene [4] gave an optimal algorithm which runs in $O(N^2 n^2)$ time, where N is the number of reflex vertices. Independently, Keil [5] developed an $(N^2 n \log n)$ time algorithm for the same problem. Later Keil and Snoeyink [6] improved this time bound to $O(n + N^2 \min(N^2, n))$.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 83–94, 2005.

2 An Algorithm for the MNCP Problem

Let the input of the MNCP problem be an n-vertex convex polygon P with k hole vertices. See Figure 1 for an example of a MNCP of a convex polygon with two hole vertices.

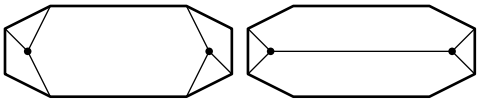


Fig. 1. Minimum weight (left) and minimum number convex partition (right) of a polygon with two hole vertices

We assume that we are given the n vertices of the perimeter $CH(P)$ of P to be $\{CH(P) = v_0, v_1, \ldots, v_{n-1}\}$ in clockwise order. The algorithm we give has preprocessing phase and a dynamic programming phase.

2.1 Preprocessing Phase of the MNCP Algorithm

We consider all possible combinations of non-crossing edges going between hole vertices, i.e., all non-crossing super-graphs on the k hole vertices. The total number of such non-crossing graphs (as we will call them for short) is at most $(2^3 \cdot 59)^k O(k^{-6})$ [2,8].

It can be shown that given a convex polygon P with hole vertices, apart from edges going between hole vertices, at most 3 p-edges incident to a hole vertex are sufficient to induce a convex partition [3], where a p-edge is an edge from a hole vertex to any vertex on the perimeter of P. (At most three p-edges suffice for each hole vertex because if a hole vertex is incident to four p-edges, there must be one that can be removed without introducing a concavity at the hole vertex. Removing it also does not introduce a concavity at the perimeter, because the polygon we consider is convex.) Therefore, for each non-crossing graph on the k hole vertices in the convex polygon, we allocate a label $i \in \{1, 2, 3\}$ to each non-convex hole vertex, which is meant to indicate how many p-edges a hole vertex would have in a convex partition if it can be constructed. There are no more than 3^k of such labellings for any non-crossing graph.

For each labelling L, we precompute the number of convex polygons N a convex partitioning of P would have if such a convex partition can be constructed from the labelling L. N is equal to the sum of the number of convex faces in the non-crossing graph plus the total number of p-edges in L (sum of the assigned labels). We then sort the labellings of all non-crossing graphs with respect to these numbers N and process them in the order of increasing N.

The idea is that, for each labelling L in the sorted list, we perform some preprocessing and thereafter run the dynamic programming algorithm given below (see Section 2.2). When we find out that a convex partition can be constructed for the current non-crossing super-graph and current labelling L we stop, since the result must be the minimum number convex partition.

To process one labelling, we allocate unique names to each hole's p-edges as follows: An edge name is a tuple (h, x), where h is a hole vertex and $x \in \{a, b, c\}$ distinguishes between the (at most) three edges the hole vertex h has.

We then consider the arrangement of lines, such that for each pair of hole vertices we have a line including them. We look at all intersections of this arrangement with the perimeter. There are $O(k^2)$ such intersections. The intersections of these lines with the perimeter partition the perimeter into $O(k^2)$ pieces. We will refer to each such piece as a "topologically homogeneous perimeter piece" (or "homogeneous piece" for short). We assume that the homogeneous pieces of the perimeter $CH(P)$ of P are $\{CH(P) = C_1, C_2, \ldots, C_\mu\}$ in clockwise order. To simplify our algorithm, if an endpoint of a homogeneous piece is a perimeter vertex, we treat the perimeter vertex as a homogeneous piece on its own. Thus a homogeneous piece is either a perimeter vertex or a piece C of the perimeter where the endpoints of C are not perimeter vertices.

We assign to each p-edge name a homogeneous piece. Since there are at most $3k$ p-edge names and each p-edge name can be assigned to each of the $O(k^2)$ homogeneous pieces, there are $O(k^{6k} \cdot 2^{3k})$ such assignments to be considered for a given labelling L.

For each homogeneous piece assignment Π to p-edge names we then take a (arbitrary) point on each homogeneous piece and connect it to all hole vertices that should have an edge going to it. Let E be the set of edges created in this way, for all homogeneous pieces. We do this to:

1. Check for possible edge intersections. (We have no intersections if and only if the set E together with all input edges do not include any intersections.) If the set E together with all edges going between hole vertices include any intersections we discard the current labelling being considered. (A convex partition consists of non-intersecting edges).
2. Find the clockwise ordering in which the geometric p-edges of p-edge names will go to the perimeter. That is, we obtain an ordering Φ of the p-edge names in the considered labelling L corresponding to the clockwise order in which the geometric p-edges will appear on the shrunk perimeter. This step takes $O(k \log k)$ time, because it is basically a sorting operation.

We then precompute the following information:

1. We store the homogeneous piece each perimeter vertex lies on.
2. For each homogeneous piece C_i, we store the most counterclockwise vertex v_{cc} and the most clockwise vertex v_c on it.
3. For each non-crossing super-graph in P, we precompute the following information and store it in a 3-dimensional table before starting the dynamic programming algorithm. For each homogeneous piece C_i, for each perimeter vertex v_j on C_i and for each hole vertex h, whose concavity is not removed by edges of the non-crossing graph, we store:
   - The collinear perimeter vertex v of v_j if such a vertex exists, otherwise we store $\perp$ indicating no such vertex exists.
   - The closest clockwise perimeter vertex v_c on C_i of v_j if such a vertex exists, otherwise we store $\perp$.

- The collinear perimeter vertex v of the half-line extension of (v_j, h) if such a vertex exists, otherwise we store $\perp$.
- The closest clockwise perimeter vertex v_c of the half-line extension of (v_j, h).

The above table entries can be computed in linear time by sweeping over each perimeter vertex at most once.

4. For each non-crossing super-graph in P, we also precompute the following information and store it in two-dimensional tables before starting the dynamic programming algorithm. For each hole vertex h whose concavity is not removed by edges of the non-crossing graph:
   (a) Let β be the concave angle at h and α its complements. We store the two edges e_l and e_r, which bound the edges in the non-crossing graph incident to h such that the edge e_r is first encountered if we traverse β clockwise. (e_l and e_r bound the angles α and β incident to h).
   (b) Let l_s be the half-line extension of an edge e_s in the current non-crossing super-graph. We store the closest clockwise and counterclockwise perimeter vertices from l_s.

Before we start the dynamic programming (see Section 2.2) we know:

1. The hole vertex an edge name's corresponding geometric p-edge is incident to.
2. To which homogeneous piece a p-edge should go.
3. The clockwise order in which the geometric p-edges cross the shrunk perimeter. That is, the clockwise ordering Φ of the p-edge names belonging to the current labelling L.
4. All edges connecting hole vertices.

We then use dynamic programming to determine whether it is possible to place the p-edges on perimeter vertices.

2.2 Dynamic Programming Phase of the MNCP Algorithm

We define $O(k^2)$ subproblems for each perimeter vertex as follows: We look at coherent subsequences of p-edge names in the considered clockwise ordering (where coherent means that a hole vertex has all its p-edge names in it or none of them). For each coherent subsequence and for each possible starting perimeter vertex v we try to place the p-edges on the perimeter (this is one subproblem). To be precise, each subproblem is considered to be in the form $(v, f, m, \{A \text{ or B}\})$ where

$v \in 1 \ldots n$ represents the starting vertex on the perimeter for the subproblem.
$f \in 1 \ldots 3k$ indicates the position of the first element of the coherent subsequence in the clockwise ordered list of edge names. Recall that the name of a p-edge (h, a) holds the information about the hole vertex h the p-edge is incident to.
$m \in 1 \ldots 3k$ represents the length of the coherent subsequence (number of named p-edges in the subsequence.)

A: Tells us we are in the problem instance where the subproblem's first p-edge is required to be incident to the starting vertex v.

B: Tells us we are in the problem instance where there is no restriction as to which vertex on the perimeter the first p-edge may be incident to.

The solution to a subproblem is the smallest perimeter piece clockwise from v that is needed to remove concavity around all hole vertices in the coherent subsequence. This solution is represented by the perimeter vertex the last p-edge in the sequence goes to.

The coherent subsequences are processed in the order of increasing length, so that processing longer subsequences can draw on the solutions for shorter subsequences.

For a given f and m, we first solve all type A subproblems and use their solutions to solve the type B subproblems. Let M be the total number of p-edges in a labelling L. If there is a solution to the MNCP problem, which can be constructed from a clockwise ordering Φ of an homogeneous piece assignment Π of edge names of a labelling L of a non-crossing super-graph, then such a solution can be found in the solutions of (v, f, M, B) subproblems. Thus if there is a solution to the MNCP problem for a given input, solving any subproblem (v, f, M, B), the dynamic programming algorithm will always report "yes, there is a solution." Otherwise it will return "no, there is no a solution."

2.3 Solving Type B Subproblems (v, f, m, B)

For a given f and m, we first solve all type A subproblems (see Subsection 2.4) and use their solutions to solve all type B subproblems in linear time. So suppose that all type A subproblems for given f and m have already been solved. We confine ourselves to the clockwise problems, because the counterclockwise problems are symmetric. We solve the (clockwise) type B problems by moving three vertices v, v^*, and v_s (counterclockwise) round the perimeter, where v is the vertex for which we desire a type B solution, v^* is the closest vertex clockwise from v for which a type A solution is known, and v_s is the solution of this type A subproblem.

First we find a vertex v_0, for which a type A solution exists. Finding such a vertex v_0 obviously takes linear time, because checking for a given vertex whether it possesses a type A solution takes a single table access and thus constant time. Initially, we place v at v_0. Next we iterate the following steps, where h denotes the hole vertex in the edge name at position f:

1. We access the solution table to check whether the subproblem (v, f, m, A) has a solution. If it does, we retrieve its solution v' and set $v^* = v$ and $v_s = v'$. (Note that in the first iteration v^* and v_s will necessarily be set, because we have $v = v_0$, for which a type A solution exists.)
2. If the clockwise angle between (h, v) and (h, v^*) is less than the clockwise angle between (h, v) and (h, v_s), we store v_s as the solution of the subproblem (v, f, m, B). Otherwise the solution "wraps round" to v or even beyond v and

thus is useless for solving type A subproblems for bigger m (for which we need the solutions of the type B subproblems, see Subsection 2.4). Consequently we store $\perp$ in the solution table to indicate that the subproblem is unsolvable (or rather that its solution is useless).

3. We find the next vertex v' counterclockwise from v. If $v' = v_0$, we are done and can terminate the process. Otherwise we set $v = v'$ and go to step 1.

Since both v and v^* are moved to each perimeter vertex at most once and since the checks and actions required for each movement take constant time, the total time complexity is linear in the number of perimeter vertices.

2.4 Solving Type A Subproblems (v, f, m, A)

Let h be the hole vertex in the edge name at position f and let C_j, $1 \leq j \leq \mu$, be the homogeneous piece allocated to the edge name at position f. We check the following cases:

1. **Initialization $m = 1$.**

   Let e_l and e_r be the edges incident to h in the non-crossing super-graph, which bound all edges incident to h in the non-crossing super-graph. (e_l and e_r are precomputed and thus obtainable in constant time. Note that it may be $e_l = e_r$). We check whether v lies on C_j. If does not, the subproblem (v, f, m, A) is unsolvable, so we store $\perp$. Otherwise we check whether the clockwise angle from (h, v) to e_l as well as the clockwise angle from e_r to (h, v) are both less than or at most equal to 180^o. If one of them is not, the subproblem (v, f, m, A) is unsolvable, so we store $\perp$. Otherwise we store the vertex v (since in this case the first and the last edge names are the same).

2. **The first and the last of the m edge names contain different hole vertices.**

   Let f', $f \leq f' < f + m - 1$, be the position of the last edge name in the subsequence that contains the same hole vertex h as the edge name at position f (i.e. the first in the subsequence). Note that there may be only one edge name that contains the hole vertex h, i.e., it may be $f' = f$.

   We check whether the subsequence of edge names starting at position f and ending at f' is valid (i.e., contains either all or none of the edge names containing a given hole vertex). This can be done in constant time by simply checking whether the solution table contains an entry for $(v, f, f' - f + 1, *)$, where $*$ may be A or B, since invalid subsequences are not stored.

   If the subsequence is not valid, the subproblem (v, f, m, A) is unsolvable and we store $\perp$ in the solution table, because then there must be edge names at positions f_1, $f < f_1 < f'$, and f_2, $f' < f_2 \leq m + f - 1$, which contain the same hole vertex h' and thus connecting both h and h' to the perimeter must lead to intersecting edges.

   If, however, the subsequence starting at position f and ending at f' is valid, we access the solution table to check whether the subproblem $(v, f, f' - f + 1, B)$ has a solution. If it has not, the subproblem (v, f, m, A) is unsolvable

and we store $\perp$. Otherwise we retrieve the solution v_1 of the subproblem $(v, f, f' - f + 1, B)$.

Next we check whether the subproblem $(v_1, f' + 1, m - f' + f - 1, B)$ has a solution. If it has not, (v, f, m, A) is unsolvable and we store $\perp$. Otherwise we retrieve the solution v_2 of the subproblem $(v_1, f' + 1, m - f' + f - 1, B)$ and check whether the clockwise angle between (h, v_1) and (h, v_2) is less than the clockwise angle between (h, v_1) and (h, v). If it is, we store v_2 as the solution of the subproblem (v, f, m, A), otherwise it is unsolvable and we store $\perp$.

Since all table accesses take constant time, this case can be solved in constant time.

3. **The first and the last of the m edge names contain the same hole vertex h.**

   Let C_l, $1 \leq l \leq \mu$, be the homogeneous piece allocated to the last edge name. (C_j is the homogeneous piece allocated to the first edge name.) In this case we have to distinguish three cases:

   (a) *h is contained in exactly two of the m edge names and is not incident to an edge of the current non-crossing super-graph.*

   Since h is not incident to an edge of the non-crossing super-graph, the two p-edges must be collinear in order to remove the concavity at h. Since we solve a type-A subproblem, one p-edge must be (h, v). We check whether v lies on C_j. If does not, the subproblem (v, f, m, A) is unsolvable, so we store $\perp$.

   Otherwise we check whether there is a perimeter vertex v_1 collinear to h and v. This can be done in constant time using the precomputed information. If there is not or if v_1 does not lie on C_l, the subproblem (v, f, m, A) is unsolvable and we store $\perp$.

   If now $m = 2$, we are already done and can store v_1 as the solution of the subproblem (v, f, m, A).

   If $m > 2$, we access the solution table to check whether the subproblem $(v, f + 1, m - 2, B)$ is solvable. If it is not, the subproblem (v, f, m, A) is unsolvable and we store $\perp$. Otherwise we retrieve the solution v_2 of the subproblem $(v, f + 1, m - 2, B)$. If does not exist, the subproblem (v, f, m, A) is unsolvable, so we store $\perp$. Otherwise we check whether the clockwise angle between (h, v) and (h, v_2) is less than or at most equal to the clockwise angle between (h, v) and (h, v_1). If it is, we store v_1 as the solution. Otherwise the subproblem (v, f, m, A) is unsolvable and we store $\perp$.

   (b) *h is contained in exactly two of the m edge names and is incident to at least one edge of the current non-crossing super-graph.*

   We check whether v lies on C_j, if it does not, the subproblem (v, f, m, A) is unsolvable and we store $\perp$. Otherwise, if $m > 2$, we access the solution table to check whether the subproblem $(v, f + 1, m - 2, B)$ is solvable. If it is not, we store $\perp$ as the solution of the subproblem (v, f, m, A). Otherwise we retrieve the solution v_1 of the subproblem

$(v, f+1, m-2, B)$. If, however, $m = 2$, we set v_1 to the next perimeter vertex clockwise from v.

Next, we find a perimeter vertex v_2 with v_2 equal to v_1 if v_1 lies on C_l. If, however, v_1 does not lie on C_l we set v_2 to the most counterclockwise vertex on C_l. The vertex v_2 can be obtained from the precomputed information in constant time. We check whether the edges (h, v) and (h, v_2) together with the edges of the non-crossing super-graph incident to h remove the concavity at h. This check can be done in constant time, because there are at most $k+1$ edges we have to consider. If the concavity is removed, we store v_2 as the solution of the subproblem (v, f, m, A).

If, however, the concavity is not removed, there must be edges e_1 and e_2 incident to h (among (h, v), (h, v_2), and the edges of the non-crossing super-graph), between which the clockwise angle (from e_1 to e_2) is greater than 180^o and between which there is no other edge incident to h. These two edges can be determined in the concavity check carried out above.

If $e_1 \neq (h, v_2)$, then the subproblem (v, f, m, A) is unsolvable, because we can only move v_2 further clockwise to obtain a solution and this obviously does not remove the concavity if by this we do not move e_1. Otherwise we determine the vertex v_3 that is closest clockwise from the half-line extension of e_2 (obtainable from the precomputed information in constant time). If v_3 does not lie in C_l we store $\perp$ as the solution of the subproblem (v, f, m, A).

Similar to the above, we check whether the edges (h, v) and (h, v_3) together with the edges of the non-crossing super-graph incident to h remove the concavity at h. If they do, we store v_3 as the solution. Otherwise the subproblem (v, f, m, A) is unsolvable, so we store $\perp$.

Since all table accesses take constant time, the whole case can be solved in constant time.

(c) *h is contained in exactly three of the m edge names.*
(Note that h is not incident to an edge of the current non-crossing super-graph in this case.)

Let f', $f < f' < f + m - 1$, be the position of the middle edge name containing h. (Note that f and $f + m - 1$ are the positions of the other two edge names.) We define the middle p-edge as the closest clockwise p-edge from v, which together with the last p-edge eliminate concavity at h.

For any given (fixed) f and m we solve all the n subproblems together (there are $O(n)$ perimeter vertices for each fixed f and m) in a coordinated way in total linear time. The general idea is that we move three vertices, v, v_1 and v_2, (clockwise) around the perimeter, with v_1 and v_2 being the end points of the edges corresponding to the edge names at positions f' and $f + m - 1$. Based on certain conditions, we move v_1 and v_2 clockwise in order to find a solution, or to find out that no solution exists. In this way we can exploit information gathered by solving

a subproblem (for a given v) to speed up the search for a solution of the next.

Let u, u_1, u_2, respectively, be the most counterclockwise vertex in the homogeneous pieces C_j, C_k, C_l, $(1 \leq j, k, l, \leq \mu)$ allocated to the first, middle, and last edge name containing h. Initially, let $v = u$ and $v_1 = u_1$, $v_2 = u_2$. If $C_j = C_k = C_l$, then we would not be able to remove the concavity around h (unless, of course, h is the only hole vertex in the whole polygon, but this easy case can be solved separately, without dynamic programming).

Thus in the case where $C_j = C_k = C_l$ we store $\perp$ in the solution table for all the n subproblems indicating they are all unsolvable. Otherwise we try to find the solution of a subproblem (v, f, m, A) by processing the checklist below. If in a point of the checklist v_1 or v_2 is moved, we restart the checks at the beginning of the list. If, we get through the whole checklist without moving v_1 or v_2, we store v_2 as the solution of the subproblem (v, f, m, A).

However, if any check or action we carry out in the checklist indicates that the currently processed subproblem (v, f, m, A) is unsolvable, we store $\perp$ in the solution table. In both the cases, where we find out the subproblem is unsolvable or we store v_2 as solution of subproblem, we continue by finding the next vertex v^* clockwise from v. If v^* is not on C_j we are done with all subproblems. Otherwise we start to work on the new subproblem (with $v = v^*$) by processing the same checklist. When we start solving the new subproblem, v_1 and v_2 are at the positions they ended up at when we processed the checklist in the preceding turn.

Checklist

i. *Checks and actions needed when v, v_1, and v_2 are not on C_j, C_k and C_l respectively.*
   A. If v is not on C_j we are done with all subproblems for the given fixed f and m. Stop.
   B. If v_1 is not on C_k, then the subproblem (v, f, m, A) is unsolvable.
   C. If v_2 is not on C_l then the subproblem (v, f, m, A) is unsolvable.

ii. *For a given v, checks and actions needed to ensure that v_1 is clockwise from v and counterclockwise from v_2.*
   A. If $v = v_2$, the subproblem (v, f, m, A) is unsolvable.
   B. If $v = v_1$, we find the next vertex v' clockwise from v_1 and set $v_1 \leftarrow v'$.
   C. If $v_1 = v_2$, we find the next vertex v' clockwise from v_2 and set $v_2 \leftarrow v'$.

iii. *Checks and actions that ensure the angles clockwise from (h, v_2) to (h, v), from (h, v_1) to (h, v_2) and from (h, v) to (h, v_1) are all not greater than 180^o.*

A. If the clockwise angle from (h, v_2) to (h, v) is greater than 180^o, we find the next vertex v' clockwise from v_2. and set $v_2 \leftarrow v'$.

B. If the clockwise angle from (h, v_1) to (h, v_2) is greater than 180^o, we find the next vertex v' clockwise from v_1 and set $v_1 \leftarrow v'$.

C. If the clockwise angle from (h, v) to (h, v_1) is greater than 180^o, the subproblem (v, f, m, A) is unsolvable.

iv. *Checking the subproblem* $(v_1, f' + 1, m - f' + f - 2, B)$.
We access the solution table to check whether the subproblem $(v_1, f' + 1, m - f' + f - 2, B)$ has a solution. If it does not, the subproblem (v, f, m, A) is unsolvable. Otherwise we retrieve the solution v' of the subproblem $(v_1, f' + 1, m - f' + f - 2, B)$. If the clockwise angle from (h, v_1) to (h, v') is greater than the clockwise angle from (h, v_1) to (h, v_2) we find the next vertex v' clockwise from v_2 and set $v_2 \leftarrow v'$.

v. *Checking the subproblem* $(v, f + 1, f' - f - 1, B)$.
We access the solution table to check whether the subproblem $(v, f + 1, f' - f - 1, B)$ has a solution. If it does not, the subproblem (v, f, m, A) is unsolvable. Otherwise we retrieve the solution v' of the subproblem $(v, f + 1, f' - f - 1, B)$. If the clockwise angle between (h, v) and (h, v') is greater than the clockwise angle between (h, v) and (h, v_1), we find the next vertex v' clockwise from v_1 and set $v_1 \leftarrow v'$.

We observe that v, v_1 and v_2 visit each perimeter vertex at most once. That is, in the above algorithm v, v_1 and v_2 start at the most counterclockwise perimeter vertex of their respective homogeneous pieces. When they are moved away from their respective homogeneous pieces, they are never moved back into their respective homogeneous pieces anywhere in the algorithm. In fact in the entry i. in the checklist,

- Step A. sees to it that v is never moved into C_j when v leaves the most clockwise vertex on C_j.
- Step B. sees to it that v_1 is never moved into C_k when v_1 leaves the most clockwise vertex on C_k.
- Step C. sees to it that v_2 is never moved into C_l when v_2 leaves the most clockwise vertex on C_l.

Thus it is clear that v, v_1 and v_2 visit each perimeter vertex at most once. Since the total movement of v, v_1 and v_2 is linear and each movement takes constant time, it takes in total linear time to solve all subproblems for a given fixed f and m. There are a constant number of fixed f and m, thus all subproblems can be solved together in total linear time.

2.5 Space and Time Complexity of the Dynamic Programming Algorithm

The memory requirement in the worst case is dominated by the $O(n \cdot k^2)$ space for the table entries.

For a given f and m, we solve all type A subproblems in linear time. We then solve the type B subproblems for given f and m also in linear time using the type A subproblems. Thus each call of the dynamic programming takes $O(n \cdot k^2)$ time. We call the dynamic programming algorithm at most $O(k^{6k-7} \cdot 2^{16k})$ times.

2.6 Outputting the Actual MNCP

We can output the actual MNCP as follows: For a homogeneous piece assignment Π of a labelling L where the dynamic programming algorithm outputs that a MNCP can be constructed, we can obtain the actual MNCP by running the dynamic programming algorithm and storing for each subproblem not only the vertex on the perimeter where the last p-edge goes, but also the middle p-edge, if a middle p-edge is required. See 3(c) of Section 2.4 above for the case when a middle p-edge is needed.

3 Analysis of the MNCP Algorithm

We considered all possible combinations of non-crossing super-graphs on the k hole vertices. There are $(2^3 \cdot 59)^k O(k^{-6})$ such non-crossing super-graphs [3]. It takes $(2^3 \cdot 59)^k O(k^{-6})$ time to enumerate all such non-crossing super-graphs [3].

For each non-crossing super-graph G', we considered all the possible labellings, with each labelling indicating the number of p-edges corresponding to each hole vertex. There are not more than 3^k such labellings for each non-crossing super-graph.

For each labelling L, we considered all possible homogeneous piece assignments Π. There are $O(k^{6k} \cdot 2^{3k})$ such assignments to be considered for a given labelling L. For each assignment, we then obtained the clockwise ordering Φ of the corresponding edge names in $O(k \log k)$ time.

For the ordering Φ of an assignment Π of a labelling L, we then check for possible edge intersections in time polynomial in k and then run the dynamic programming algorithm which takes linear ime each time it is called.
From the given MNCP algorithm we arrive at the following theorem:

Theorem 1. *For any convex polygon P with k hole vertices, the MNCP problem can be solved in the following time complexities:*

- *For any constant k, it can be solved in linear time.*
- *For $k = O(\frac{\log n}{\log \log n})$, it can be solved in time polynomial in n.*

Proof. From the analysis of the MNCP algorithm the total time taken to solve the MNCP problem for a constant number k of hole vertices is $O(n)$. For $k = O(\frac{\log n}{\log \log n})$, it can be solved in $O((2^3 \cdot 59 \cdot 3)^k) \cdot k^{-6} \cdot k^{6k} \cdot 2^{3k} \cdot n)$ which simplifies

to $O(n \cdot k^{6k-5} \cdot 2^{16k})$. When k grows and $k \geq 8$, by far the fastest growing factor in the time complexity of Theorem 1 is k^{6k-5}. If $k = O(\frac{\log n}{\log \log n})$ then the value of k^{6k-5} is polynomial in n like all the other factors in the given time complexity [3]. Therefore Theorem 1 holds.

We observe that the above problem is fixed-parameter tractable when the problem is parameterized by the number k of the hole vertices [1]. It is straightforward to generalize this result to the case where we have as input a PSLG G and k is the total number of holes and/or reflex vertices in the convex hull of G.

References

1. Downey, R., Fellows, M.: Parameterized Complexity. Springer-Verlag, New York (1999).
2. Garcia, A., Noy, M., Tejel, J.: Lower Bounds on the Number of Crossing-free Subgraphs of K_N. Computational Geometry, Theory and Applications, Vol. 16. Elsevier Science, Amsterdam, Netherlands (2000) 211–221.
3. Grantson, M.: Fixed-Parameter Algorithms and Other Results for Optimal Convex Partitions. LU-CS-TR:2004-231, ISSN 1650-1276 Report 152. Lund University, Sweden (2004).
4. Greene, D.: The Decomposition of Polygons into Convex Parts. In: F.P. Preparata, editor,Computational Geometry, vol. 1 of Adv. Comput. Res. JAI Press, London, England, (1983) 235–259
5. Keil, J.: Decomposing a Polygon into Simpler Components. SIAM Journal on Computing. Society of Industrial and Applied Mathematics, Vol 14. Philadelphia, PA, USA (1985) 799–817
6. Keil, J., Snoeyink, J.: On the Time Bound for Convex Decomposition of Simple Polygons. In proceedings of the 10th Canadian Conference on Computational Geometry. Montreal, Canada (1998) 54–55
7. Lingas, A.: The Power of Non-Rectilinear Holes. Lecture Notes in Computer Science, Vol. 140. Springer-Verlag, Heidelberg, Germany (1982) 369-383
8. Santos, F., Seidel, R.: A Better Upper Bound on the Number of Triangulations of a Planar Point Set. arXiv:math.CO/0204045 v2 (2002).

Tight Time Bounds for the Minimum Local Convex Partition Problem

Magdalene Grantson and Christos Levcopoulos

Department of Computer Science,
Lund University, Box 118, 221 Lund, Sweden
{magdalene, christos}@cs.lth.se

Abstract. Let v be a vertex with n edges incident to it, such that the n edges partition an infinitesimally small circle C around v into convex pieces. The minimum local convex partition (MLCP) problem asks for two or three out of the n edges that still partition C into convex pieces and that are of minimum total length. We present an optimal algorithm solving the problem in linear time if the edges incident to v are sorted clockwise by angle. For unsorted edges our algorithm runs in $O(n \log n)$ time. For unsorted edges we also give a linear time approximation algorithm and a lower time bound.

1 Introduction

We consider the minimum local convex partition (MLCP) problem. Its input is a vertex v with n incident edges, such that the edges partition an infinitesimally small circle C around v into convex pieces. (The radius of C is less than the shortest edge incident to v). It is solved by selecting two or three out of the n edges that still partition C into convex pieces and that are of minimum total length.

We present an optimal algorithm which solves the MLCP problem in linear time when the edges are sorted clockwise by angle and in $O(n \log n)$ time otherwise. For unsorted edges we also give:

1. A linear time algorithm, which finds a $(1 + \epsilon)$- or $(1.5 + \epsilon)$-approximation depending on whether the optimal solution has 3 or 2 edges, respectively.
2. An $\Omega(n \log n)$ worst-case time lower bound for approximating the MLCP problem in the algebraic computation tree within any factor less than 1.5.

Our optimal algorithm can be used directly to solve the minimum weight convex partition (MWCP) problem [4] where the input is a convex polygon with a single hole vertex. The general MWCP problem (a convex polygon with several hole vertices) can be solved by partitioning the input polygon into some kind of convex pieces such that each piece contains a hole vertex, and then using our optimal algorithm to remove the concavity around each hole vertex [4]. The MWCP problem has applications in computer graphics [10], image processing [9], and database systems [7].

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 95–105, 2005.

2 Optimal Algorithm for the MLCP Problem

To simplify the explanation of our optimal MLCP algorithm, we assume that no two edges are of the same length. If any two edges are of the same length, we consider the edge with the larger index — w.r.t. some arbitrary order of the edges — as the longer edge among the two.

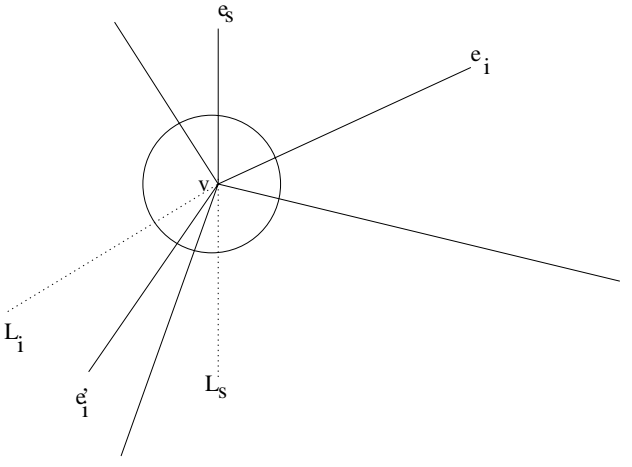


Fig. 1. Shortest edge e_s and its half-line extension

Our algorithm works in two phases: First we check for a two edge solution and then for a three edge solution. The minimum of the two obtained solutions solves the MLCP problem.

Let e_s be the shortest edge incident to v and let L_s be its half-line extension (see Figure 1). We denote the region bounded by a clockwise walk from e_s (including e_s) to L_s (excluding L_s) by R_{cw} and its complement w.r.t. C by R_{ccw}. (We assume that neither R_{cw} nor R_{ccw} are empty, because otherwise the problem has no solution.) We number the edges in R_{cw} in clockwise order such that $e_s = e_1$. Let e_k, $1 \leq k < n$, denote the last edge in R_{cw}.

If e_l and e_r are edges s.t. e_r is at most 180^o clockwise from e_l and if L_l and L_r are the half-line extensions of e_l and e_r, we refer to the region bounded by a clockwise walk from e_l to e_r and from L_l to L_r as the *wedge* and the *op-wedge* (opposite wedge) of (e_l, e_r), respectively.

2.1 Checking for a Two Edge Solution

With the help of a sweep-line l rotating around v, for each edge e_i, $1 \leq i \leq k$, in R_{cw} we try to find an edge e'_i in R_{ccw}, such that e_i and e'_i are collinear. Since each edge is considered exactly once in this search, the overall time complexity is $O(n)$ if the edges are sorted by angle. Otherwise we have a time complexity of $O(n \log n)$, which stems from the complexity of sorting the edges.

2.2 Checking for Three Edge Solution

Observation 1. *The shortest edge incident to v must always be selected for three edge solutions.*

Proof. We show that the assumption that e_s is not selected leads to a contradiction. So let us assume that e_s does not have to be selected. In this case there are three edges e', e'', $e''' \in E - \{e_s\}$, which constitute a solution to the MLCP problem. Hence, e', e'', and e''' partition the circle C into three convex pieces. W.l.o.g. we may assume that e_s lies in the sector of C bounded by e' and e''. The edges e_s and e''' partition C into two pieces, one containing e' and the other containing e''. At least one of the two has to be convex, so we may assume w.l.o.g. that e' lies in the convex region. Now we observe that the edges e_s, e'' and e''' also partition C into convex pieces, and since e_s is shorter than e', we obtain a contradiction. (Contrary to our assumption, e', e'', and e''' are no solution to the MLCP problem, because they do not have minimum total length.)

The remaining two edges are selected as follows: For each edge e_i, $2 \leq i \leq k$, in R_{cw} we find the edge e'_i in R_{ccw} that forms a convex partition of minimum length (among those containing e_s and e_i, see Figure 1). Since only edges in the op-wedge (e_s, e_i) can form a convex partition with e_s and e_i, we need not search all of R_{ccw}; the op-wedge (e_s, e_i) suffices. Even better: only for $i = 2$, we have to search the whole op-wedge (e_s, e_2) for e'_2. For $i > 2$ the search can be restricted to the op-wedge (e_{i-1}, e_i), reusing previous results. Thus each edge is considered exactly once, yielding a linear time complexity if the edges are sorted by angle and $O(n \log n)$ time otherwise.

Theorem 1. *The MLCP problem can be solved in linear time if the n edges incident to the vertex v are sorted by angle and in $O(n \log n)$ time otherwise.*

3 Linear Time Approximation

For unsorted edges we designed a linear time approximation algorithm by:

1. Showing that we can get a 2-approximation in linear time.
2. Using the 2-approximation solution we select a subset of m edges incident to v, where m depends on a chosen approximation quality ϵ ($\epsilon > 0$). We show that running our optimal MLCP algorithm on these m edges yields a solution that is a $(1+\epsilon)$- or $(1.5+\epsilon)$-approximation depending on whether the optimal solution consists of three or two edges, respectively. In the algebraic computation tree the overall running time is $O(n \log \frac{1}{\epsilon})$.

3.1 A 2-Approximation Algorithm for the MLCP Problem

Let E be the set of edges incident to v. The following algorithm reduces the number of edges in the set E to three by recursively:

- Finding the upper median edge length $|e_M|$ and grouping the edges in E into two sets $E_S = \{e \in E \mid |e| \leq |e_M|\}$ and $E_L = \{e \in E \mid |e| > |e_M|\}$.
- If the edges in E_S partition C into convex pieces, we discard all edges in E_L and apply the algorithm recursively to E_S.
- If the edges in E_S do not partition C into convex pieces, there must be bounding edges $b_1, b_2 \in E_S$, such that a half-line starting at v rotated clockwise around v from b_1 to b_2 sweeps an angle of $\theta < 180^o$ and meets all edges in E_S. In this case we apply the algorithm recursively to $E_L \cup \{e_s, b_1, b_2\}$.

The edges b_1 and b_2 can be found during the split of the edges into E_S and E_L as follows: The first two edges assigned to E_S are used to initialize b_1 and b_2 ensuring that the clockwise angle from b_1 to b_2 is $\leq 180^o$. Afterwards if a new edge is added to E_S, it is checked whether it lies on a clockwise walk from the current b_1 to the current b_2. If it is, the old b_1 and b_2 are kept. If it is not, it is checked to which of the two edges b_1 and b_2 it is closer and this edge is then replaced by the new edge. This also allows a simple check whether the edges in E_S partition C into convex pieces. If the angle measured in clockwise direction from b_1 to b_2 is less than 180 degrees, the edges in E_S do not induce a convex partition, otherwise they do.

Pseudocode of the 2-Approximation Algorithm

Input: A set E of n distinct edges partitioning the infinitesimally small circle C around v into convex pieces.

Output: Edges e_s, e'', e''' that partition the infinitesimally small circle C around v into convex pieces and that satisfy $|e_s| + |e''| + |e'''| \leq 2l_{\text{opt}}$, where l_{opt} is the length of the optimal solution of the MLCP problem.

1. Select the edge with minimum length e_s.
2. **if** $n \leq 5$
3. **then** use a brute force algorithm to find 2 edges e'', e''' partitioning C into convex pieces s.t. $|e_s| + |e''| + |e'''|$ is minimal
4. **stop**
5. **else** Select the edge e_M with the **upper median** length.
6. $E_S \leftarrow \{e \in E \mid |e| \leq |e_M|\ \}$
7. $E_L \leftarrow \{e \in E \mid |e| > |e_M|\ \}$
8. Let $b_1, b_2 \in E_S$ be the two edges bounding the maximum piece of C.
9. **if** edges in E_S partition C into convex pieces
   ($*$ b_1, b_2 enclose a maximum angle $\theta \geq 180$, see above. $*$)
10. **then** $E \leftarrow E_S$ ($*$ Discard all edges in E_L $*$)
11. **else** $E \leftarrow E_L \cup \{e_s, b_1, b_2\}$ ($*$ Discard all edges in $E_S - \{e_s, b_1, b_2\}$ $*$)
12. Let $n =$ the number of edges in E.
13. **goto** step 2.

Theorem 2. *There is a linear time approximation algorithm which achieves an approximation ratio of 2 for the MLCP problem.*

Proof. Using the worst-case linear-time median-finding algorithm devised by Blum, Floyd, Pratt, Rivest, and Tarjan [3], step 5 in the pseudocode above

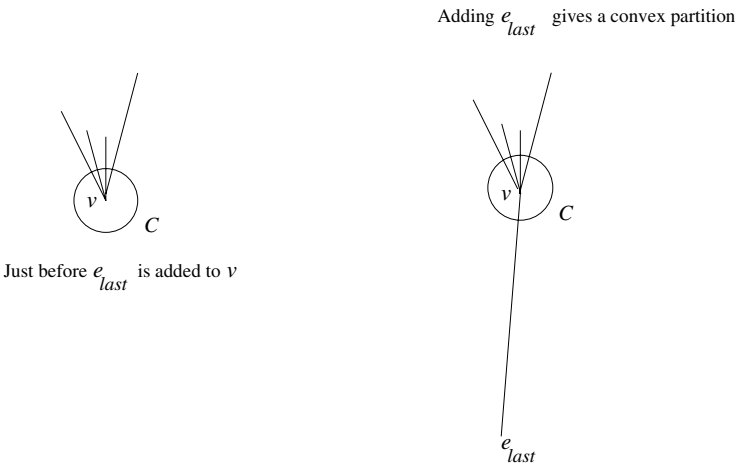


Fig. 2. Figure showing a convex partition of C after adding e_{last}

takes linear time. The rest of the steps in all take linear time, because the main work to be done is the split into the two sets E_S and E_L. For this split each edge has to be considered once. In the recursion, the set of edges to be split has roughly half the size of the set used in the previous step. So the total number of operations is basically $cn + c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} + ... = 2cn \in O(n)$, where c is a constant that measures the costs of processing one edge and n is the initial number of edges. Thus the total time complexity of above algorithm is linear.

To show that the above algorithm gives a 2-approximation of the optimum, we observe that the shortest edge e_s is always in the solution produced by this algorithm (because it is in E_S and in $E_L \cup \{e_s, b_1, b_2\}$ and thus is never discarded). In order to get information about the lengths of the other two edges, let us assume for the moment that the edges emanating from v are sorted by length into a list and one by one they are added to v from the shortest until we arrive at a convex partition of C (see Figure 2).

Let e_{last} be the last edge added to v, which eventually resulted in a convex partition of C, and let R be the concave region before e_{last} was inserted (see Figure 2). We observe that the above algorithm finds the solution that consists of the edges e_{last}, e_s and an appropriate third edge e'' with $|e''| \leq |e_{\text{last}}|$. Consequently, we know that for the length l_{apx} of the result of the above algorithm, $l_{\text{apx}} \leq |e_s| + 2|e_{\text{last}}|$ holds. This implies

$$l_{\text{apx}} \leq 2(|e_s| + |e_{\text{last}}|). \tag{1}$$

In addition, we observe that the optimal solution has to contain an edge e' in the region R for the convex partition property to be fulfilled (regardless of whether the optimal solution has two or three edges). Since the region R contains only edges of length $\geq |e_{\text{last}}|$ (all shorter edges have already been added beforehand), we conclude that

$$|e'| \geq |e_{\text{last}}|. \tag{2}$$

As a consequence we have that the length l_{opt} of the optimal result is $l_{\text{opt}} \geq 2|e_s| + |e_{\text{last}}|$ if the optimal solution is made up of three edges and $l_{\text{opt}} \geq |e_s| + |e_{\text{last}}|$ if it is made up of two edges. Thus in any case we have $l_{\text{opt}} \geq |e_s| + |e_{\text{last}}|$. Substituting this into (1) we get

$$l_{\text{apx}} \leq 2(|e_s| + |e_{\text{last}}|) \leq 2l_{\text{opt}}.$$

This completes the proof.

Note that it may be possible to tighten the bound on the quality of the result of the approximation algorithm after it has terminated and one knows the solution it yields. We know that

$$l_{\text{apx}} \leq |e_s| + 2|e_{\text{last}}|$$

and that

$$|e_s| + |e_{\text{last}}| \leq l_{\text{opt}}.$$

Therefore we also know (by multiplying the two inequalities — from $a \leq b$ and $c \leq d$ it follows $ac \leq bd$) that

$$l_{\text{apx}}(|e_s| + |e_{\text{last}}|) \leq l_{\text{opt}}(|e_s| + 2|e_{\text{last}}|)$$

and consequently

$$l_{\text{apx}} \leq l_{\text{opt}} \frac{|e_s| + 2|e_{\text{last}}|}{|e_s| + |e_{\text{last}}|}.$$

Depending on the values of $|e_s|$ and $|e_{\text{last}}|$ the factor on the right may be much smaller than 2 (its value can be computed, because we know the values of $|e_s|$ and $|e_{\text{last}}|$ from the result of the approximation algorithm).

3.2 A Linear Time Approximation Algorithm

We now use the length of the longest edge e_{last} in the solution of the approximation algorithm above to discard long edges which we know cannot be part of the optimal solution.

Reducing the Number of Edges in the Input Set E.
In this section, we select a constant number m of edges from the input set E of n edges using the length of the longest edge e_{last} in the solution of the 2-approximation algorithm. We then run our optimal MLCP algorithm on the m edges to obtain a $(1+\epsilon)$- or $(1.5+\epsilon)$-approximation, depending on the number of edges in the optimal solution.

Let e_s, e'', and e_{last} be the edges found by the approximation algorithm. Then we have $|e_s| < |e''| < |e_{\text{last}}|$ and $|e_s| + |e''| + |e_{\text{last}}| = l_{\text{apx}}$. If the optimal solution

has three edges e_s, e and e' we have $|e_s| < |e| < |e'|$ and $|e_s| + |e| + |e'| = l_{\text{opt}}$. If the optimal solution has two edges e and e', we have $|e| \leq |e'|$ and $|e| + |e'| = l_{\text{opt}}$. We observe that in both cases $|e'|$ must be less than or equal to $2|e_{\text{last}}|$, because if $|e'| > 2|e_{\text{last}}|$, then we have

optimal solution has two edges:

$$\begin{aligned} l_{\text{opt}} &= |e| \ + |e'| \\ &> |e| \ + 2|e_{\text{last}}| \\ &\geq |e_s| + 2|e_{\text{last}}| \\ &\geq |e_s| + |e''| + |e_{\text{last}}| \quad = \quad l_{\text{apx}} \end{aligned}$$

optimal solution has three edges:

$$\begin{aligned} l_{\text{opt}} &= |e_s| + |e| + |e'| \\ &> |e_s| + |e| + 2|e_{\text{last}}| \\ &> |e_s| + 2|e_{\text{last}}| \\ &\geq |e_s| + |e''| + |e_{\text{last}}| \quad = \quad l_{\text{apx}} \end{aligned}$$

That is, we can derive $l_{\text{opt}} > l_{\text{apx}}$, which is a contradiction.

Using the observation that the longest edge e' in the solution of the optimal algorithm cannot be longer than $2|e_{\text{last}}|$ we select a constant number m of "potential edges" as follows:

- Execute the above algorithm to find the longest edge e_{last} in the approximation solution.
- From the set E of all n edges we select (in linear time) a set

$$E' = \{e \in E \mid |e| \leq 2|e_{\text{last}}|\}.$$

- We group the edges in E' into buckets such that each bucket contains edges within a given range of length. That is:
  - Partition the range of lengths from $|e_s|$ to $2|e_{\text{last}}|$ (all edges in E' have a length in this range) in such a way that the length difference within a bucket is at most $\frac{\epsilon}{2}|e_{\text{last}}|$. Let k be the number of buckets created. It is

$$k \leq \left\lceil \frac{2|e_{\text{last}}|}{\frac{\epsilon}{2}|e_{\text{last}}|} \right\rceil = \left\lceil \frac{4}{\epsilon} \right\rceil .$$

    Since there is no edge of length less than $|e_s|$, a more precise upper bound on the number of buckets is

$$k = \left\lceil \frac{2|e_{\text{last}}| - |e_s|}{\frac{\epsilon}{2}|e_{\text{last}}|}. \right\rceil$$

  - Each bucket i, $1 \leq i \leq k$, contains edges of length in the interval

$$\left[|e_s| + (i-1)\frac{\epsilon}{2}|e_{\text{last}}|, |e_s| + i\frac{\epsilon}{2}|e_{\text{last}}| \right)$$

    The only exception is the last bucket k where we include the right boundary. This is because the fraction $\frac{2|e_{\text{last}}| - |e_s|}{\frac{\epsilon}{2}|e_{\text{last}}|}$ may evaluate to an integer.

In the algebraic computation tree model it takes $O(n \log \frac{4}{\epsilon})$ time to group the edges in E' into their respective buckets if we do a binary search on the $\lceil \frac{4}{\epsilon} \rceil$ buckets.

One can also group the edges in E' into their respective buckets in linear time by using an *index computation scheme* as follows: For each edge $e \in E'$, we compute the corresponding bucket index i as:

$$i = \begin{cases} k, & \text{if } |e| = 2|e_{\text{last}}|, \\ \left\lfloor \dfrac{|e| - |e_s|}{\frac{\epsilon}{2}|e_{\text{last}}|} \right\rfloor + 1, & \text{otherwise.} \end{cases}$$

Since $1 \leq i \leq k$, it takes constant time to locate the bucket i, a given edge e belongs to. We note, however, that the floor function is not contained (or takes more than constant time) in the algebraic computation tree model.

– W.l.o.g. let us assume the shortest edge $e_s \in E'$ incident to v is vertical and above v. Let L_s be the half line extension of e_s. For each bucket i, $1 \leq i \leq k$, let edge e_{cw} be the edge which is first reached when one walks clockwise from L_s along the circle C. In a similar way let edge e_{ccw} be the edge in bucket i which is reached first when one moves counter-clockwise from L_s along the circle C. From each bucket i, we select the edges e_{cw} and e_{ccw} (if they exist) giving us no more than $2k$ edges, where $2k$ is upper bounded by a constant depending on ϵ. The number of edges obtained in this way we denote by m. The shortest edge e_s and its half-line extension L_s divides C into two parts which we call C_{cw} and C_{ccw}. For each bucket i, the edges e_{cw} with $e_{\text{cw}} \cap C_{\text{cw}} \neq \emptyset$ and e_{ccw} with $e_{\text{ccw}} \cap C_{\text{ccw}} \neq \emptyset$ are selected to "replace" the rest of the edges intersecting C_{cw} and C_{ccw}, respectively. The rest of the edges intersecting C_{cw} and C_{ccw} are deleted, because:
  1. In terms of their angular position, e_{cw} and e_{ccw} induce the largest angles from e_s. Thus they can replace any other edge in bucket i which can be in the optimal 3-edge solution. (e_{cw} and e_{ccw} are further away from e_s than any other edge in bucket i.)
  2. The maximum length difference between two edges in a bucket is $\frac{\epsilon}{2}|e_{\text{last}}|$.

We show how to get from the selected number of edges an $(1+\epsilon)$- or $(1.5+\epsilon)$-approximation of the optimal solution of the MLCP problem in linear time.

Theorem 3. *If the optimal solution to the MLCP problem is made up of three edges, there is a linear time $(1+\epsilon)$-approximation to the optimum.*

Proof. We assume that the optimal solution is made up of three edges, which partition C into convex pieces. Let e_s, e, and e_i with $|e_s| < |e| < |e'|$ be these edges. Let $|e_s| < |e''| < |e'''|$ be the edges in the approximate solution obtained by running the optimal MLCP algorithm on the set of $m \leq 2 \cdot \lceil \frac{4}{\epsilon} \rceil$ selected edges. (It takes $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ time to run our optimal MLCP algorithm given the selected m edges as input). We observe that discarding some of the n input edges emanating from v may result in discarding edge(s) which are in the optimal solution. Let $l_{\text{opt}} = |e_s| + |e| + |e'|$ be the total length of the edges in the optimal solution and $l_{\text{apx}} = |e_s| + |e''| + |e'''|$ be the total length of the edges in the approximate solution.

If an edge in the optimal solution in C_{ccw} is deleted from bucket i, we could choose the edge e_{ccw} in bucket i to replace it. In the same way if we eliminated

an edge in C_{cw}, we could choose the edge e_{cw} from the same bucket to replace it. Thus the total edge length of the counterpart of e and e' will be longer than $|e| + |e'|$ by at most $\epsilon|e_{last}|$. (The maximum length difference between two edges in a bucket is $\frac{\epsilon}{2}|e_{\text{last}}|$.) Thus we infer that if we run the optimal MLCP algorithm on the chosen m edges we always get a solution which is not more than $l_{\text{opt}} + \epsilon|e_{\text{last}}|$. From Equation 2 (in proof of Theorem 2) we know

$$|e_{\text{last}}| \leq |e'|$$

and consequently

$$|e_{\text{last}}| \leq l_{\text{opt}}.$$

Hence

$$l_{\text{apx}} \leq l_{\text{opt}} + \epsilon|e_{\text{last}}| \leq (1+\epsilon) \cdot l_{\text{opt}}.$$

So far we assumed that the optimal solution is made up of three edges. However, there are cases where it contains only two edges. For these cases we have the following theorem:

Theorem 4. *If the optimum solution to the MLCP problem is made up of two edges, there is a linear time* $(1.5+\epsilon)$*-approximation to the optimum.*

Proof. Let l_{opt} be the total length of edges (e, e') in the optimal solution (in this case the shortest edge e_s need not be selected), and l_{apx} be the total length of edges in the approximate solution when the optimal MLCP algorithm is run on the chosen m edges as input. From the way we choose the edges, we know that $|e_{\text{last}}| < l_{\text{opt}}$. Among the m edges we could choose e_s and the counterparts of (e, e') using the same reasoning as in the proof of Theorem 3. Thus we have

$$l_{\text{apx}} \leq |e_s| + l_{\text{opt}}(1+\epsilon).$$

We observe that $|e_s| \leq \frac{1}{2} l_{\text{opt}}$, since $|e_s| \leq |e|$ and $|e_s| \leq |e'|$, and thus

$$l_{\text{apx}} \leq 0.5 l_{\text{opt}} + l_{\text{opt}}(1+\epsilon) = (1.5+\epsilon) \cdot l_{\text{opt}}.$$

Note that the approximation factor of $(1.5+\epsilon)l_{\text{opt}}$ obtained by our algorithm almost matches the lower bound given in the next section.

4 Lower Time Bound

We give an $\Omega(n \log n)$ worst-case time lower bound for approximating the MLCP problem for unsorted edges within a factor less than 1.5 by a reduction from the set disjointness (SD) problem.

The SD Problem

Input: Two sets S_1 and S_2 of real numbers.

Output: Whether $S_1 \cap S_2$ is empty.

The SD problem has a $\Omega(n \log n)$ lower bound in the algebraic computation tree model [1,2,6,5].

To reduce the SD problem to the MLCP problem, we transform the sets S_1 and S_2 into sets of (unit length) edges that are θ, $(180 + \theta)$ degrees, respectively, clockwise from the upward half-line incident to a vertex v (see Figure 3). We then run the optimal MLCP algorithm on these edges. To ensure that it always finds a solution, we add three dummy edges e_d, e_d^*, e_d^{**} as shown in Figure 3.

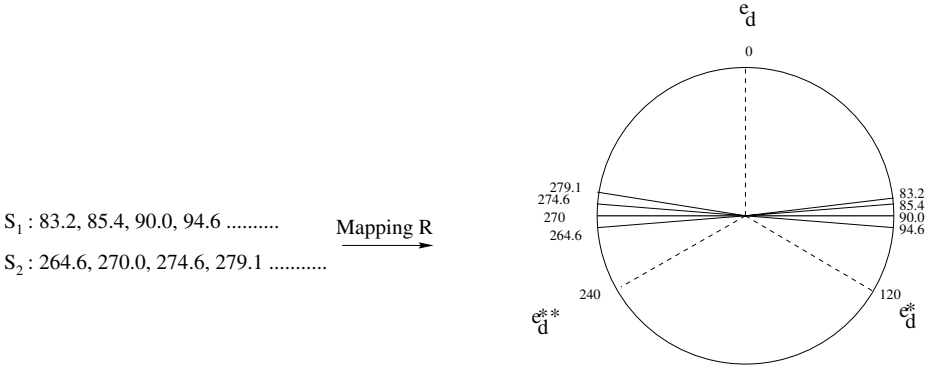


Fig. 3. Reducing the SD to the MLCP problem

A solution of length 2 indicates that the set intersection is non-empty (i.e., there exist two edges, $e \in S_1$ and $e^* \in S_2$, which are collinear). A solution of length 3 indicates that the set intersection is empty. There are cases where the optimum solution is of length 2 and the approximate MLCP algorithm yields a solution with length 3. In this case we have a 1.5-approximation. Hence in the worst case, the length of the non-optimal solution can never be < 1.5 times the length of the optimal solution.

As a consequence we deduce that if we could solve the MLCP problem in $o(n \log n)$ time with an approximation factor < 1.5, we could solve the SD problem in $o(n \log n)$ time. However, this contradicts the known lower time bound of the SD problem. We thus obtain the following theorem:

Theorem 5. *Solving the MLCP problem with an approximation factor* < 1.5 *takes* $\Omega(n \log n)$ *time in the worst case in the algebraic computation tree.*

References

1. M. Ben-Or. Lower Bounds for Algebraic Computation Trees. *Proc. ACM Symp. Th. Comp.*, 80–86. ACM Press, New York, NY, USA 1983
2. M. Ben-Or. Algebraic Computation Trees in Characteristic $p > 0$. *Proc. IEEE Symp. Found. Comp. Sci.*, 534–539. IEEE Press, Piscataway, NJ, USA 1994

3. M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time Bounds for Selection. *Journal of Computer and System Sciences*, 7(4):448–461. Academic Press, San Diego, CA, USA 1973
4. M. Grantson. *Fixed-Parameter Algorithms and Other Results for Optimal Convex Partitions.* Licentiate thesis, LU-CS-TR:2004-231, ISSN 1650-1276 Report 152. Lund University, Sweden 2004
5. D. Grigoriev. Randomized Complexity Lower Bounds. *Proc. ACM Symp. Th. Comp.*, 219–223. ACM Press, New York, NY, USA 1998
6. D. Grigoriev, M. Karpinski, F. Meyer auf der Heide, and R. Smolensky. A Lower Bound for Randomized Algebraic Decision Trees. *Computational Complexity* 6(4):357–375. Birkhäuser Verlag, Basel, Switzerland 1997
7. E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On Two-Dimensional Data Organization, Part I. *Fundaments Informaticae* 2:211–226. Polish Mathematical Society, Warsaw, Poland 1979
8. A. Lubiw. The Boolean Basis Problem and How to Cover Some Polygons by Rectangles. *SIAM Journal on Discrete Mathematics* 3:98–115. Society of Industrial and Applied Mathematics, Philadelphia, PA, USA 1990
9. D. Moitra. Finding a Minimum Cover for Binary Images: An Optimal Parallel Algorithm. *Algorithmica* 6:624–657. Springer-Verlag, Heidelberg, Germany 1991
10. D. Plaisted and J. Hong. A Heuristic Triangulation Algorithm. *Journal of Algorithms* 8:405–437 Academic Press, San Diego, CA, USA 1987

I/O-Efficiently Pruning Dense Spanners

Joachim Gudmundsson[1] and Jan Vahrenhold[2]

[1] NICTA*, Sydney, Australia
joachim.gudmundsson@nicta.com.au
[2] Westfälische Wilhelms-Universität Münster,
Institut für Informatik, 48149 Münster, Germany
jan@math.uni-muenster.de

Abstract. Given a geometric graph $G = (S, E)$ in $\mathbb{R}^d$ with constant dilation t, and a positive constant ε, we show how to construct a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges using $\mathcal{O}(\text{sort}(|E|))$ I/O operations.

1 Introduction

Complete graphs represent ideal communication networks but they are expensive to build; sparse spanners represent low cost alternatives. The number of edges of the spanner network is a measure of its sparseness; other sparseness measures include the weight, the maximum degree, and the number of Steiner points. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and many other areas, and have been subject to considerable research [2,6,12,16,25]. Recently spanners found interesting practical applications in areas such as metric space searching [29,30] and broadcasting in communication networks [3,26].

Consider a set S of n points in the Euclidean space $\mathbb{R}^d$. Throughout this paper, we will assume that d is constant. A network on S can be modeled as an undirected graph G with vertex set S and with edges $e = (u, v)$ of weight $wt(e)$. We will study Euclidean networks, which are geometric networks where the weight of the edge $e = (u, v)$ is equal to the Euclidean distance $|uv|$ between its two endpoints u and v. If G is a geometric graph, then $\delta_G(p, q)$ denotes the Euclidean length of a shortest path in G between p and q. Hence, G is a *t-spanner* for S if $\delta_G(p, q) \leq t|pq|$ for any two points p and q of S. The minimum value t such that G is a t-spanner for S is called the *dilation* of G. A subgraph G' of G is a t'-spanner of G, if $\delta_{G'}(p, q) \leq t' \cdot \delta_G(p, q)$ for any two points p and q of S.

Many algorithms are known that compute t-spanners with $\mathcal{O}(|S|)$ edges that have additional properties such as bounded degree, small spanner diameter (i.e., any two points are connected by a t-spanner path consisting of only a small number of edges), low weight (i.e., the total length of all edges is proportional to the weight of a minimum spanning tree of S), and fault-tolerance; see, e.g.,

* National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 106–116, 2005.

[2,6,7,8,12,15,16,17,21,24,25,31,34], and the surveys [18,32]. All these algorithms compute t-spanners for any given constant $t > 1$. Chen *et al.* [13] showed that the lower bound for computing any t-spanner for a given set S of points in $\mathbb{R}^d$ is $\Omega(|S| \log |S|)$ in the algebraic computation tree model.

For the analysis in this paper we use the standard two-level I/O model [1] which defines the following parameters:

$$N = \#\text{ of objects in the problem instance,}$$
$$M = \#\text{ of objects fitting in internal memory,}$$
$$B = \#\text{ of objects per disk block,}$$

where $N \gg M$ and $1 \leq B \leq M/2$. An *input/output operation* (or simply *I/O*) consists of reading a block of contiguous elements from disk into internal memory or writing a block from internal memory to disk. Computations can only be performed on objects in internal memory. This model of computation captures the characteristics of working with massive data sets that are too large to fit into main memory and thus are stored on disk. Examples of massive graphs include the "web graph", telecommunication networks, or social networks [9,19].

In the two-level I/O model, we measure the efficiency of an algorithm by the number of I/Os it performs, the amount of disk space it uses (in units of disk blocks), and the internal memory computation time. Aggarwal and Vitter [1] developed matching upper and lower I/O bounds for a variety of fundamental problems such as sorting and permuting. For example, they showed that sorting N items in external memory requires $\Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os while scanning N items in external memory obviously can be done in $\Theta(\frac{N}{B})$ I/Os. The upper bounds for sorting and for scanning N items are often abbreviated as $\mathcal{O}(\text{sort}(N)) = \mathcal{O}(\frac{N}{B} \log_{M/B} \frac{N}{B})$ and as $\mathcal{O}(\text{scan}(N)) = \mathcal{O}(\frac{N}{B})$, and we will use these notations throughout this paper.

I/O-efficient algorithms have been developed for several problem domains, including computational geometry, graph theory, and string processing. The practical merits of the developed algorithms have been explored by a number of authors. General recent surveys can be found in [4,35], and there are also more specific surveys that consider I/O-efficient graph algorithms [23,33]. Results related to I/O-efficiently constructing (planar) spanners for point sets, sometimes allowing Steiner points and/or respecting polygonal obstacles in the plane, have been obtained by several authors [20,27,28].

In this paper we consider the problem of I/O-efficiently *pruning* a given t-spanner, even if it has a super-linear number of edges. That is, given a geometric graph $G = (S, E)$ in $\mathbb{R}^d$ with constant dilation t, and a positive constant ε, we consider the problem of constructing a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges.[1]

In the internal memory model two algorithms are known to prune a given t-spanner in time $\mathcal{O}(|E| \log |S|)$. The greedy algorithm of [16,21] can be used to compute a $(1+\varepsilon)$-spanner G' of G. However, efficient implementations of the greedy algorithm are very complex. In [21], the edge set is partitioned into a

[1] The constants implicit in the "Big-Oh" notation depend on $1/\varepsilon^d$.

logarithmic number of sets that are processed in phases. In each phase a cluster cover and a cluster graph is computed by running Dijkstra's algorithm in parallel from all the cluster centers. A simpler approach was presented in [22], using the well-separated pair decomposition, that produces a $(1+\varepsilon)$-spanner G' of G with $\mathcal{O}(|S|)$ edges. The I/O-efficient algorithm presented in this paper is inspired by the latter algorithm. More specifically, given a geometric graph $G = (S, E)$ in $\mathbb{R}^d$ with constant dilation t, and a positive constant ε, we show how to I/O-efficiently construct a $(1+\varepsilon)$-spanner of G with only $\mathcal{O}(|S|)$ edges using $\mathcal{O}(\mathrm{sort}(|E|))$ I/Os. This bound matches the (internal memory) complexity of the algorithm in [22].

While building a sparse spanner is asymptotically faster than pruning a dense spanner, the latter technique allows to specifically designate edges that should participate and edges that are not allowed in the sparse spanner to be constructed.

2 Preliminaries

Our algorithm is similar to the internal memory algorithm by Gudmundsson *et al.* [22] in that it uses the well-separated pair decomposition to decide which edges that can be pruned. We briefly review their algorithm in this section.

In [22] it was shown that a simple way of pruning an existing t-spanner G into a $(1+\varepsilon)$-spanner of G with only $\mathcal{O}(|S|)$ edges is to use the well-separated pair decomposition (WSPD). For completeness we include a description of the WSPD.

Definition 1. *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are well-separated with respect to s if there are two disjoint balls C_A and C_B, having the same radius, such that C_A contains A and, C_B contains B, and the distance between C_A and C_B is at least s times the radius of C_A. We refer to s as the separation ratio.*

Definition 2 ([11]). *Let S be a set of points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A well-separated pair decomposition (WSPD) for S with respect to s is a sequence $\{A_i, B_i\}, 1 \leq i \leq m$, of pairs of non-empty subsets of S, such that*

1. *$A_i \cap B_i = \emptyset$ for all $i = 1, \ldots, m$,*
2. *for each unordered pair $\{p, q\}$ of distinct points of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $q \in A_i$ and $p \in B_i$,*
3. *A_i and B_i are well-separated with respect to s for all $i = 1, \ldots, m$.*

The integer m is called the size of the WSPD. Callahan and Kosaraju show that a WSPD of size $m = \mathcal{O}(|S|)$ can be computed in $\mathcal{O}(|S| \log |S|)$ time. Their algorithm uses a binary tree T, called the *split tree*. We briefly describe the main idea. They start by computing the bounding box of S, which is successively split by d-dimensional hyperplanes, each of which is orthogonal to one of the axes. If a box is split, they take care that each of the two resulting boxes contains at

least one point of S. As soon as a box contains exactly one point, the process stops (for this box). The resulting binary tree T stores the points of S at its leaves; one leaf per point. Also, each node u of T is associated with a subset of S. We denote this subset by S_u; it is the set of all points of S that are stored in the subtree of u.

The split tree T can be computed in $\mathcal{O}(|S| \log |S|)$ time. Callahan and Kosaraju show that, given T, a WSPD of size $m = \mathcal{O}(|S|)$ can be computed in $\mathcal{O}(|S|)$ time. Each pair $\{A_i, B_i\}$ in this WSPD is represented by two nodes u_i and v_i of T, i.e., we have $A_i = S_{u_i}$ and $B_i = S_{v_i}$.

Even though $\sum_{i=1}^{\ell}(|A_i| + |B_i|)$ can be quadratic in $|S|$, it was shown by Callahan [10] that $\sum_{i=1}^{\ell} \min(|A_i|, |B_i|) = \mathcal{O}(|S| \log |S|)$.

Theorem 1 ([11]). *Let S be a set of points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A WSPD for S with respect to s having size $\mathcal{O}(s^d|S|)$ can be computed in $\mathcal{O}(|S| \log |S| + s^d|S|)$ time.*

Now, assume that we are given a t-spanner $G = (S, E)$. Compute a WSPD $\{A_i, B_i\}$, $1 \leq i \leq m$, for S, with separation ratio $s = 4(1 + (1 + \varepsilon)t)/\varepsilon$ and $m = \mathcal{O}(|S|)$. Let $G' = (S, E')$ be the graph that contains for each i, exactly one (arbitrary) edge (x_i, y_i) of E with $x_i \in A_i$ and $y_i \in B_i$, provided such an edge exists. It holds that G' is a $(1 + \varepsilon)$-spanner of G [22], and hence:

Fact 1 (Theorem 3.1 in [22]). *Given a real constant $\varepsilon > 0$ and a t-spanner $G = (S, E)$, for some real constant $t > 1$, one can compute a $(1+\varepsilon)$-spanner G' of G with $\mathcal{O}(|S|)$ edges in time $\mathcal{O}(|E| \log |S|)$.*

WSPD in External Memory. Govindarajan *et al.* [20] showed how to compute a split tree and the well-separated pair decomposition I/O-efficiently.

Fact 2 (Theorem 1 in [20]). *Given a set P of N points in $\mathbb{R}^d$ and a separation constant $s > 0$, a well-separated pair decomposition for P can be computed in $\mathcal{O}(\mathrm{sort}(N))$ I/Os using $\mathcal{O}(N/B)$ blocks of external memory.*

In the process they build a split tree T of P. The idea is to construct T recursively. They construct a partial split tree T' whose leaves have size $\mathcal{O}(N^\alpha)$ for some constant $1 - \frac{1}{6d} \leq \alpha < 1$. Then recursively build the split tree for the leaves, proceeding with an optimal internal memory algorithm for every leaf whose size is at most M. When the split tree has been computed they simulate the internal memory algorithm by Callahan and Kosaraju [11] in external memory by applying time-forward processing on the computation trees.

3 Tree-Labeling Techniques

Pruning algorithms based on the concept of well-separated pairs need to identify, for each edge, the unique pair of a well-separated pair decomposition separating the endpoints of the edge. To do so, the internal memory algorithm

of Gudmundsson *et al.* [22] answers a number of ancestor queries in the split tree corresponding to the well-separated pair decomposition. Mimicking this behaviour, i.e. asking queries one-by-one, in the external memory seems to lead to a prohibitive $\Omega(|S| \log_B |S|/B)$ I/O-complexity, and thus we will reformulate the pruning algorithm in terms of orthogonal range searching where the query ranges correspond to well-separated pairs.

As each component of a well-sepatated pair can be seen as the union of points stored in some subtree, we introduce a labeling of the tree in which each subtree is labeled with (integers from) some interval in $[1 \ldots |S|]$. Exploiting the specific properties of the labeling, we also establish a natural mapping from well-separated pairs to query ranges (Section 4).

The following three lemmas demonstrate the specifics of the labeling scheme and show that any tree can be labeled I/O-efficiently in a hierarchical manner.

Lemma 1. *Given a tree T with N nodes, we can label all $\mathcal{O}(N)$ leaves in left-to-right order in $\mathcal{O}(\mathrm{sort}(N))$ I/Os.*

Proof. We first compute an Euler-Tour for T using the algorithm of Chiang *et al.* [14] and, based upon this tour, we compute a labeling of the nodes according to the BFS-levels of T. The overall process takes $\mathcal{O}(\mathrm{sort}(N))$ I/Os. Then, we again traverse T according to the Euler-Tour and, observing that leaves correspond to local minima of the BFS-level labeling, we can identify and label them during this traversal. The correctness of the left-to-right labeling follows from the fact that each node in the tree is visited in pre-order and thus, each node is visited before its right sibling. As the cost for the traversals is dominated by the cost for computing the Euler-Tour, the overall complexity of labeling the leaves is $\mathcal{O}(\mathrm{sort}(N))$ I/Os. □

Lemma 2. *Given a tree T with N nodes whose leaves are labeled in left-to-right order, we can label each internal node v with an interval $[l_v, r_v]$, $l_v, r_v \in \mathbb{N}$, such that the following holds:*

1. *Each leaf in the subtree rooted at v is labeled with some integer $\ell(v) \in [l_v, r_v]$.*
2. *There exists at least one leaf in the subtree rooted at v that is labeled with an integer $\ell(v) \in [l_v, r_v]$.*
3. *The interval $[l_v, r_v]$ is the minimal interval having this property.*

The I/O-cost for computing this labeling is in $\mathcal{O}(\mathrm{sort}(N))$.

Proof. We prove Lemma 2 by giving an algorithm with an I/O-complexity $\mathcal{O}(\mathrm{sort}(N))$ and showing that it computes a labeling with the desired properties.

The approach of this algorithm is to label the tree bottom-up and to assign to each internal node the minimal interval encompassing the intervals assigned to its children. For the "base case" of our algorithm we transform the label $\ell(v)$ assigned to a leaf v into an interval $[\ell(v), \ell(v)]$. This labeling obviously conforms with requirements of Lemma 2.

To propagate these levels upwards, we first sort the nodes of the tree according to their BFS-level in decreasing order and also label each node with its

BFS-level, its BFS-number, and the BFS-number of its parent. Computing the BFS-level, the BFS-number, and the parents' BFS-number for each node can be done using Euler-Tour techniques in $\mathcal{O}(\text{sort}(N))$ I/Os. Starting with i set to the maximum BFS-level. We then repeatedly extract all nodes on BFS-level i and $i-1$ from the sorted array. We sort all nodes on BFS-level i according to the BFS-number of their parent and sort all nodes on BFS-level $i-1$ according to their BFS-number. We then simultaneously scan both arrays and update each node v on BFS-level $i-1$ with the minimum interval encompassing the intervals assigned to the nodes on BFS-level i having v as their parent (i.e. v's children).

Inductively, we see that the correctness of the labeling follows from the correctness of the labeling on leaf level. The overall complexity is $\mathcal{O}(\text{sort}(N))$ I/Os as the algorithm performs a constant number of Euler-Tour computations and as each node participates in a constant number of sorting steps. □

Observation 1. *The bounds and properties derived in Lemma 1 and Lemma 2 also hold if each leaf v is labeled with an interval $[l_v, r_v]$ such that all these intervals are disjoint and the interval endpoints l_v are assigned to the leaves in increasing order from left to right.*

Note that the labeling proposed in Observation 1 can be applied to a split tree T built for the vertices of a geometric graph $G = (S, E)$. In this setting, the vertices in S are labeled according to the left-to-right order in which they appear in the leaves of T. The process described in Lemma 1 together with Observation 1 then implies a relabeling of the graph's vertices, i.e., each vertex $s \in S$ is labeled with a unique integer in $\ell(s) \in [1 \dots |S|]$. The following lemma shows that this labeling can be mapped to the edges in an I/O-efficient way.

Lemma 3. *Given a unique relabeling of the vertices of a geometric graph $G = (S, E)$, we can relabel the edges in E such that each edge $e = (v, w) \in E$ is labeled $(\ell(v), \ell(w))$ where $\ell(v), \ell(w) \in [1 \dots |S|]$ are the unique labels assigned to v and w. Given the set E of edges and a tree storing the labeled vertices in its leaves, we can relabel all edges in $\mathcal{O}(\text{sort}(|E|))$ I/Os.*

Proof. To relabel the edge, we first extract the labeled vertices from the tree using Euler-Tour techniques in $\mathcal{O}(\text{sort}(|S|))$ I/Os and sort them according to their original label. We then sort all edges according to the (original) label of their respective source vertices, and in a synchronized scan over both sorted lists, we can relabel the source vertices of all edges. Finally, we repeat this process for the list of edges sorted according to the (original) labels of their respective target vertices and obtain a relabeling of the target vertices. The above algorithm clearly runs in $\mathcal{O}(\text{sort}(|S| + |E|)) = \mathcal{O}(\text{sort}(|E|))$ time. □

4 An Algorithm for Pruning Dense Spanners

We are now ready to describe our algorithm for I/O-efficiently pruning a dense t-spanner $G = (S, E)$ such that the resulting graph is a $(1 + \varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges.

Our algorithm first computes a well-separated pair decomposition $\{A_i, B_i\}$ with separation ratio $s = 4(1+(1+\varepsilon)t)/\varepsilon$, using the algorithm of Govindarajan *et al.* [20] and spending an overall number of $\mathcal{O}(\text{sort}(|S|))$ I/Os. The well-separated pair decomposition is represented by a split tree having $\mathcal{O}(|S|)$ leaves which is laid out on disk in $\mathcal{O}(|S|/B)$ disk blocks.

We then use the technique presented in the proof of Lemma 1 to label all *vertices* stored in the leaves from left to right and to label each leaf v with the minimal interval containing the labels of the points stored with v, that is we assign to each vertex v of the graph an unique integer $\ell(v) \in [1 \ldots |S|]$. Finally, we perform a labeling of the internal nodes that fulfills the requirements of Lemma 2. By Lemma 1 and Lemma 2 the complexity computing this labeling is $\mathcal{O}(\text{sort}(|S|))$.

Lemma 4. *The above labeling of the nodes in the split tree has the property that each component C of a well-separated pair $\{A_i, B_i\}$ corresponds to an interval $[l(C), r(C)], C \in \{A_i, B_i\}$, and that the points whose labels fall into $[l(C), r(C)]$ are exactly the members of the component C.*

Proof. Fix a component C of a given well-separated pair $\{A_i, B_i\}$. By definition of the split tree, there exist nodes w_{A_i} and w_{B_i} corresponding to the components of this well-separated pair. Let $v \in \{w_{A_i}, w_{B_i}\}$ be one of these nodes, and let $[l_v, r_v]$ be the interval assigned to v by the algorithm given in the proof of Lemma 2. This algorithm guarantees that there exists at least one point that is stored in the subtree rooted at v whose label falls into $[l_v, r_v]$, and that all other points stored in this subtree are also labeled with an integer $\ell \in [l_v, r_v]$. By the definition of the split tree, the points stored in the subtree rooted at v are exactly the points in the component of $\{A_i, B_i\}$ corresponding to v. For the reverse inclusion assume that there exists a point p that is not stored in the subtree rooted at v and whose label $\ell(p)$ also falls into $[l_v, r_v]$. As the points are labeled according to the left-to-right order of the leaves (see Observation 1), this means that the labels of points in the subtree rooted at v are either all less than or greater than $\ell(p)$. Assume that they are all less than $\ell(p)$. Let $\ell_{\max}$ be the maximum label of all elements in the subtree rooted at v. Then the labels of all elements in the subtree rooted at v are contained in $[l_v, \ell_{\max}]$. As $\ell_{\max} < \ell(p) \leq r_v$, we derive a contradiction to the minimality of $[l_v, r_v]$ (see Lemma 2). This completes the proof. □

The algorithm of Gudmundsson *et al.* [22] prunes a dense spanner by only keeping one edge connecting the two components of each well-separated pair $\{A_i, B_i\}$ considered. Based upon Lemma 4, we can restate this pruning processes as a special case of the range-reporting problem. We first identify each edge $e = (v, w)$ in the original spanner with a point $p_e := (\ell(v), \ell(w)) \in [1 \ldots |S|]^2$ (see Lemma 3). Using this terminology, we can derive a corollary to Lemma 4:

Corollary 1. *Let T be a split tree for $G = (S, E)$ whose nodes have been labeled with intervals according to Lemma 2 and let a and b two nodes of T that correspond to a well-separated pair $\{A_i, B_i\}$. An edge $e = (v, w) \in E$ connects two vertices $v \in A_j$ and $w \in B_i$ if and only if $\ell(v) \in [l_a, r_a]$ and $\ell(w) \in [l_b, r_b]$.*

Let the set $\mathcal{E}$ be defined as $\mathcal{E} := \{(\ell(v), \ell(w)) \in [1 \dots |S|]^2 \mid (v, w) \in E\}$. The above corollary allows us to perform the pruning algorithm for each well-separated pair $\{A_i, B_i\}$ corresponding to two nodes a and b in the split tree by performing an orthogonal range reporting query with query range $[l_a, r_a] \times [l_b, r_b]$ on the set $\mathcal{E}$ while reporting exactly one point. Except for the edge corresponding to the point reported, all edges connecting points in A_i and B_i can be pruned, and this implies that the pruned spanner consists exactly of all edges corresponding to the results of all range queries.

What remains to show is that all range reporting queries can be performed I/O-efficiently. First of all, note that constructing the set $\mathcal{E}$ from the set E of edges can be done in $\mathcal{O}(\text{sort}(|E|))$ I/Os using the algorithm described in the proof of Lemma 3. In a similar way, we can construct the query ranges $[l_a, r_a] \times [l_b, r_b]$ for all pairs $\{A_i, B_i\}$ in the well-separated pair decomposition: We extract the labels of all nodes in the split tree and use two successive sort-merge steps to generate the set $\mathcal{Q}$ of $\mathcal{O}(|S|)$ query ranges in $\mathcal{O}(\text{sort}(|S|))$ I/Os. The next lemma shows that we can I/O-efficiently process all $|\mathcal{Q}|$ range queries while reporting at most a constant number of answers per query.

Lemma 5. *Given a set $\mathcal{Q}$ of orthogonal range queries on a set $\mathcal{E}$ of points in the plane where $|\mathcal{Q}| \in \mathcal{O}(|\mathcal{E}|)$, we can process all queries in $\mathcal{O}(\text{sort}(|\mathcal{E}|))$ I/Os while at the same time reporting no more than two answers per query.*

Proof. We will process all queries in a batched manner using a variant of the algorithm proposed by Arge *et al.* [5]. Their algorithm can answer a set of N queries on a set of $\mathcal{O}(N)$ points in $\mathcal{O}(\text{sort}(N) + K/B)$ I/Os where K is the size of the answer set. Their algorithm utilizes a technique called *distribution sweeping* that combines multi-way divide-and-conquer with a plane-sweeping approach. Roughly speaking, the plane is subdivided into $\Theta(\sqrt{M/B})$ vertical strips each containing approximately the same number of points. Each strip is then swept top-to-bottom, and for each query range spanning a strip, all points inside the range are reported. The algorithm is then applied recursively to (parts of) query ranges falling within one of the strips. As, on each level, each query range appears at most three times (at most one "middle" part spanning one or more strips and at most two "excess" parts falling within a strip) there are $\mathcal{O}(|\mathcal{Q}|) = \mathcal{O}(|\mathcal{E}|)$ query ranges (or parts thereof) on each level, and it can be shown that processing one level can be done in $\mathcal{O}(|\mathcal{E}|/B)$ I/Os. As the recursion tree is of height $\mathcal{O}(\log_{M/B} |\mathcal{E}|/B)$ and is processed top-down, the overall algorithm takes $\mathcal{O}(\text{sort}(\mathcal{E}))$ I/Os *not counting* the I/Os needed to report the answer set.

Unfortunately, we cannot bound the size of the (complete) answer set in our problem setting by a better bound than $\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{E}|)$, and thus we need to modify the algorithm of Arge *et al.* as follows: we first use one top-down-sweep to process the query ranges on the current level of recursion with respect to their "middle" parts and as soon as we find a point p contained in the current query range q, we label q with the name of p, output the answer (q, p), and stop processing q. In a second top-down sweep we distribute the "excess" parts of the query ranges that have not been labeled with an answer to the corresponding sub-problems,

that is we stop processing ranges for which we already have found an answer. As each query range appears in at most two subproblems on each level where it can span a slab, there are at most two answers that can be reported before a query range is not processed anymore, and this in turn results in an answer set of size $\mathcal{O}(|\mathcal{E}|)$. The I/O-complexity of processing one level of recursion is not affected by this modification, hence the overall algorithm runs in $\mathcal{O}(\text{sort}(|\mathcal{E}|))$ I/Os. □

As mentioned above, we run the algorithm described in the proof of Lemma 5 on a point set of size $\mathcal{O}(|E|)$ and a query set of size $\mathcal{O}(|S|)$, and thus we obtain an answer set of size $\mathcal{O}(|S|)$ spending no more than $\mathcal{O}(\text{sort}(|E|))$ I/Os. A simple duplicate-removal step taking $\mathcal{O}(\text{sort}(|S|))$ I/Os then reduces the output to at most one answer per query. The $\mathcal{O}(|S|)$ edges corresponding to the points in the answer set then form the desired pruned spanner [22]. This yields our main result:

Theorem 2. *Given a geometric graph $G = (S, E)$ in $\mathbb{R}^d$ which is a t-spanner for S for some constant $t > 1$ and given a constant $\varepsilon > 0$, we can compute a $(1 + \varepsilon)$-spanner $G' = (S, E')$ of G with $E' \subseteq E$ and $|E'| \in \mathcal{O}(|S|)$ spending $\mathcal{O}(\text{sort}(|E|))$ I/Os.*

The obvious open problem is whether the complexity of I/O-efficiently pruning dense spanners can be improved to $\mathcal{O}(\text{scan}(|E|) + \text{sort}(|S|))$ I/Os.

References

1. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, Sept. 1988.
2. I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
3. K. M. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.
4. L. A. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*. Kluwer, 2002. 313-357.
5. L. A. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. Theory and practice of I/O-efficient algorithms for multidimensional batched searching problems. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 685–694, 1998.
6. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symposium on Theory of Computing*, pages 489–498, 1995.
7. S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.
8. P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry: Theory and Applications*, 28:11–18, 2004.
9. A. L. Buchsbaum and J. R. Westbrook. Maintaining hierarchical graph views. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 566–575, 2000.

10. P. B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications*. Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1995.
11. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.
12. B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *International Journal of Computational Geometry and Applications*, 5:124–144, 1995.
13. D. Z. Chen, G. Das, and M. Smid. Lower bounds for computing geometric spanners and approximate shortest paths. *Discrete Applied Mathematics*, 110:151–167, 2001.
14. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149, 1995.
15. G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Annual ACM Symposium on Computational Geometry*, pages 53–62, 1993.
16. G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry & Applications*, 7:297–315, 1997.
17. G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 1995.
18. D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers, Amsterdam, 2000.
19. S. Eubank, V. A. Kumar, M. V. Marathe, A. Srinivasany, and N. Wang. Structural and algorithmic aspects of massive social networks. In J. I. Munro, editor, *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 718–727, 2004.
20. S. Govindarajan, T. Lukovszki, A. Maheswari, and N. Zeh. I/O-efficient well-separated pair decomposition and its application. In *Proc. 8th European Symposium on Algorithms*, volume 1879 of *Lecture Notes in Computer Science*, pages 220–231. Springer-Verlag, 2000.
21. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.
22. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graph. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.
23. I. Katriel and U. Meyer. Elementary graph algorithms in external memory. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*, pages 62–84. Springer, Berlin, 2003.
24. J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithmic Theory*, pages 208–213, 1988.
25. C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.
26. X.-Y. Li. Applications of computational geometry in wireless ad hoc networks. In X.-Z. Cheng, X. Huang, and D.-Z. Du, editors, *Ad Hoc wireless networking*. Kluwer, 2003.

27. T. Lukovszki, A. Maheshwari, and N. Zeh. I/O-efficient batched range counting and its applications to proximity problems. In *Proc. 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *Lecture Notes in Computer Science*, pages 244–255, Berlin, 2001. Springer.
28. A. Maheshwari, M. Smid, and N. Zeh. I/O-efficient shortest path queries in geometric spanners. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Proc. 7th Workshop on Algorithms and Data Structures*, volume 2125 of *Lecture Notes in Computer Science*, pages 287–299, Berlin, 2001. Springer.
29. G. Navarro and R. Paredes. Practical construction of metric t-spanners. In *5th Workshop on Algorithmic Engineering and Experiments*, pages 69–81. SIMA Press, 2003.
30. G. Navarro, R. Paredes, and E. Chávez. t-spanners as a data structure for metric space searching. In *9th International Symposium on String Processing and Information Retrieval*, volume 2476 of *Lecture Notes in Computer Science*, pages 298–309. Springer, 2002.
31. J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1:99–107, 1991.
32. M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science Publishers, Amsterdam, 2000.
33. L. I. Toma and N. Zeh. I/O-efficient algorithms for sparse graphs. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*, pages 85–109. Springer, Berlin, 2003.
34. P. M. Vaidya. A sparse graph almost as good as the complete graph on points in K dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.
35. J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, June 2001.

On the Minimum Size of a Point Set Containing Two Non-intersecting Empty Convex Polygons

Kiyoshi Hosono and Masatsugu Urabe

Department of Mathematics, Tokai University,
3-20-1 Orido, Shimizu, Shizuoka, 424-8610 Japan

Abstract. Let $n(k,l)$ be the smallest integer such that any set of $n(k,l)$ points in the plane, no three collinear, contains both an empty convex k-gon and an empty convex l-gon, which do not intersect. We show that $n(3,5) = 10$, $12 \leq n(4,5) \leq 14$, $16 \leq n(5,5) \leq 20$.

1 Introduction

Erdős [1] asked the following combinatorial geometry problem in 1979: Find the smallest integer $n(k)$ such that any set of $n(k)$ points in the plane, in general position, i.e., no three points are collinear, contains the vertices of a convex k-gon, whose interior contains no points of the set. Such a subset is called an *empty convex k-gon* or a *k-hole*. Klein [2] found $n(4) = 5$, and $n(5) = 10$ was determined by Harborth [3]. Horton [4] showed that $n(k)$ does not exist for $k \geq 7$. The value of $n(6)$ is still open.

We consider a related problem. Let H_1 and H_2 be a pair of holes in a given point set. We say that H_1 and H_2 are *vertex disjoint* if $H_1 \cap H_2 = \emptyset$. If, moreover, their convex hulls are disjoint, we simply say that the two holes are *disjoint*; $ch(H_1) \cap ch(H_2) = \emptyset$ where ch stands for the convex hull. We study the following function: Let $n(k,l)$ denote the smallest integer such that any set of $n(k,l)$ points in general position in the plane contains a pair of disjoint holes with a k-hole and a l-hole. Clearly, $n(3,3) = 6$ and $n(k,7) = +\infty$. In [5] and [6], we studied the maximum number of pairwaise disjoint k-holes and we found $n(3,4) = 7$ and $n(4,4) = 9$, where Figure 1 gives $n(4,4) \geq 9$. In this paper, we estimate the other values; $n(3,5) = 10$, $12 \leq n(4,5) \leq 14$, $16 \leq n(5,5) \leq 20$.

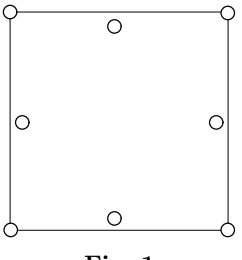


Fig. 1

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 117–122, 2005.

2 Definitions and Notations

In this paper, we consider only planar point sets in general position. For such a set P, we denote $P = V(P) \cup I(P)$ such that $V(P)$ is the boundary vertices on $ch(P)$. We denote a k-hole by $(v_1 v_2 \cdots v_k)$ if the vertices are located with order anti-clockwise. When indexing a set of t points, we identify indices modulo t.

Let a, b and c be any three points in the plane. We denote the *convex cone* between $r(a;b)$ and $r(a;c)$ by $C(a;b,c)$, where $r(a;b)$ is the ray emanating from a and passing through b. For $\delta = b$ or c of $C(a;b,c)$, let δ' be a point collinear with a and δ so that a lies on the segment $\delta\delta'$. A convex region is also said to be *empty* if its interior contains no elements of P. Namely, if $C(a;b,c)$ is not empty, we can consider an element p of P in the interior of $C(a;b,c)$ such that $C(a;b,p)$ is empty. We call such p the *attack point* from $r(a;b)$ to $r(a;c)$.

Let R be a convex region and consider $C(a;b,c)$ such that $\{a,b,c\}$ is contained in R. For $C_R(a;b,c) = C(a;b,c) \cap R$, we define $\alpha_R(a;b,c)$ as the element of P in the interior of $C_R(a;b,c)$ such that $C_R(a;b,\alpha_R(a;b,c))$ is empty. Finally, $H(ab;c)$ or $H(ab;\overline{c})$ denotes the open half-plane bounded by the line ab, containing c or not, respectively.

3 Results

In this section, we estimate the values $n(k,5)$ for $k = 3,4,5$. We first show the following result.

Theorem 1. $n(3,5) = 10$.

Before showing Theorem 1, we propose the next lemma.

Lemma 1. *If a* 10 *point set* P *contains a* 6*-hole* Q*, then* P *has a pair of disjoint holes with a* 3*-hole and a* 5*-hole.*

Proof. Let $Q = (q_1q_2q_3q_4q_5q_6)$ and consider the convex cone Γ_i determined by $r(q_{i+3};q_i)$ and $r(q_{i-1};q_{i+2})$ for $i = 1,3,5$, where $\Gamma_1 \cup \Gamma_3 \cup \Gamma_5$ covers the area $R^2 \backslash ch(Q)$ and each Γ_i contains exactly one point q_{i+1} of Q in the interior. Since Γ_i contains at least two points of $P \backslash Q$ for some i, we obtain a 5-hole $Q' = Q \backslash \{q_{i+1}\}$ and three points in the convex region $\Gamma_i \backslash ch(Q')$. □

Proof of Theorem 1. The lower bound holds by $n(5) = 10$ as shown in Fig. 2.

For the upper bound we show that every 10 point set P has a disjoint pair of a 3-hole and a 5-hole. By $n(5) = 10$ we obtain a 5-hole $F = (p_1p_2p_3p_4p_5)$ in P. Denote the finite or infinite region bounded with $r(p_{i-1};p_i)$, $r(p_{i+2};p_{i+1})$ and the line segment p_ip_{i+1} by B_i for $1 \le i \le 5$. If some B_i contains a point of P, we are done by Lemma 1 since P has a 6-hole. Suppose that every B_i is empty.

We claim that there are consecutive vertices, say p_3 and p_4 in F such that $\angle p_2p_3p_4 + \angle p_3p_4p_5 > \pi$. Then we consider three convex regions $R_1 = H(p_4p_5;\overline{p_1})$, $R_2 = H(p_2p_3;\overline{p_1}) \backslash R_1$ and $R_3 = C(p_1;p_2',p_5') \backslash (R_1 \cup R_2)$. If some R_i contains at

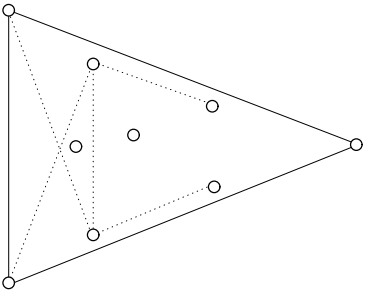


Fig. 2

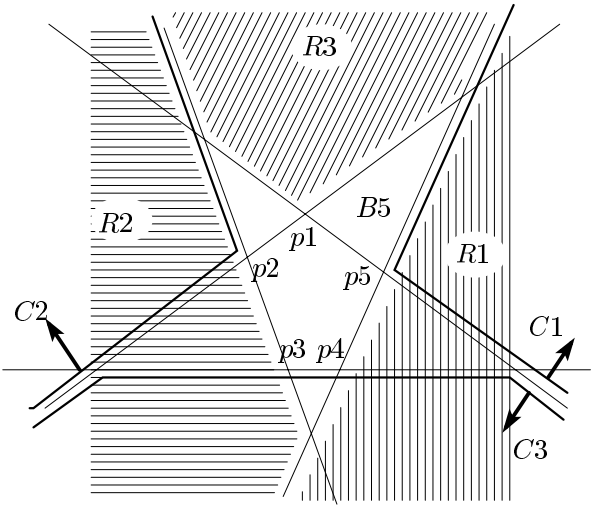


Fig. 3

least three points, we have a triangle in this region and a 5-hole F. We can suppose that each R_i is not empty and contains one or two points.

We first suppose that R_3 contains exactly two points. Then we can assume that $C_1 = C(p_5; p'_1, p'_4)$ is empty since, otherwise, the convex region $R_3 \cup C_1 \cup B_5$ contains at least three points. Similarly, $C_2 = C(p_2; p'_1, p'_3)$ is also empty, which implies that $C_3 = H(p_3p_4; \overline{p_1}) \backslash (C_1 \cup C_2)$ contains exactly three points. Then we have a triangle in C_3 and a 5-hole F. See Fig. 3.

Suppose that R_3 contains exactly one point. If C_1 or C_2 contains at least two points, we are immediately done. Then we suppose that C_3 has exactly two points and that each of C_1 and C_2 contains exactly one point. Moreover, $H(p_3p_4; \overline{p_1}) \cap (C_1 \cup C_2)$ can be assumed to be empty. Then if $C(p_4; p_3, p'_5)$ is empty, R_1 contains exactly three points. Therefore, we can assume that each of $C(p_4; p_3, p'_5)$ and $C(p_3; p_4, p'_2)$ contains exactly one point by symmetry.

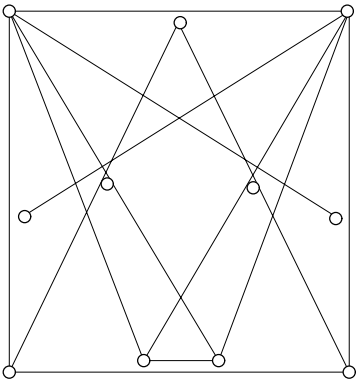


Fig. 4

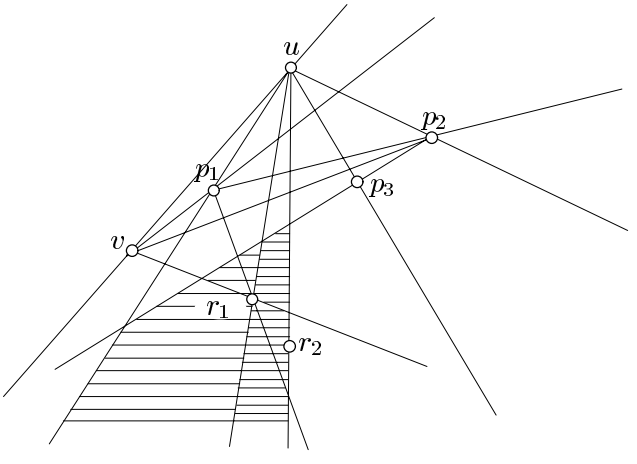


Fig. 5

Let q_1, q_2 or q_3 be in R_3, C_2 or $C(p_4; p_3, p_5')$, respectively. If both q_1 and q_3 are in $H(p_1p_3; p_4)$, a 6-hole $(q_1p_1p_3q_3p_4p_5)$ appears and we are done. If either q_1 or q_3 is in $H(p_1p_3; p_4)$, we obtain a 5-hole and a 3-hole in $H(p_1p_3; p_2)$. Hence, the remaining case is that $H(p_1p_3; \overline{p_4})$ contains both q_1 and q_3. If $\triangle q_1q_2q_3$ contains p_2, we obtain a 5-hole $(q_1p_2q_3p_3p_1)$ and a 3-hole in $H(p_1p_3; p_4)$. If not so, we obtain F itself and $\triangle q_1q_2q_3$. □

Theorem 2. $12 \leq n(4, 5) \leq 14$.

Proof. Figure 4 gives an 11 point set which does not contain both a 4-hole and a 5-hole, following that $n(4, 5) \geq 12$.

For a 14 point set P, we take any edge uv of $ch(P)$ and consider the point p_1 with $\{u, v, p_1\}$ anti-clockwise order such that $C(u; v, p_1)$ is empty. If $C(p_1; u, v')$ is not empty, we obtain a 4-hole $Q = (uvp_1\alpha(p_1; u, v'))$. Since $C(p_1; u', \alpha(p_1; u, v'))$

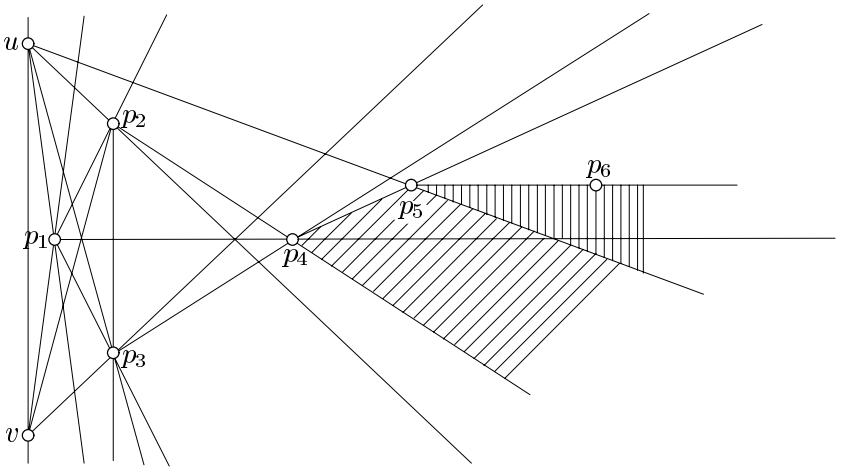


Fig. 6

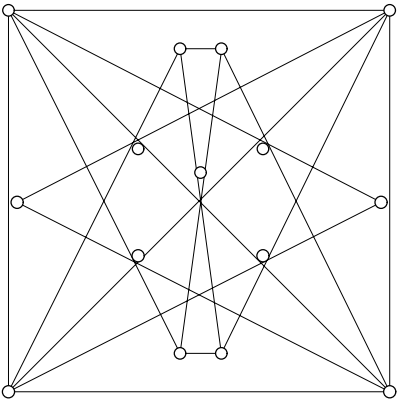


Fig. 7

contains precisely 10 points in the interior, we also obtain a 5-hole in it by $n(5) = 10$.

Suppose that $C(v; u, p_1)$ is empty and let $p_2 = \alpha(p_1; v', u')$. We can assume that $\triangle vp_2p_1$ is empty since, otherwise, we obtain a 4-hole $(p_2up_1\alpha(p_2; p_1, v))$ and $H(\alpha(p_2; p_1, v)p_2; v)$ contains 10 points. Then since, if $C(p_2; v, u')$ is empty, we obtain a 4-hole $(p_2p_1v\alpha(p_2; u', p_1'))$ and $H(p_2\alpha(p_2; u', p_1'); u)$ contains 10 points, we consider $p_3 = \alpha(p_2; v, u')$.

We can assume that $C(p_3; p_2', u')$ is empty. In fact, if this region contains a unique interior point q, we obtain a 4-hole (vqp_3p_1) and $H(up_3; p_2) \cup \{u\}$ with 10 points. If this contains at least two points, we consider two attack points of $r_1 = \alpha(u; p_1, p_3)$ and $r_2 = \alpha(u; r_1, p_3)$. If $\triangle p_1vr_2$ contains r_1, we have a 5-hole

$(up_1r_1r_2p_3)$ or $(up_1r_1r_2\alpha(r_2;u,p_3))$, respectively if $C(r_2;u,p_3)$ is emty or not. Then since $H(r_2p_3;p_2)$ or $H(r_2\alpha(r_2;u,p_3);p_2)$ contains 8 points, it has a 4-hole by $n(4)=5$. See Fig. 5 where the shaded portions are empty. For otherwise, we have a 4-hole of either $(r_1r_2p_1v)$ or $(r_2r_1p_1v)$ and $H(ur_2;p_3)\cup\{u\}$ with 10 points.

We now suppose that $C(p_3;u',p_1')$ is empty since, otherwise, we obtain a 4-hole $(p_3p_1v\alpha(p_3;u',p_1'))$ and $H(p_3\alpha(p_3;u',p_1');u)$ with 10 points. We can assume that $C(p_2;p_3,u')$ is empty since, otherwise, we have a 5-hole $(p_2up_1p_3\alpha(p_2;p_3,u'))$ and $H(p_2\alpha(p_2;p_3,u');\overline{u})$ with 8 points. By symmetry, $C(p_3;p_2,v')$ is also empty. For $p_4=\alpha(p_3;v',p_1')$, we suppose that $C(p_2;p_4,u')$ is empty since, otherwise, we obtain a 5-hole $(p_4p_2p_1p_3\alpha_R(p_4;p_3,p_2'))$ where $R=C(p_1;p_2,p_3)$ and $H(p_4\alpha_R;\overline{p_1})$ with 7 or 8 points.

We consider $p_5=\alpha(p_4;p_3',p_1')$ by symmetry. If $\alpha_{R'}(p_5;p_4,u')$ exists for $R'=C(p_2;p_4,u)$, we obtain a 5-hole $(p_5up_2p_4\alpha_{R'})$ and $H(p_5\alpha_{R'};\overline{p_4})$ with at least 6 points. If $C_{R'}(p_5;p_4,u')$ is empty, we obtain a 5-hole $(p_6p_5p_4p_3v)$ for $p_6=\alpha(p_5;u',p_4')$ and $C(p_5;p_6,p_4')$ contains 6 points in the interior as shown in Fig. 6. □

Theorem 3. $16\le n(5,5)\le 20$.

Proof. Figure 7 gives a 15 point set which does not contain two disjoint 5-holes, referred to in [5]. For any set of 20 points, there is a line which halves the set. Then the upper bound holds by $n(5)=10$. □

References

1. Erdős, P.: Some combinatorial problems in geometry. Proceedings Conference University Haifa, Lecture Notes in Mathematics **792** (1980) 46–53
2. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. Compositio Math. **2** (1935) 463–470
3. Harborth, H.: Konvexe Fünfecke in ebenen Punktmengen. Elem. Math. **33** (1978) 116–118
4. D.Horton, J.: Sets with no empty convex 7-gons. Canad. Math. Bull. **26** (1983) 482–484
5. Hosono, K., Urabe, M.: On the number of disjoint convex quadrilaterals for a plannar point set. Comp. Geom. Theory Appl. **20** (2001) 97–104
6. Urabe, M.: On a partition into convex polygons. Disc. Appl. Math. **64** (1996) 179–191

Three Equivalent Partial Orders on Graphs with Real Edge-Weights Drawn on a Convex Polygon

Hiro Ito

Department of Communications and Computer Engineering,
School of Informatics, Kyoto University, Kyoto 606-8501, Japan
itohiro@i.kyoto-u.ac.jp

Abstract. Three partial orders, cut-size order, length order, and operation order, defined between labeled multigraphs with the same order are known to be equivalent. This paper extends the result on edge-capacitated graphs, where the capacities are real numbers, and it presents a proof of the equivalence of the three relations. From this proof, it is also shown that we can determine whether or not a given graph precedes another given graph in polynomial time.

1 Introduction

Let $N = \{x_0, x_1, \ldots, x_{n-1}\}$ be the set of vertices of a convex polygon P in the plane, where the vertices are arranged in this order counter-clockwisely, and hence (x_i, x_{i+1}) is an edge of P for $i = 0, 1, \ldots, n-1$ (We adopt the residue class on n for treating integers in N, i.e., $i \pm j$ is $i' \in N$ such that $i' \equiv i \pm j \pmod{n}$). An internal angle of P may be π. We consider graphs whose node set corresponds to N, i.e., the node set is $\{0, 1, \ldots, n-1\}$ and each node i is assigned to x_i, and each edge $e = (i, j)$ of the graph is represented by a line segment $x_i x_j$.

We adopt the cyclic order for treating integers (or numbered vertices) in N. Thus for $i, j \in N$,

$$[i, j] = \begin{cases} \{i,\ i+1,\ \ldots,\ j\}, & \text{if } i \le j, \\ \{i,\ i+1,\ \ldots,\ n-1,\ 0,\ 1,\ \ldots,\ j\}, & \text{if } i > j; \end{cases}$$

for $i, j, k \in N$, $i \le j \le k$ means $j \in [i, k]$; for $i, j, k, h \in N$, $i \le j \le k \le h$ means that i, j, k, h appear in this order when we traverse the nodes of $[i, h]$ from i to h. For notational simplicity, $\{i\}$ may be written as i. For a graph G, $E(G)$ means the edge set of G.

In this paper all graphs are regarded as weighted graphs, i.e., we introduce a weight function $w_G : E(G) \to \boldsymbol{R}$ and a weighted graph G always has a weight function w_G in this paper.

Three relations, cut-size order, length order, and operation order, were introduced between vertex-labeled graphs in Reference [5] and shown that they are equivalent [4,5]. However, the proof in Reference [5] is for only multigraphs with the same number of edges and without edge weights. The proof for the general case have been appeared in only Technical Notes [4]. This paper shows a new proof, which is more simple than the previous one, for the general case.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 123–130, 2005.

2 Definitions

We introduce some terms as follows.

Linear Cuts. For a graph G and a pair of distinct nodes $i, j \in N$, a *linear cut* $C_G(i, j)$ is an edge set:

$$C_G(i, j) = \{(k, h) \in E(G) \mid k \in [i, j-1],\ h \in [j, i-1]\}.$$

Fig. 1 show examples of linear cuts. The capacity of a linear cut $C_G(i, j)$ is defined as

$$c_G(i, j) = \sum_{e \in C_G(i,j)} w_G(e).$$

For two subsets N' and N'' of nodes,

$$w_G(N', N'') = \sum_{i \in N', j \in N''} w_G(i, j).$$

The *degree* of a node $i \in N$ of a graph G is defined as $c_G(i, i+1) = w_G(i, [i+1, i-1])$ and may be simply denoted by $d_G(i)$. As a generalization of degree, $d_G(N')$ denotes $w_G(N', N-N')$ for $N' \subset N$. From them, $c_G(i, j) = d_G([i, j-1])$, since they means the same thing.

We introduce a relation based on sizes of linear cuts as follows. For two weighted graph G and G', $G \preceq_c G'$ means that $c_G(i, j) \leq c_{G'}(i, j)$ for all $i, j \in N$, $i \neq j$. This relation is known to be a partial order, since it is easily obtained from the following result presented by Skiena [7].

Theorem 1. *For two weighted graphs G and G', if $c_G(i, j) = c_{G'}(i, j)$ for all $i, j \in N$, $i \neq j$, then $G = G'$.* □

Sum of Edge Lengths. For an edge (i, j) of a weighted graph G and a convex n-gon P, let $\mathrm{dist}(i, j)$ be a length of the line segment $x_i x_j$. We define a sum of weighted edge length of G with respect to P as

$$s_P(G) = \sum_{(i,j) \in E(G)} w(i, j) \cdot \mathrm{dist}(i, j).$$

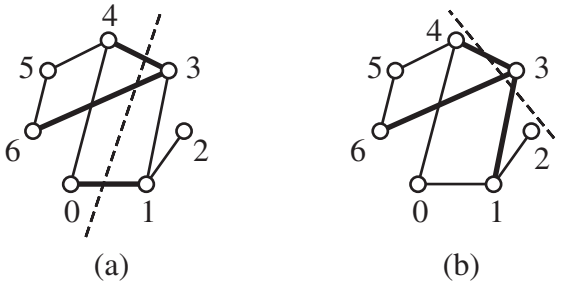


Fig. 1. Linear cuts: (a) $C_G(1, 4)$, (b) $C_G(3, 4)$

We introduce a relation based on the measure as follows. For two weighted graph G and G', $G \preceq_l G'$ means that $s_P(G) \leq s_P(G')$ for all convex n-gons P. Graph drawing is a very important research aea and the sum of edge lengths is a crucial criterion for evaluating drawing methods [1].

Cross-Operations. We introduce an operation transforming a graph to another one. For a weighted graph G, two distinct $i, j \in N$ and a real value Δ, $\mathrm{ADD}_G(i, j; \Delta)$ means adding Δ to $w(i, j)$ (if $(i, j) \notin E(G)$, adding an edge (i, j) to $E(G)$ previously). The reverse operation of ADD can be defined, i.e., $\mathrm{REMOVE}_G(i, j; \Delta)$ means $\mathrm{ADD}_G(i, j; -\Delta)$. We extend these operations in the case $i = j$, i.e., both $\mathrm{ADD}_G(i, i; \Delta)$ and $\mathrm{REMOVE}_G(i, i; \Delta)$ mean doing nothing. For nodes $i, j, k, h \in N$ with $i \leq j \leq k \leq h$ and a positive $\Delta > 0$ (see, Fig. 2), a *cross-operation* $X_G(i, j, k, h; \Delta)$ is applying.

$\mathrm{REMOVE}_G(i, j; \Delta)$, $\mathrm{REMOVE}_G(k, h; \Delta)$, $\mathrm{ADD}_G(i, k; \Delta)$, and $\mathrm{ADD}_G(j, h; \Delta)$.

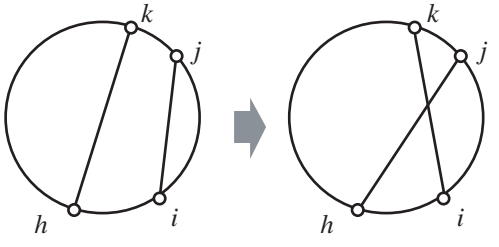


Fig. 2. Cross-operation $X(i, j, k, h; 1)$

If some of $\{i, j, k, h\}$ are equal, a cross-operation may increase edges. In fact, if $i = j < k < h < i$ or $i = j < k = h < i$ (or the cases symmetric with respect to one of them), then the total edge weights increases (see, (a) and (b) of Fig. 3). If $j = k$ or $i = h$, the edge set is not changed (see, (c) and (d) of Fig. 3).

We introduce a relation based on cross-operations as follows. For two weighted graph G and G', $G \preceq_o G'$ means that G' can be obtained from G by applying finite number (including zero) of cross-operations. Cross-operations are very similar to 2-switches, presented by Hakimi [2,3] and developed by West [8]. The only deference between them is that the order of i, j, k, h is not a matter in 2-switches.

3 Equivalence of the Three Relations

We have the following theorem.

Theorem 2. *Three relations $\preceq_c$, $\preceq_l$, and $\preceq_o$ are equivalent.* □

This theorem was shown in [5] for graphs with the same size (number of edges), but for the general case a proof is shown only in Technical Notes [4]. Moreover,

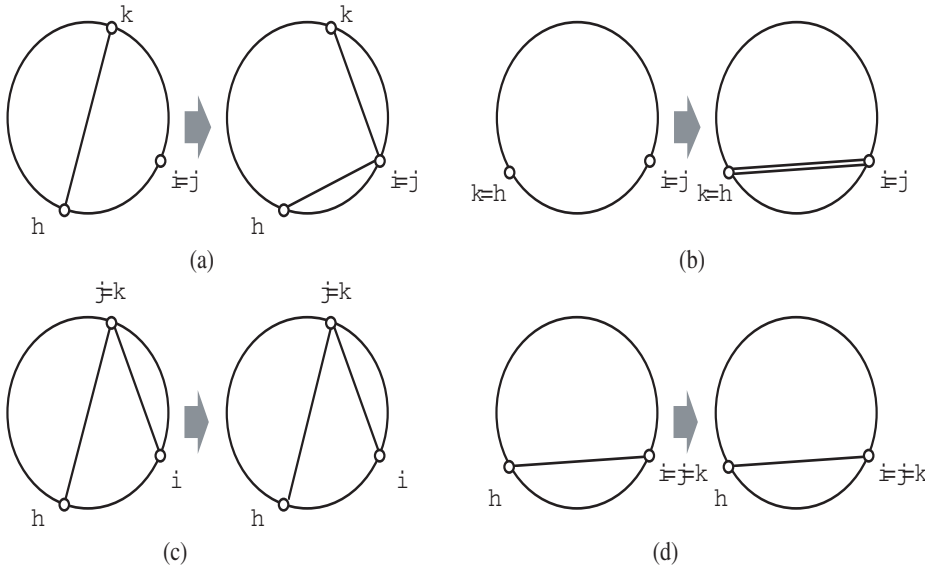


Fig. 3. These cross-operations $X(i, j, k, h; 1)$ when some of nodes are the same

these proofs were a bit long and complicated. We show a more simple proof of this theorem in this section.

In the remaining part of this section, we consider that all weighted graphs are complete graphs without loss of generality, since $(i, j) \notin E$ is equivalent to $w_G(i, j) = 0$. Hence a weighted graph can be represented by a pair of a node set N and a weight function w: $G = (N, w)$. Define a zero weighted graph $G_\emptyset = (N, w_\emptyset)$ as $w_\emptyset(i, j) = 0$ for all $i, j \in N$.

Note that $c_{G_\emptyset}(i, j) = 0$ for any $i, j \in N$ $(i \neq j)$, and $S_P(G_\emptyset) = 0$ for any polygon P. For any pair of $G = (N, w)$ and $G' = (N, w')$, we define $G - G' = (N, w'')$ as $c''(i, j) := c(i, j) - c'(i, j)$ for every $i, j \in N$. $G \preceq G'$ ($\preceq$ is any one of $\preceq_l$, $\preceq_c$, and $\preceq_o$) is equivalent to $G - G' \preceq G_\emptyset$. Therefore, it is enough to consider $G' = G_\emptyset$ for proving Theorem 2, as a result of this fact, the proof of Theorem 2 consists of three parts:

(1) $G \preceq_o G_\emptyset \Rightarrow G \preceq_l G_\emptyset$, (Lemma 1)
(2) $G \preceq_l G_\emptyset \Rightarrow G \preceq_c G_\emptyset$, (Lemma 2) and
(3) $G \preceq_c G_\emptyset \Rightarrow G \preceq_o G_\emptyset$. (Lemma 3)

Lemma 1 ([5]). *If $G \preceq_o G_\emptyset$, then $G \preceq_l G_\emptyset$.*

Proof: It is clear from the triangle inequality. □

Lemma 2 ([5]). *If $G \preceq_l G_\emptyset$, then $G \preceq_c G_\emptyset$.*

Proof. Suppose that $G \preceq_c G_\emptyset$ does not hold, i.e., there are $i, j \in N$ such that $c_G(i, j) > 0$. We construct a polygon P satisfying $S_P(G) > 0$ as follows.

$X = \{x_k \mid k \in [i, j-1]\}$ and $Y = \{x_k \mid k \in [j, i-1]\}$. Let $p, r > 0$ be real numbers. Put all vertices $x_i \in X$ in a circle whose center is $(0, 0)$ and radius is r. Put all vertices $x_i \in Y$ in a circle whose center is $(p, 0)$ and radius is r. We can locate all vertices satisfying the above conditions and convexity for any r and p. By letting p be far larger than r, $S_P(G) > 0$. □

Lemma 3. *If $G \preceq_c G_\emptyset$, then $G \preceq_o G_\emptyset$.*

In this paper we show a new proof, which is more simple than the previous one, for this lemma. The following proposition is well-known. Since the proof is easy, it is omitted.

Proposition 1. *Let $A, B, C \subset N$ be three mutually disjoint subsets and G be a weighted graph, then*

$$d_G(A \cup B) + d_G(B \cup C) = d_G(B) + d_G(A \cup B \cup C) + 2w_G(A, C).$$

□

Proof of Lemma 3. Assume that $G \preceq_c G_\emptyset$, i.e.,

$$d_G([i, j]) = c_G(i, j+1) \leq 0 \text{ for all } i, j \in N. \tag{1}$$

We use an example shown in Fig. 4 (a) for a help of understanding. Let k_0 be the largest integer such that

$$d_G([i, j]) = 0 \text{ for all } (i, j) \in \{(i, j) \mid i, j \in N, j - i < k_0\}. \tag{2}$$

Note that the residue class is used for the difference. (For an example for Fig. 4 (a), $k_0 = 2$ since $d_G(0) = d_G(1) = \cdots = d_G(11) = 0$, $d_G([0, 1]) =$

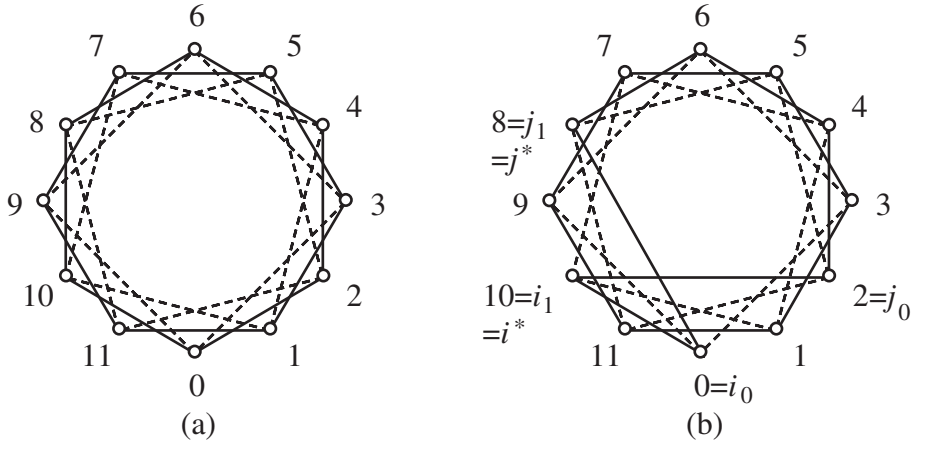


Fig. 4. An example of G: $w(e) = 1$ for solid edges and $w(e) = -1$ for broken edges

$d_G([1,2]) = \cdots = d_G([11,0]) = 0$, and $d_G([0,2]) = -2 < 0$.) If $k_0 \geq \lfloor n/2 \rfloor$, $G = G_\emptyset$. Hence, we assume $k_0 < \lfloor n/2 \rfloor$. Then their exists (i_0, j_0) such that $j_0 - i_0 = k_0$ and

$$d_G([i_0, j_0]) < 0. \tag{3}$$

(For an example for Fig. 4 (a), $i_0 = 0$ and $j_0 = 2$.) By considering Proposition 1 with $A = \{i_0\}$, $B = [i_0 + 1, j_0 - 1]$, and $C = \{j_0\}$, we obtain

$$\begin{aligned} &d_G([i_0, j_0 - 1]) + d_G([i_0 + 1, j_0]) \\ &= d_G([i_0 + 1, j_0 - 1]) + d_G([i_0, j_0]) + 2w_G(i_0, j_0). \end{aligned}$$

Thus

$$\begin{aligned} &w_G(i_0, j_0) \\ &= \frac{d_G([i_0, j_0 - 1]) + d_G([i_0 + 1, j_0]) - d_G([i_0 + 1, j_0 - 1]) - d_G([i_0, j_0])}{2} \\ &> 0, \end{aligned} \tag{4}$$

since $d_G([i_0, j_0 - 1]) = d_G([i_0 + 1, j_0]) = d_G([i_0 + 1, j_0 - 1]) = 0$, and $d_G([i_0, j_0]) < 0$. (In the example, $w_G(0, 2) = 1 > 0$.) Let I be a set of (i, j) $(i, j \in N)$ satisfying the following conditions:

(a) $i < i_0 \leq j_0 < j$, and
(b) $d_G([i', j']) < 0$ for all $i < i' \leq i_0$ and $j_0 \leq j' < j$.

(For an example for Fig. 4 (a), $I = \{(11, 3), (11, 4), \ldots, (11, 9), (10, 3), (10, 4), \ldots, (10, 8), (9, 3), (9, 4), \ldots, (9, 7), (8, 3), (8, 4), (8, 5), (8, 6), (7, 3), (7, 4), (7, 5), (6, 3), (6, 4), (5, 3)\}$.) $I \neq \emptyset$ since $(i_0 - 1, j_0 + 1) \in I$. Let (i_1, j_1) be an extremal element of I, i.e., they satisfies (a), (b), and

(c) there are $i_1 < i_2 \leq i_0$ and $j_0 \leq j_2 < j_1$ such that $d_G([i_1, j_2]) = d_G([i_2, j_1]) = 0$

(For an example for Fig. 4 (a), $i_1 = 10$, $j_1 = 8$, $i_2 = 11$, $j_2 = 7$.) Such i_1, j_1 (and i_2, j_2) must exist since $d_G([i, j]) = 0$ for $i, j \in N$ with $j - i > n - k_0$ (note $d_G([i, j]) = d_G([j + 1, i - 1])$ and (2)). By considering Proposition 1 with $A = [i_1, i_2 - 1]$, $B = [i_2, j_2]$, and $C = [j_2 + 1, j_1]$ (see, Fig. 5), we obtain

$$\begin{aligned} &d_G([i_1, j_2]) + d_G([i_2, j_1]) \\ &= d_G([i_2, j_2]) + d_G([i_1, j_1]) + 2w_G([i_1, i_2 - 1], [j_2 + 1, j_1]). \end{aligned}$$

Thus

$$\begin{aligned} &w_G([i_1, i_2 - 1], [j_2 + 1, j_1]) \\ &= \frac{d_G([i_1, j_2]) + d_G([i_2, j_1]) - d_G([i_2, j_2]) - d_G([i_1, j_1])}{2} > 0, \end{aligned} \tag{5}$$

since $d_G([i_1, j_2]) = d_G([i_2, j_1]) = 0$ from (c), $d_G([i_2, j_2]) < 0$ from (b), and $d_G([i_1, j_1]) \leq 0$. (For an example for Fig. 4 (a), $w_G(10, 8) = 1 > 0$.) Hence there is a pair $i^* \in [i_1, i_2 - 1]$ and $j^* \in [j_2 + 1, j_1]$ such that

$$w_G(i^*, j^*) > 0. \tag{6}$$

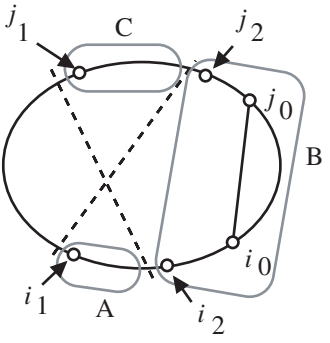


Fig. 5. Relation between nodes and cuts

(For an example for Fig. 4 (a), $i^* = i_1 = 10$ and $j^* = i_1 = 8$.) Since $(i^*, j^*) \in I$, it satisfies (b), i.e.,

$$d_G([i,j]) < 0 \text{ for all } i^* < i \leq i_0 \text{ and } j_0 \leq j < j^*. \tag{7}$$

From (4), (6), and (7) we can apply a cross-operation $X(i_0, j_0, j^*, i^*; \Delta)$ on G with

$$\Delta = \min\{w_G(i_0, j_0), w_G(i^*, j^*), \min_{i^* < i \leq i_0, j_0 \leq j < j^*} \{-d_G([i,j])/2\}\} > 0.$$

(For an example for Fig. 4 (a), $\Delta = 1$ and we obtain a graph of Fig. 4 (b) by the cross-operation.)

Now, we have found a cross-operation that makes G be closer to $G_\emptyset$. By applying the preceding discussion iteratively, we can find a sequence of cross-operations that makes G be closer to $G_\emptyset$. For completing the proof, we must show that the length of the sequence is finite. It is shown as follows.

Let G' be a graph obtained by applying $X(i_0, j_0, j^*, i^*; \Delta)$ to G. There are three cases: (I) $\Delta = w_G(i_0, j_0)$, (II) $\Delta = \min_{i^* < i \leq i_0, j_0 \leq j < j^*}\{-d_G([i,j])/2\}$, and (III) $\Delta = w_G(i^*, j^*)$. We consider each case as follows.

(I) $\Delta = w_G(i_0, j_0)$. In this case, $w_{G'}(i_0, j_0)$ becomes zero. Then by applying Proposition 1 with $A = \{i_0\}$, $B = [i_0 + 1, j_0 - 1]$, and $C = \{j_0\}$, we obtain $d_{G'}([i_0, j_0]) = 0$. Thus, the number of zero-linear-cuts of G' is greater than the one of G. Therefore (I) occurs at most $\binom{n}{2} < n^2$ times.

(II) $\Delta = \min_{i^* < i \leq i_0,\ j_0 \leq j < j^*}\{-d_G([i,j])/2\}$. Let i' and j' be nodes satisfying $i^* < i' \leq i_0$, $j_0 \leq j' < j^*$, and $\Delta = -d_G([i,j])/2$. Thus $d_{G'}([i', j'])$ becomes zero. Hence the number of zero-linear-cuts of G' is greater than the one of G. Therefore (II) occurs at most $\binom{n}{2} < n^2$ times.

(III) $\Delta = w_G(i^*, j^*)$. It is enough to consider a case of $\Delta < w_G(i_0, j_0)$, because if $\Delta = w_G(i_0, j_0)$, then case (I) could be applied. Then $w_{G'}(i_0, j_0) > 0$ and $w_{G'}(i^*, j^*) = 0$. In this case, we try again to find another pair of (i^*, j^*) for the same (i_0, j_0) (the same (i^*, j^*) be never found since $w_{G'}(i^*, j^*) = 0$). Thus (III) occurs successively at most $\binom{n}{2} < n^2$ times.

From (I)–(III), the length of the sequence of cross-operations is less than n^4. By using the sequence, G is transformed into $G_\emptyset$, i.e., $G \preceq_o G_\emptyset$. □

Proof of Theorem 2. Follows immediately from Lemmas 1, 2, and 3. □

Corollary 1. *Three relations $\preceq_c$, $\preceq_l$, and $\preceq_o$ are all partial orders.*

Proof. Clear from Theorem 2 and that $\preceq_c$ is a partial order. □

From Theorem 2, these three partial orders can be denoted by $\preceq$ simply. Moreover, we easily get the next.

Corollary 2. *Whether or not $G \preceq G'$ for a given pair of graphs G and G' can be determined in polynomial time.*

Proof. Clear from Theorem 2 and that the number of linear-cuts is $O(n^2)$. □

4 Concluding Remarks and Future Work

This paper extends the three orders, cut-size order, length order, and operation order, onto real capacitated (vertex labeled) graphs, and presents a proof for the equivalence of them.

Theorem 2 guarantees that there is a sequence of graphs $G = G_0$, G_1, …, $G_p = G'$ such that G_i $(i = 1, \ldots, p)$ can be obtained from G_{i-1} by applying a cross operation if $G \prec G'$. These graphs G_i $(i = 1, \ldots, p)$ may be not simple even if G and G' are both simple. Whether or not there is a sequence consists of simple graphs only in this case is an interesting problem. Some results have been obtained for this problem [6], but our conjecture that such sequence always exists if $d_G(i) = d_{G'}(i)$ for all $i \in N$ remains for future work.

References

1. Battista, G. D., Eades, P., Tamassia, R., Tollis, I. G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, NJ (1999)
2. Hakimi, S. L.: On Realizability of a Set of Integers as Degrees of the Vertices of a Linear Graph. I. J. Soc. Indust. Appl. Math., **10** (1962) 496–506
3. Hakimi, S. L.: On Realizability of a Set of Integers as Degrees of the Vertices of a Linear Graph II. Uniqueness. J. Soc. Indust. Appl. Math., **11** (1963) 135–147
4. Ito, H.: Relation among Edge Length of Convex Planar Drawings, Size of Linear Cuts, and Cross-Operations on Graphs. IPSJ SIG Notes, **2002**, 29 (2002) 27–34
5. Ito, H.: Sum of Edge Lengths of a Multigraph Drawn on a Convex Polygon. Computational Geometry, **24** (2003) 41–47
6. Ito, H.: On Transformation of Graphs with Preserving Their Simpleness. IPSJ SIG Notes, **2004**, 109 (2004) 1–8
7. Skiena, S. S.: Reconstructing Graphs from Cut-Set Sizes. Information Processing Letters, **32** (1989) 123–127
8. West, D. B.: Introduction to Graph Theory. Prentice Hall, NJ (1996)

Wedges in Euclidean Arrangements

Jonathan Lenchner

IBM T.J. Watson Research Institute,
Yorktown Heights, NY 10598
lenchner@us.ibm.com

Abstract. Given an arrangement of n not all coincident lines in the Euclidean plane we show that there can be no more than $\lfloor 4n/3 \rfloor$ wedges (i.e. two-edged faces) and give explicit examples to show that this bound is tight. We describe the connection this problem has to the problem of obtaining lower bounds on the number of ordinary points in arrangements of not all coincident, not all parallel lines, and show that there must be at least $\lceil (5n+6)/39 \rceil$ such points.

1 Introduction

As an extension of our investigation of Euclidean line arrangements where not all lines are coincident and not all lines are parallel [10], we have been led to consider bounds on the number of faces (cells) that contain just two (unbounded) edges. We call such an unbounded face a **wedge**.

Given a collection of points, a line which passes through precisely two of the points is called an **ordinary line**. Analogously, given an arrangement of lines, call a point which lies at the intersection of precisely two lines an **ordinary point**. The classical Theorem of Sylvester states that given a collection of not all collinear points there must exist an ordinary line. In [10] we prove the following sharp dual to Sylvester's Theorem:

Theorem 1. *Given any finite set of lines in the Euclidean plane, not all coincident and not all parallel, then there is a point where precisely two lines intersect.*

The proof relies on results of Kelly and Moser [9] and Csima and Sawyer [3] which give lower bounds on the number of ordinary lines in the dual collection of not all collinear points. In [10] we give analogous bounds on the number of ordinary points in line arrangements satisfying the hypotheses of Theorem 1.

In the following section we explain the connection between ordinary points and wedges and motivate our investigation. The next section develops the theory of wedges and culminates with a proof of the $\lceil 4n/3 \rceil$ bound on the maximum number of wedges. We then point out that a known algorithm for finding the envelope of an arrangement can also be used to find all wedges in time $O(n \log n)$. We wrap up by sharpening our lower bound on the number of ordinary points satisfying the conditions of Theorem 1, using the theory of wedges to remove caveats for small n.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 131–142, 2005.

2 Euler's Relation: Connection to Wedges

We begin by transferring our arrangement of lines in the Euclidean plane to an arrangement of circles on the sphere, all of which pass through the south pole. To do this, imagine the plane sitting on top of the sphere and project points from the plane to the sphere by stereographic projection through the south pole. Finally join up all circles by adding the south pole itself. If V denotes the number of vertices, E the number of edges, and F the number of faces in the induced arrangement on the sphere, then

$$V - E + F = 2. \tag{1}$$

Now, putting

$$t_j = number\ of\ vertices\ where\ j\ lines\ cross$$
$$p_k = number\ of\ faces\ surrounded\ by\ k\ edges$$

following the example of Melchior [11] as given in [6], write

$$Y = \sum_{j\geq 2}(3-j)t_j + \sum_{k\geq 2}(3-k)p_k. \tag{2}$$

With this notation we have

$$\sum_{j\geq 2} t_j = V, \qquad \sum_{k\geq 2} p_k = F. \tag{3}$$

Furthermore, since every edge is shared by two faces,

$$\sum_{k\geq 2} kp_k = 2E \tag{4}$$

and every edge is incident to two vertices,

$$\sum_{j\geq 2} jt_j = E. \tag{5}$$

Plugging relations (3), (4), (5) together with Euler's relation (1) into equation (2) gives

$$Y = 3(V - E + F) = 6. \tag{6}$$

Now, returning to the Euclidean plane, the point at the south pole has n lines crossing, so if we exclude this point from the vertex set, and denote by τ_j and ρ_k the analogs of t_j and p_k, then in the plane, we obtain

$$\sum_{j\geq 2}(3-j)\tau_j + \sum_{k\geq 2}(3-k)\rho_k = n + 3. \tag{7}$$

In equation (7) only the τ_2 and ρ_2 terms are positive and they both have coefficient 1. τ_2 is the number of vertices where two lines cross, and ρ_2 is the

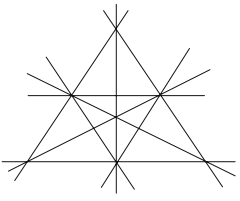


Fig. 1. Example with $n + 3$ wedges

number of wedges. Upper bounds on the number of wedges therefore immediately imply lower bounds on the number of ordinary points. In particular, if we knew that there could be no more than $n + 2$ wedges, relation (7) would immediately imply Theorem 1.

However, there are line arrangements with $n + 3$ wedges. Figure 1 gives an arrangement of 9 lines with 12 wedges.

This arrangement has a family of "essentially equivalent variations" with respect to the property of having $n + 3$ wedges as the examples of Figure 2 illustrate. The essential ingredients are two oppositely oriented similar triangles, with coinciding medians which each cut pairs of adjacent wedges.

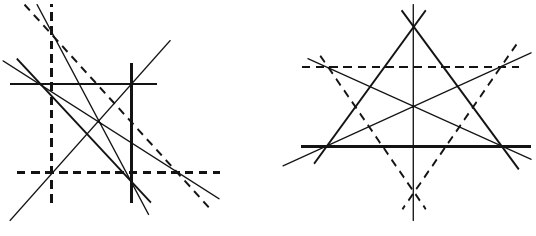


Fig. 2. Essentially equivalent variations in the case of $n = 9$ lines and 12 wedges

One might be tempted to conjecture that these are the only line arrangements with $n + 3$ wedges. However, Figure 3 gives an example with 16 lines and 12 wedges and another with 20 wedges and 15 lines. We describe a process for generating arrangements with $4n/3$ wedges for any $n \geq 6$, $n \equiv 0 \pmod 3$ in Theorem 3.

3 The Theory of Wedges

Lemma 1. *In any arrangement of $n \geq 3$ lines, not all of which are coincident and no two of which are parallel, there are at most n wedges.*

Proof. Order the lines cyclically by slope $l_1, ..., l_n$. One obtains n adjacency pairs (l_1, l_2), ...,(l_{n-1}, l_n), (l_n, l_1). A wedge must be formed between some such pair,

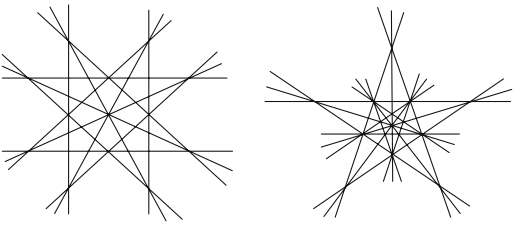


Fig. 3. $4n/3$ examples

on one unbounded end or the other (what we shall refer to as either the "top" or "bottom," where the choice of "top" is arbitrary). Since no two lines are parallel, each line contains at least two points of intersection. Hence, if (l_i, l_{i+1}) corresponds to a wedge on "top," it cannot correspond to a wedge on "bottom" (and vice versa). The lemma follows. □

It is very easy to achieve an arrangement of n lines, no two parallel, with n wedges, as the example in Figure 4 shows.

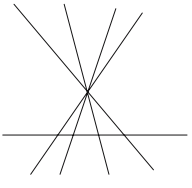


Fig. 4. n wedges with no two lines parallel

Another example is provided by extending the edges of a regular n-gon. If we label the lines which extend the edges in counter-clockwise order according to how they are attached in the n-gon, starting with an arbitrary l_1, as $l_1, ..., l_n$, then $(l_1, l_{\lceil n/2 \rceil})$ is a wedge. Continuing around cyclically we obtain n wedges. While there are other convex n-gons whose extended edges yield n wedges, many do not.

Lemma 2. *An arrangement of n lines with $n + k$ wedges must contain k pairs of parallel lines, no two pairs of which are parallel to one another.*

Proof. Adding a line to an arrangement can add at most two wedges: one at the "top" of the line and one at the "bottom." To see this, suppose that adding a line l_k could add more than one wedge at the "top." Far from all points of intersection, in the direction of "top," l_k lies between two lines, l_1 and l_2 say. For l_k to create more than one wedge at the "top" there must not have been a wedge between l_1 and l_2 before placing l_k, and there must be a pair of such faces after placing l_k. But for there to have been no wedge earlier means that there

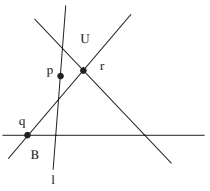


Fig. 5. 4 or more lines with just 3 wedges

had to be a line l_j intersecting both l_1 and l_2 in the direction of "top" after the point where l_1 and l_2 intersect (i.e closer to "top"). But then clearly the newly placed line l_k can create a wedge with l_1 or l_2, but not both.

Adding a third parallel line between two existing parallel lines clearly cannot contribute any wedges. The lemma follows by first placing a maximal subset of the n lines, no two of which are parallel, and then placing the remaining lines and recording the maximum number of wedges that can be thereby obtained. □

Figure 5 is an example of 4 lines with 3 wedges, and is thus an example of n lines with less than n wedges. It is also an example of a convex n-gon whose edges do not extend to form n wedges. In fact one can produce examples with 3 wedges and arbitrarily many lines. In Figure 5, we may either add many lines parallel to the line l, all placed between points p and r, or if we prefer an example with no two lines parallel, add lines through p which pass into the regions U and B, but do not pass through the point q.

Lemma 3. *Let $\mathcal{L}$ be a line arrangement and let $\mathcal{C}$ denote the extreme points of the convex hull of the intersection points of the members of $\mathcal{L}$. Then $\mathcal{L}$ must contain at least one wedge for each element of $\mathcal{C}$.*

Proof. $\mathcal{C}$ is formed by first taking intersection points of the members of $\mathcal{L}$, finding those points in the collection of intersection points which lie in the convex hull, and then removing non-extreme points. Hence the elements of $\mathcal{C}$ are all themselves intersection points. Any two edges e_i, e_j, which emanate from a point $p \in \mathcal{C}$ and extend from p to infinity, each form an edge of a wedge. There may be an edge between e_i and e_j but nonetheless we can associate at least one wedge with the point $p \in \mathcal{C}$. The same is true for any other point $q \in \mathcal{C}$, and clearly any edge extending from p to infinity is distinct from any edge extending from q to infinity, and hence their associated wedges are distinct. The lemma follows. □

Corollary 1. *In any arrangement of n lines, not all of which are parallel, there must be at least 3 wedges.*

Proof. If all the intersection points of the collection of lines $\mathcal{L}$ lie on one line then we have a family of parallel lines cutting a single line. In this case there are 4 wedges. Otherwise some three intersection points form a triangle and Lemma 3 applies. □

The following was first proved by Ching and Lee in [2].

Corollary 2. *In any arrangement of n lines it is possible to find the convex hull of the set of intersection points in time* $O(n \log n)$.

Proof. Without loss of generality, assume no line has infinite slope. Sort the lines by increasing slope, and in case of ties, by increasing y-intercept, labelling the sorted sequence $l_1, ..., l_n$. Follow this with a sequence $l'_1, ..., l'_n$ which is the same sequence, but this time sorted by increasing slope, and in case of ties, by *decreasing* y-intercept. Consider in turn the intersection of pairs

$$(l_1, l_2), ..., (l_{n-1}, l_n), (l_n, l'_1), (l'_1, l'_2), ..., (l'_{n-1}, l'_n), (l'_n, l_1).$$

Each wedge appears in this list. Lemma 3 implies that the set of intersection points of line pairs in this list includes the extreme points of the convex hull of intersection points of all pairs. Compute the convex hull of this set of up to $2n$ intersection points, not a priori knowing which are the extreme points, using any of the known $O(n \log n)$ algorithms. □

Once we have found the convex hull it is easy enough to generate a list of all intersection points on the hull, still in $O(n \log n)$ time. For each point found in the final convex hull above, keep track of the "first" and "last" lines intersecting at that point in the cyclical ordering $l_1, .., l_n, l'_1, ..., l'_n, l_1$. In other words if l_2, l_3, l_4 intersect at a point p, then l_2 is first and l_4 last. If l'_n, l_1, l_2 intersect at p, then l'_n is first and l_2 last. If $l, k \in \{l_1, .., l_n, l'_1, ..., l'_n\}$ are the first and last lines crossing at a point p_i and $\hat{l}, \hat{k} \in \{l_1, .., l_n, l'_1, ..., l'_n\}$ are the first and last lines crossing at the next point p_{i+1} of the convex hull, where points are ordered in clockwise order, then we must just gather the intersection points of each line between $k, \hat{l}$ (in the cyclical ordering), and the line segment connecting p_i, p_{i+1}. The collection of all such points, together with the original hull points, is a complete hull point enumeration.

It is also possible to enumerate all wedges in $O(n \log n)$ time, however, not simply by enumerating the hull points. There may be "inner" wedges - wedges that emanate from a point inside the convex hull of the intersection points. The left hand diagram in Figure 6 provides an example.

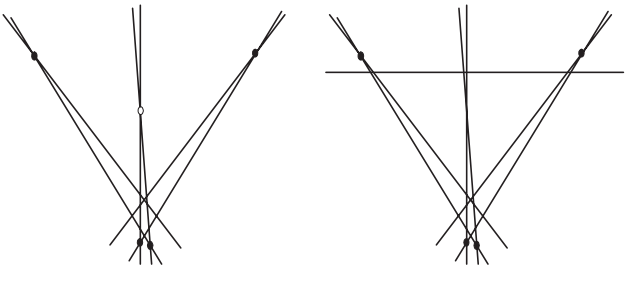


Fig. 6. Inner wedges and the difficulty detecting them

The convex hull points are marked with solid dots. The vertex of the inner wedge is marked with a hollow dot on the left. The difficulty is detecting the difference between the diagram on the left, which has an inner wedge, and the diagram on the right, which does not. Note that we can fit arbitrarily many inner wedges next to the wedge on the left, thus showing that there can be $\Omega(n)$ inner wedges. In fact it is not clear that one can tell whether an arbitrary face, given by (l_i, l_{i+1}) in the cyclical ordering, is a wedge in less than $O(n)$ time. Nonetheless, finding wedges is related to computing the *envelope* of an arrangement. Recall that the envelope of an arrangement is defined as the polygon whose boundary consists of the bounded edges of all the unbounded faces. Keil's algorithm [8] for computing the envelope of an arrangement actually pieces together the envelope by completely delineating all unbounded faces. As a result we have:

Theorem 2 (Keil). *In any arrangement of n lines in the plane, it is possible to completely describe all unbounded faces in time $\mathcal{O}(n \log n)$.*

Corollary 3. *In any arrangement of n lines in the plane, it is possible to find all wedges in time $O(n \log n)$.*

Our main result is the following:

Theorem 3. *In an arrangement of n lines, not all of which are coincident, there are at most $\lfloor 4n/3 \rfloor$ wedges. Furthermore, this bound is tight.*

Proof. We begin by establishing the $\lfloor 4n/3 \rfloor$ bound. Given n lines, in order for there to be $n + k$ wedges, by Lemma 2, requires k independent pairs of parallel lines. But between the k pairs of parallel lines there can be no wedge. In other words, of the $2n$ pairs

$$(l_1, l_2), \ldots, (l_{n-1}, l_n), (l_n, l'_1), \ldots, (l'_{n-1}, l'_n), (l'_n, l_1),$$

$2k$ of the pairs cannot be wedges. It follows that $n + k \leq 2n - 2k$, i.e. $k \leq n/3$. So the number of wedges is at most $\lfloor 4n/3 \rfloor$.

It remains to establish tightness of the bound. To this end we give explicit examples showing the bound being attained. We first give examples for the case when $n \geq 6$ is a multiple of 3. Start with a convex $2n/3$-gon with opposite sides parallel (for example a regular $2n/3$-gon) and extend the edges to infinity[1]. Label the extended edges (lines) in clockwise order according to how they are attached on the polygon, starting with an arbitrary edge: l_1,l_2,...,$l_{2n/3}$. Then the wedges are formed by $(l_1, l_{n/3})$, $(l_2, l_{n/3+1})$, etc. By virtue of the fact that we started with a convex polygon with opposite sides parallel, the line that bisects the wedge $(l_1, l_{n/3})$ also bisects the wedge formed by the pair of corresponding parallel lines, i.e. $(l_{n/3+1}, l_{2n/3})$. There is one bisector line for each such pair of wedges. Since there were $2n/3$ original lines with $2n/3$ wedges, adding $n/3$

[1] It is also possible to do this construction starting with two identically oriented regular $n/3$-gons with a common center point, of arbitrary relative sizes, with one $n/3$-gon rotated by $\pi/(n/3)$ degrees.

bisector lines adds $2n/3$ additional wedges, yielding a total of n lines with $4n/3$ wedges.

The case $n \geq 6$ where n is a multiple of 3 is thus established. Figure 7 establishes the cases $n = 3, 4$, and 5.

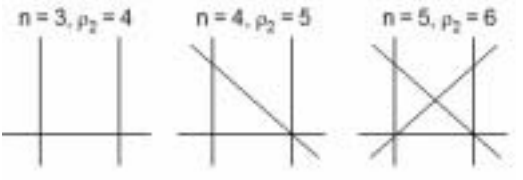


Fig. 7. $\lfloor 4n/3 \rfloor$ cases $n = 3, 4$ and 5

It remains to consider the cases of $n = m + 1$ and $n = m + 2$ where $m \geq 6$ with m a multiple of 3. Note that $\lfloor \frac{4(m+1)}{3} \rfloor = \frac{4m}{3} + 1$ and $\lfloor \frac{4(m+2)}{3} \rfloor = \frac{4m}{3} + 2$ so we need to show how to add a line at a time to an example like those of Figures 1 through 3 while also adding a single wedge. For this purpose, find a circle big enough so that it contains all intersection points. Traverse the circle clockwise beginning at its intersection with any pair of parallel lines. Label the lines as they are encountered l_1, l_2,...,l_n,l_2,l_1,l_3,l_5,l_4,l_6,...,l_n. We encounter a pair of parallel lines, followed by a splitting line, a pair of parallel lines, followed by a splitting line, and so forth, until we run through all lines. After reaching l_n we encounter the parallel lines in reverse order, but otherwise the overall order is unchanged. Now suppose we add a new line l_{n+1} that splits the wedge (l_2, l_3). Since this new line has no parallel, the new ordering will be l_1,l_2,l_{n+1},l_3...,l_n,l_2,l_1,l_{n+1},l_3,l_5,l_4,l_6,...,l_n. Previously (l_1, l_3) was a wedge. Now (l_1, l_{n+1}) is a wedge since l_{n+1} plainly intersects l_1 after l_3 does, which was the previous last point of intersection. Of course, (l_{n+1}, l_3) is not a wedge, since it is also bounded by l_1. In any case, the net result of adding the splitter l_{n+1} is that we have added a wedge. Similarly we can split the wedge (l_3, l_4) with a line l_{n+2} yielding the sequence l_1,l_2,l_{n+1},l_3,l_{n+2},l_4,...,l_n,l_2,l_1, l_{n+1},l_3,l_{n+2},l_5,l_4,l_6,...,l_n, and thereby adding another wedge. Figure 8 illustrates this construction, beginning with a regular hexagon.

We have thus established the tightness of the $\lfloor 4n/3 \rfloor$ bound and hence the theorem follows. □

Given a suitable random distribution on the space of lines, it is interesting to now consider the *expected* number of wedges in an arrangement. If we could ignore the possibility of there being inner wedges, the problem would reduce to that of determining the expected number of extreme points of the convex hull of the vertices of the arrangement, since (i) almost surely the vertex of a wedge does not happen to lie somewhere in the middle of a supporting line, and (ii) we can assume there is no more than one wedge at an extreme point, because almost surely, no three lines are coincident.

It is an interesting fact that the expected number of extreme points in the convex hull of the points of intersection of lines chosen randomly as duals from a uniform distribution of points in the unit square $[0,1]^2$ or unit disk $\{z : \|z\| \leq 1\}$ is actually bounded by a constant for all n, where n is the number of lines. See [7] and [5] respectively.

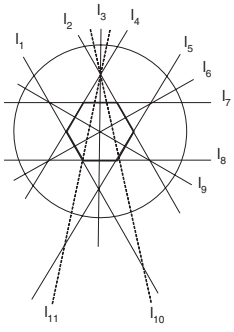


Fig. 8. Adding a wedge at a time in the cases $n \equiv 1, 2 \pmod 3$

As $n \to \infty$ it clearly becomes more and more unlikely that a randomly chosen internal unbounded face is a wedge, but this does not necessarily imply that the number of internal wedges goes to zero, or is even bounded.

4 Bound Results

In [10] we give a lower bound of $\lceil 5n/39 \rceil$ on the number of ordinary points in an arrangement of n not all coincident, not all parallel lines. A famous conjecture of Dirac and Motzkin states that if $n \neq 7, 13$ then in not all collinear collections of n points there must be $n/2$ ordinary lines. If this conjecture is true then the $\lceil 5n/39 \rceil$ bound can be improved to $\lceil n/6 \rceil$, except for the cases $n = 7, 13$. We can use the theory of wedges to get a precise statement in the $n = 7$ case.

Lemma 4. *An arrangement of $n = 7$ not all coincident, not all parallel lines must contain at least 2 ordinary points.*

Proof. Let $\mathcal{L}$ be an arrangement of lines satisfying the conditions of the lemma. If the number of wedges, $\rho_2 \leq 8$ then relation (7) implies that there must be at least 2 ordinary points. Hence assume $\rho_2 = 9$. By Lemma 2, $\mathcal{L}$ must contain at least two pairs of parallel lines. See Figure 9.

For there to be only one ordinary point, some two of the remaining three lines must intersect some two adjacent points $\{a, b, c, d\}$ in the given cyclical point ordering. Thus assume that the line l_5 intersects the point a and line l_6 intersects the point b. If l_5 does not pass through d and l_6 does not pass through c, then the 6 lines $\{l_1, ..., l_6\}$ form just 6 wedges. Laying the last line can add

at most 2 wedges, contradicting $\rho_2 = 9$. Hence assume that l_5 intersects both points a, d. If now l_6 does not pass through point c, then prior to the placement of l_7, l_6 intersects l_5, l_2, and l_3 at ordinary points. Laying just one line can only take care of one of these points - so at least two will remain ordinary.

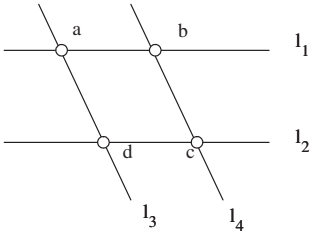


Fig. 9. Initial placement of the 2 mandatory sets of parallel lines

We are thus left with one final case, where l_6 passes through both points b and c. Now line l_7 must create one new wedge so that $\rho_2 = 9$. Without loss of generality, suppose l_7 passes through the point a. We are then in one of the situations of Figure 10 and so clearly have more than one ordinary point. The lemma follows. □

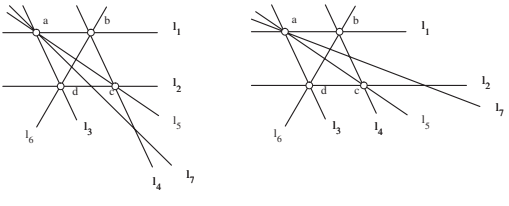


Fig. 10. Possible placements of l_7

A configuration of 7 lines meeting the hypotheses of Lemma 4 with 2 ordinary points is given in Figure 11. Additional best-possible bounds are easily obtained as in Table 1. The case of 8 lines comes from the $\lceil 5n/39 \rceil$ bound plus

Table 1. Ordinary point bounds for small n

# lines	3	4	5	6	7	8
Min # of ordinary points	2	2	2	1	2	2

the observation that if we remove the top horizontal line from Figure 1 we obtain an arrangement of 8 lines with 2 ordinary points.

With these results in hand it is possible to slightly refine the $\lceil 5n/39 \rceil$ result, while simplifying the basic counting argument from [10].

Theorem 4. *In an arrangement of n not all collinear, not all coincident lines in the Euclidean plane, there must be at least $\lceil (5n+6)/39 \rceil$ ordinary points.*

Proof. We consider the problem embedded in the real projective plane, where the Csima-Sawyer Theorem [3] says that there must be at least $\lceil 6n/13 \rceil$ ordinary points except when $n = 7$. The $n = 7$ case is handled by Lemma 4.

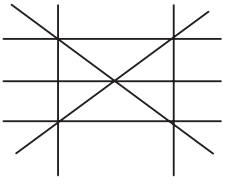


Fig. 11. An example with 7 lines and just 2 ordinary points

If our result were false then more than $\lceil \frac{6n}{13} - \frac{5n+6}{39} \rceil = \lceil \frac{n}{3} - \frac{2}{13} \rceil$ of these ordinary points would have to lie on the line at infinity. In other words there would have to be at least $\frac{n}{3} - \frac{2}{13}$ pairs of parallel lines. To this arrangement add the line at infinity. This "kills off" the at least $\lceil \frac{n}{3} - \frac{2}{13} \rceil$ ordinary points and creates at most $\lfloor n - \frac{2n}{3} + \frac{4}{13} \rfloor = \lfloor \frac{n}{3} + \frac{4}{13} \rfloor$ new ordinary points.

By Csima-Sawyer applied to the new arrangement (as long as $n \neq 6$, a case which is covered by Table 1) we have at least $\lceil \frac{6(n+1)}{13} \rceil$ ordinary points. But then there must have been at least $\lceil \frac{6(n+1)}{13} - \frac{n}{3} - \frac{4}{13} \rceil = \lceil \frac{5n+6}{39} \rceil$ finite ordinary points earlier, contradicting our initial assumption. The result is thus proved. □

5 Concluding Remarks

We leave a number of problems for future work. Given an arrangement of lines chosen randomly as duals from a uniform distribution of points in either the unit square or unit disk, does the expected number of internal wedges tends to zero as the number n of lines tends to infinity? Given these distributions we could then conclude that the expected number of wedges was bounded by a constant for all n.

Theorem 4 does not help resolve the exceptional $n = 13$ case, left over in case the $n/2$ conjecture is true. In the standard Sylvester problem, not all collinear point configurations with minimal possible numbers of ordinary lines are known for $n = 3, ..., 14, 16, 18, 22$ [1]. In the Sharp Dual case, computing best possible bounds on the number of ordinary points in arrangements of not all parallel, not all coincident lines for low n appears harder. It is not apparent how to push the methods of Lemma 4 much further.

In the Sharp Dual case it would be nice to get insights into best possible bounds for arbitrarily large n, and thus see if the $5n/39$ (respectively $n/6$, assuming the $n/2$ conjecture) bound is asymptotically tight for any $n \to \infty$. In

the standard Sylvester case it is known that if the $n/2$ conjecture is true, then $n/2$ would be a tight bound, at least for even n due to the examples of Böröczky (see [3]). I conjecture that my bounds are not asymptotically tight for any n and that the asymptotically best bound is likely the same as it is in the standard Sylvester problem.

Finally, it would be interesting to extend the analysis of wedges to 3 and higher dimensions. In dimension 3 the unbounded cells that take the place of wedges are the "three-faced cells." In exceptional cases there can also be two-faced cells. Such an analysis could inform the "ordinary plane problem" akin to Sylvester's ordinary line problem. See for example the paper of Devillers and Mukhopadhyay [4].

Acknowledgements

I thank Hervé Brönnimann, H. Richard Gail and Bill Steiger for their discussions and the anonymous referees for their valuable commentary.

References

1. P. Borwein and W. Moser. A survey of Sylvester's problem and its generalizations. *Aequationes Mathematicae*, 40: 111-135, 1990.
2. Y. T. Ching and D. T. Lee. Finding the diameter of a set of lines. *Pattern Recognition*, 18: 249-255, 1985.
3. J. Csima and E. Sawyer. There exist $6n/13$ ordinary points. *Discrete and Computational Geometry*, 9: 187-202, 1993.
4. O. Devillers and A. Mukhopadhyay. Finding an ordinary conic and an ordinary hyperplane. *Nordic Journal of Computing*, 6: 462-468, 1999.
5. L. Devroye and G. Toussaint. Convex hulls for random lines. *Journal of Algorithms*, 14: 381-394, 1993.
6. S. Felsner. *Geometric Graphs and Arrangements*. Vieweg and Sohn-Verlag, Wiesbaden, Germany, 2004.
7. M. Golin, S. Langerman and W. Steiger. The convex hull for random lines in the plane. *Proceedings of Japan Conference on Computational Geometry*, 2002.
8. M. Keil. A simple algorithm for determining the envelope of a set of lines. *Information Processing Letters*, 39: 121-124, 1991.
9. L. Kelly and W. Moser. On the number of ordinary lines determined by n points. *Canadian Journal of Mathematics*, 10: 210-219, 1958.
10. J. Lenchner. On the dual and sharpened dual of Sylvester's theorem in the plane. *IBM Research Report*, RC23411 (W0409-066), 2004.
11. E. Melchior. Über vielseite der projektiven eberne. *Deutsche Math.*, 5: 461-475, 1940.

Visual Pascal Configuration and Quartic Surface

Yoichi Maeda

Tokai University, Hiratsuka, Kanagawa, 259-1292, Japan
maeda@keyaki.cc.u-tokai.ac.jp

Abstract. For any two lines and a point in $\mathbf{R}^3$, there is a line having intersections with these two lines and the point. This fact implies that two lines in $\mathbf{R}^3$ make a *visual intersection* from any viewpoint even if these lines are in twisted position. In this context, the well-known Pappus' theorem in $\mathbf{R}^2$ is simply extended as that in $\mathbf{R}^3$, i.e., if the vertices of a spatial hexagon lie alternately on two lines, then from any viewpoint, three visual intersections of opposite sides are *visual collinear*. In a similar way, Pascal's theorem is also extended in $\mathbf{R}^3$, i.e., if the vertices of a spatial hexagon lie on a cone, three visual intersections of opposite sides are *visual collinear* from the viewpoint at the vertex of the cone. In this case, for six vertices in $\mathbf{R}^3$ we obtain a quartic surface as the set of viewpoints. We will investigate this surface depending on the vertices of a spatial hexagon. A relation between non-singular cubic curve and complete quadrilateral is naturally and geometrically derived.

1 Introduction

In general, two lines in $\mathbf{R}^3$ look like having an intersection from any viewpoint even if these lines are in twisted position. This *visual intersection* is precisely defined as a ray from the viewpoint passing through these two lines. In this context, the well-known Pappus' theorem in $\mathbf{R}^2$ is simply extended as that in $\mathbf{R}^3$, i.e., if the vertices of a spatial hexagon lie alternately on two lines, then from any viewpoint, three visual intersections of opposite sides are *visual collinear*, the state that rays from the viewpoint lie on a plane. In a similar way, Pascal's theorem as the generalization of Pappus' theorem is also extended in $\mathbf{R}^3$, however, the viewpoints, which satisfy visual collinear are generally restricted in a quartic surface. We will investigate this surface depending on the vertices of a spatial hexagon.

In this paper, let us focus on the cases that the equation of the quartic surface is reducible. We will classify these reducible quartic surfaces into three types; Desargues type(1,1,1,1), Quadratic type(2,1,1), and Cubic type(3,1) in Section 2. In the study of these types, a relation between non-singular cubic curve and complete quadrilateral is naturally and geometrically derived. In Section 2, the equation of the quartic surface is introduced. We start to study the simplest Desargues type(1,1,1,1) in Section 3. Moreover, Quadratic(2,1,1) and Cubic(3,1) types are investigated in Section 4 and 5, respectively.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 143–150, 2005.

2 Pascal Configuration and Quartic Surface

Let $\{P_i\}_{i=1,2,\ldots,6}$ be distinct six points in $\mathbf{R}^3$ and $\mathcal{S}$ be the set of viewpoints from where these six points are in *visual Pascal configuration* which means that six points look like on a conic from the viewpoint. Actually, $\mathcal{S}$ is the set of vertex $V = (x, y, z)$ of a cone(possibly degenerated into two planes) on which all six points lie. This set $\mathcal{S}$ is generally a quartic surface represented in the following theorem.

Theorem 1. *$\mathcal{S}$ is the set which satisfies the quartic equation:*

$$D_{314}D_{425}D_{516}D_{623} = D_{324}D_{415}D_{526}D_{613} \tag{1}$$

where D_{ijk} is the determinant:

$$D_{ijk} := \begin{vmatrix} x & y & z & 1 \\ x_i & y_i & z_i & 1 \\ x_j & y_j & z_j & 1 \\ x_k & y_k & z_k & 1 \end{vmatrix}. \tag{2}$$

Proof. The idea is the projection on a plane and the cross ratio of conic section. Six points $\{P_i = (x_i, y_i, z_i)\}_{i=1,2,\ldots,6}$ are on a cone centered at $V = (x, y, z)$. Without loss of generality(or adequate coordinate transformation), assume that $z_i \neq z$. Regard V as the origin, and consider the central projection to the $z = 1$ plane with respect to the origin. On the plane, the projected six points are given as $\{P_i' = ((x_i - x)/(z_i - z), (y_i - y)/(z_i - z), 1)\}_{i=1,2,\ldots,6}$. These coplanar points are on a conic, if and only if, the following cross ratios are equal [2](pp.131):

$$[\overrightarrow{P_1'P_3'}, \overrightarrow{P_1'P_4'}, \overrightarrow{P_1'P_5'}, \overrightarrow{P_1'P_6'}] = [\overrightarrow{P_2'P_3'}, \overrightarrow{P_2'P_4'}, \overrightarrow{P_2'P_5'}, \overrightarrow{P_2'P_6'}]. \tag{3}$$

Equation (3) is equivalent to

$$\begin{aligned} &\sin\angle P_3'P_1'P_4' \cdot \sin\angle P_5'P_1'P_6' : \sin\angle P_4'P_1'P_5' \cdot \sin\angle P_6'P_1'P_3' \\ &= \sin\angle P_3'P_2'P_4' \cdot \sin\angle P_5'P_2'P_6' : \sin\angle P_4'P_2'P_5' \cdot \sin\angle P_6'P_2'P_3', \end{aligned} \tag{4}$$

where these angles are oriented. Moreover, using cross product($|\overrightarrow{OA} \times \overrightarrow{OB}| = |\overrightarrow{OA}||\overrightarrow{OB}| \sin\angle AOB$), Equation (4) is equivalent to

$$\begin{aligned} &(\overrightarrow{P_1'P_3'} \times \overrightarrow{P_1'P_4'})_z(\overrightarrow{P_1'P_5'} \times \overrightarrow{P_1'P_6'})_z : (\overrightarrow{P_1'P_4'} \times \overrightarrow{P_1'P_5'})_z(\overrightarrow{P_1'P_6'} \times \overrightarrow{P_1'P_3'})_z \\ &= (\overrightarrow{P_2'P_3'} \times \overrightarrow{P_2'P_4'})_z(\overrightarrow{P_2'P_5'} \times \overrightarrow{P_2'P_6'})_z : (\overrightarrow{P_2'P_4'} \times \overrightarrow{P_2'P_5'})_z(\overrightarrow{P_2'P_6'} \times \overrightarrow{P_2'P_3'})_z. \end{aligned}$$

Finally, the following direct calculation completes the proof:

$$(\overrightarrow{P_j'P_i'} \times \overrightarrow{P_j'P_k'})_z = \begin{vmatrix} \frac{x_i - x}{z_i - z} - \frac{x_j - x}{z_j - z} & \frac{y_i - y}{z_i - z} - \frac{y_j - y}{z_j - z} \\ \frac{x_k - x}{z_k - z} - \frac{x_j - x}{z_j - z} & \frac{y_k - y}{z_k - z} - \frac{y_j - y}{z_j - z} \end{vmatrix} = \frac{D_{ijk}}{(z_i - z)(z_j - z)(z_k - z)}.$$

□

Remark 1. If $\mathcal{S}$ is regarded as a set in projective space $\mathbf{P}_3$, the determinant D_{ijk} in (2) is defined as follows:

$$D_{ijk} := \begin{vmatrix} x & y & z & w \\ x_i & y_i & z_i & 1 \\ x_j & y_j & z_j & 1 \\ x_k & y_k & z_k & 1 \end{vmatrix}. \tag{5}$$

Remark 2. By the permutation $\begin{pmatrix} 2 & 3 & 4 \\ 4 & 2 & 3 \end{pmatrix}$, Equation (1) is equivalent to

$$D_{123}D_{345}D_{561}D_{246} = D_{612}D_{234}D_{456}D_{135}, \tag{6}$$

which we use in the succeeding sections.

Note that $D_{ijk} = 0$ is the equation of the plane passing through three points P_i, P_j, and P_k. Hence, the intersection line of two planes defined by three points and the others(for example $[D_{123} = 0] \cap [D_{456} = 0]$) is included in $\mathcal{S}$ because $D_{123} = D_{456} = 0$ in Equation (6). In addition, the lines $\overline{P_iP_j}(i \neq j)$ are included in $\mathcal{S}$ because, for example, if $P(x, y, z) \in \overline{P_1P_2}$, then $D_{123} = D_{612} = 0$ in Equation (6). In consequence, $\mathcal{S}$ includes $25(={}_6C_3/2+{}_6C_2)$ lines in general. Geometrically, these facts are trivial, i.e., from a viewpoint in $[D_{123} = 0] \cap [D_{456} = 0]$, P_1, P_2 and P_3 are *visual collinear*, and also P_4, P_5 and P_6, therefore, these points are in *visual Pappus configuration*. On the other hand, from a viewpoint in $\overline{P_1P_2}$, P_1 and P_2 look like one point, therefore, these points are generally in *visual Pascal configuration* (a conic section is determined by five points in $\mathbf{R}^2$).

Now it is also trivial that $\mathcal{S} = \mathbf{R}^3$ if six points are on two lines (P_1, P_2 and P_3 are collinear, and also P_4, P_5 and P_6, then $D_{123} = D_{456} = 0$ in Equation (6)). In the following argument, let us assume that any five points are not coplanar, and any three points are not collinear. Under these conditions, the possibility of reducible surface is the following three types in degree:

$$\begin{cases} \text{Desargues type:} & (1,1,1,1) \\ \text{Quadratic type:} & (2,1,1) \\ \text{Cubic type:} & (3,1), \end{cases}$$

where Desargues type is named after the figure which is used in the proof of Desargues' theorem(see, Fig. 1(left)). This classification is determined by the number of coplanar 4-tuples; three, two, and one, respectively.

3 Desargues Type

In this section, assume that there exist three planes on which four of the six points lie. Then there are concurrent three edges as the intersection of two planes such that exact two points are on each edge(see, Fig. 1(left)). Let P_0 be the intersection point(possibly at infinity) of three edges. The problem is the relation between the fourth plane and the given six points.

Theorem 2. *Let four points* $\{P_1, P_2, P_3, P_4\}$ *be on the same plane, and also* $\{P_3, P_4, P_5, P_6\}$, $\{P_5, P_6, P_1, P_2\}$ *be coplanar. Then the surface* $\mathcal{S}$ *is composed of four planes: three planes* $P_1P_2P_3P_4$, $P_3P_4P_5P_6$, $P_5P_6P_1P_2$, *and the fourth plane which is the polar plane of* P_0 *with respect to any quadratic surface passing through the six points* P_i.

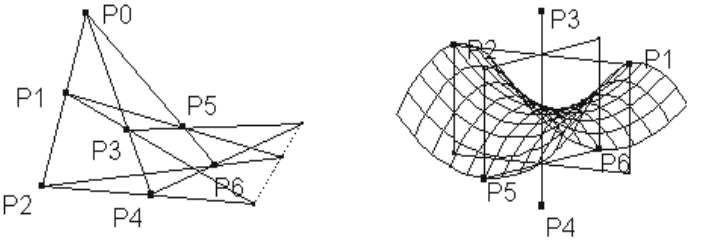


Fig. 1. Desargues and Quadratic types

Proof. Under an appropriate affine transformation, the coordinates of the six points are given by:

$$P_1(x_1, 0, 0),\ P_2(x_2, 0, 0),\ P_3(0, y_1, 0),\ P_4(0, y_2, 0),\ P_5(0, 0, z_1),\ P_6(0, 0, z_2),$$

where $x_1x_2y_1y_2z_1z_2 \neq 0$ and $x_1 \neq x_2$, $y_1 \neq y_2$, $z_1 \neq z_2$. Then, P_0 is the origin $(0, 0, 0)$, and using Equation (6), the set $\mathcal{S}$ is determined by three planes $xyz = 0$ and the fourth plane given by:

$$\frac{x_1 + x_2}{x_1x_2}x + \frac{y_1 + y_2}{y_1y_2}y + \frac{z_1 + z_2}{z_1z_2}z - 2 = 0.$$

This plane passes through $R_1(\frac{2x_1x_2}{x_1+x_2}, 0, 0)$, $R_2(0, \frac{2y_1y_2}{y_1+y_2}, 0)$, and $R_3(0, 0, \frac{2z_1z_2}{z_1+z_2})$. Let $\mathcal{Q}$ be a quadratic surface passing through the six points P_i. To complete the proof, it is enough to show the three cross ratios $[P_0, R_1, P_1, P_2]$, $[P_0, R_2, P_3, P_4]$, and $[P_0, R_3, P_5, P_6]$ are equal to -1 (harmonic division).

$$[P_0, R_1, P_1, P_2] = \frac{0 - x_1}{x_1 - \frac{2x_1x_2}{x_1+x_2}} \frac{\frac{2x_1x_2}{x_1+x_2} - x_2}{x_2 - 0} = -1,$$

even if $x_1 + x_2 = 0$. In the same way, the other cross ratios are also equal to -1. □

Example 1. If the six points are the vertices of a regular octahedron, then P_0 is the center of the octahedron and the fourth plane is the infinite plane. The other simple example is the vertices of a prism. In this case, P_0 is at infinity, and the fourth plane is the plane which is parallel to the upper and lower base cutting the prism in half.

4 Quadratic Type

In the next, let us consider the case that there are two planes on which four of the six points lie(see, Fig. 1(right)).

Theorem 3. *Assume that both* $\{P_1, P_2, P_3, P_4\}$ *and* $\{P_3, P_4, P_5, P_6\}$ *are coplanar, however,* $\{P_1, P_2, P_5, P_6\}$ *are not coplanar. Then the set* $\mathcal{S}$ *is composed of the two planes and a doubly ruled surface* $\mathcal{Q}$*(hyperbolic paraboloid or hyperboloid of one sheet). In addition, the two points* Q_1, Q_2 *in* $\mathcal{Q} \cap \overline{P_3P_4}$ *are harmonic with respect to* P_3 *and* P_4*, i.e., the cross ratio* $[P_3, P_4, Q_1, Q_2]$ *is equal to* -1*.*

Proof. Since $\{P_1, P_2, P_5, P_6\}$ are not coplanar, the four lines $\overline{P_1P_5}$, $\overline{P_5P_2}$, $\overline{P_2P_6}$ and $\overline{P_6P_1}$ are on the quadratic surface $\mathcal{Q}$, that is to say, $\mathcal{Q}$ is doubly ruled. Under an appropriate affine transformation, the coordinates of the six points are given by:

$$P_1(0,1,1),\ P_2(0,-1,1),\ P_3(x_3,y_3,z_3),\ P_4(x_4,y_4,z_4),\ P_5(1,0,-1),\ P_6(-1,0,-1),$$

where $z_3 - 1 : x_3 = z_4 - 1 : x_4$ and $z_3 + 1 : y_3 = z_4 + 1 : y_4$. Then, using Equation (6) the two planes are $(z_3 - 1)x - (z - 1)x_3 = 0$ and $(z_3 + 1)y - (z + 1)y_3 = 0$, and the quadratic surface $\mathcal{Q}$ is written as:

$$\begin{aligned}&(4y_3y_4 - (z_3+1)(z_4+1))\,(2x - z + 1)(2x + z - 1) - \\ &(4x_3x_4 - (z_3-1)(z_4-1))\,(2y - z - 1)(2y + z + 1) = 0,\end{aligned}$$

or, equivalently,

$$2(2x_3x_4 + 2y_3y_4 - z_3z_4 - 1)p(x,y,z) + (2x_3x_4 - 2y_3y_4 + z_3 + z_4)h(x,y,z) = 0, \quad (7)$$

where $p(x,y,z) = x^2 - y^2 + z$ and $h(x,y,z) = -2x^2 - 2y^2 + z^2 + 1$.

Let $(\lambda x_3 + \mu x_4, \lambda y_3 + \mu y_4, \lambda z_3 + \mu z_4)$ be an intersection with $\mathcal{Q}$ and the line $\overline{P_3P_4}$ where $\lambda + \mu = 1$. Applying this to Equation (7)(exchange the constant term of $h(x,y,z)$ for $(\lambda+\mu)^2$),

$$\begin{aligned}&\lambda^2\,(2(2x_3x_4 + 2y_3y_4 - z_3z_4 - 1)p(x_3,y_3,z_3) + (2x_3x_4 - 2y_3y_4 + z_3 + z_4)h(x_3,y_3,z_3)) + \\ &\mu^2\,(2(2x_3x_4 + 2y_3y_4 - z_3z_4 - 1)p(x_4,y_4,z_4) + (2x_3x_4 - 2y_3y_4 + z_3 + z_4)h(x_4,y_4,z_4)) = 0.\end{aligned} \quad (8)$$

If it is proved that neither P_3 nor P_4 lie on $\mathcal{Q}$, then there are two solutions (possibly complex) in the equation above, say, (λ_1, μ_1) and (λ_2, μ_2), and note that $\lambda_1\mu_2 + \lambda_2\mu_1 = 0$. Then, the cross ratio is(use $\lambda_1 + \mu_1 = \lambda_2 + \mu_2 = 1$)

$$[P_3, P_4, Q_1, Q_2] = [P_3, P_4, \lambda_1 P_3 + \mu_1 P_4, \lambda_2 P_3 + \mu_2 P_4] = \frac{\mu_1\lambda_2}{\lambda_1\mu_2} = -1.$$

The rest is to prove the next lemma. □

Lemma 1. *Neither P_3 nor P_4 lie on $\mathcal{Q}$.*

Proof. Let α be the plane on which P_1, P_2, P_3 and P_4 lie, and P_0 be the intersection point $\alpha \cap \overline{P_5P_6}$(possibly at infinity)(Fig. 2(center)). Then three points P_0, P_3 and P_4 are collinear on α. In addition, the line $\overline{P_0P_1}$ is tangent to $\mathcal{Q}$ at P_1, because both $\overline{P_1P_5}$ and $\overline{P_1P_6}$ lie on $\mathcal{Q}$, i.e., the plane $P_1P_5P_6$ is tangent to $\mathcal{Q}$ at P_1. In the same way, the line $\overline{P_0P_2}$ is tangent to $\mathcal{Q}$ at P_2. If $P_3 \in \mathcal{Q}$ and $P_4 \notin \mathcal{Q}$, then the solution of Equation (8) is $(\lambda, \mu) = (1, 0)$, hence $\overline{P_3P_4}$ is tangent to $\mathcal{Q}$ at P_3. This means that $P_3 = P_1$ or P_2 which is contradiction. In the same reason, the case that $P_3 \notin \mathcal{Q}$ and $P_4 \in \mathcal{Q}$ is also impossible. Finally, suppose that $P_3, P_4 \in \mathcal{Q}$. Then Equation (8) is satisfied for any (λ, μ), therefore, the line $\overline{P_3P_4}$ lies on $\mathcal{Q}$. Since any three points of P_i are not collinear, $\alpha \cap \mathcal{Q} = \overline{P_1P_2} \cup \overline{P_3P_4}$, in particular, $\overline{P_1P_2}$ lies on $\mathcal{Q}$. For $\mathcal{Q}$ is doubly ruled, it is impossible that three lines $\overline{P_1P_2}, \overline{P_1P_5}$ and $\overline{P_1P_6}$ lie on $\mathcal{Q}$. □

Example 2. In the case that $x_3 = y_3 = x_4 = y_4 = 0$, $\mathcal{Q}$ is a hyperbolic paraboloid if and only if $z_3 + z_4 = 0$(see, Fig. 1). On the other hand, $\mathcal{Q}$ is a hyperboloid of one sheet if and only if $z_3z_4 = -1$.

5 Cubic Type

Finally, let us investigate the cubic type$(3, 1)$. Already seen in the previous sections, *harmonic division*(cross ratio is -1) plays an important role. This harmonic division has a close relation with *complete quadrilateral*. We can also see this relation in the cubic type. In this section, assume that P_1, P_2, P_3, P_4 are coplanar and lie on the plane α, and let P_0 be the intersection point $\alpha \cap \overline{P_5P_6}$. Let us focus on the intersection curve C with the plane α and the other cubic surface. In Fig. 2, thick curves represent the intersection C; three lines, one line and a conic, and a cubic curve, respectively. It seems that three diagonal intersections D_1, D_2, D_3 of quadrilateral $P_1P_2P_3P_4$ are on C. In Desargues type, this is trivial, because D_1 is P_0, and D_2 and D_3 are on the polar of P_0 with respect to any quadratic curve passing through four points $P_i (i = 1, 2, 3, 4)$. As

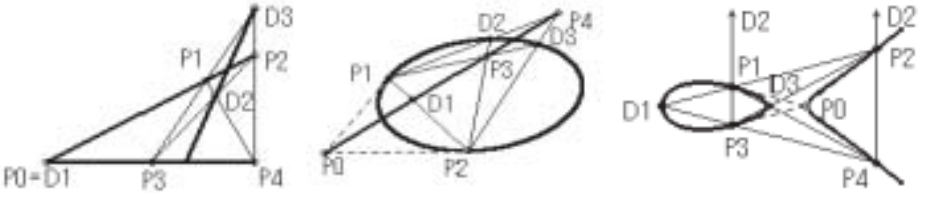


Fig. 2. Intersection of the plane α and the cubic surface: Desargues(left), Quadratic(center), Cubic(right) type

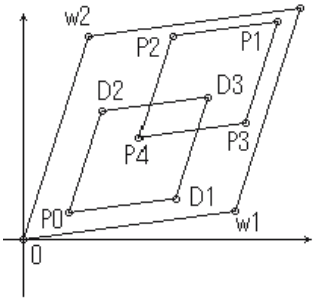


Fig. 3. Period Parallelogram

for Quadratic type, it is proved by Pascal's theorem. In fact, let D_2, D_3 be intersections of the conic section and $\overline{P_2P_3}, \overline{P_1P_3}$ respectively as in Fig. 2(center). Let us regard $P_1, P_1, D_3, P_2, P_2, D_2$ as vertices of a hexagon and P_0P_3 as the Pascal line [1](pp.176,213). Then P_0, P_3 and $R_1 \equiv \overline{P_1D_2} \cap \overline{P_2D_3}$ are collinear, and the fact $[P_3, P_4, Q_1, Q_2] = -1$ implies that $R_1 = P4$.

On the other hand, it also seems that the line $\overline{P_0P_i}$ is tangent to C at $P_i(i = 1, 2, 3, 4)$, in each case of Fig. 2. This fact is trivial in Desargues type. As for Quadratic type, it is proved in the proof of Lemma 1. In this way, the following theorem is naturally and geometrically introduced. The statement is more generalized in complex.

Theorem 4. *For any non-singular cubic curve C and a point P_0 on C, there are four points $P_i(i = 1, 2, 3, 4)$ such that the line $\overline{P_0P_i}$ are tangent to C at $P_i(i = 1, 2, 3, 4)$. In addition, three diagonal intersections $D_i(i = 1, 2, 3)$ of the quadrilateral $P_1P_2P_3P_4$ lie on C.*

Proof. The key point of the proof is the group law on a cubic curve. For any non-singular cubic curve C, there exist complex numbers ω_1, ω_2(called *periods*) in the complex u plane, and the mapping $u \to P(u)$ which is a homomorphism from the additive group of complex numbers onto the group of complex points on C [3](pp.43-45). Note the kernel of this homomorphism is the lattice $L = \mathbf{Z}\omega_1 + \mathbf{Z}\omega_2 = \{n_1\omega_1 + n_2\omega_2 : n_1, n_2 \in \mathbf{Z}\}$, hence one can identify any point on C with a complex number which lies in the *period parallelogram* (the parallelogram whose sides are the period ω_1 and ω_2(Fig. 3)). Now let P_0 correspond to p_0, i.e., $P(p_0) = P_0$. Since the line $\overline{P_0P_i}$ is tangent to C at $P_i(i = 1, 2, 3, 4)$, $2P_i + P_0 = \mathcal{O}$ in the addition law on C where $\mathcal{O}$ is the zero element for the group law. Therefore, without loss of generality, we can set $P(-\frac{p_0}{2}) = P_1, P(-\frac{p_0}{2} + \frac{\omega_1}{2}) = P_2, P(-\frac{p_0}{2} + \frac{\omega_2}{2}) = P_3$, and $P(-\frac{p_0}{2} + \frac{\omega_1+\omega_2}{2}) = P_4$. Let $u_1 = p_0 + \frac{\omega_1}{2}$, then, $\left(p_0 + \frac{\omega_1}{2}\right) + \left(-\frac{p_0}{2}\right) + \left(-\frac{p_0}{2} + \frac{\omega_1}{2}\right) \equiv 0$ and $\left(p_0 + \frac{\omega_1}{2}\right) + \left(-\frac{p_0}{2} + \frac{\omega_2}{2}\right) + \left(-\frac{p_0}{2} + \frac{\omega_1}{2} + \frac{\omega_2}{2}\right) \equiv 0$, which means that the point $P(u_1) \in C$ satisfies $P(u_1) + P_1 + P_2 = \mathcal{O}$ and $P(u_1) + P_3 + P_4 = \mathcal{O}$, i.e., both $\{P_1, P_2, P(u_1)\}$ and $\{P_3, P_4, P(u_1)\}$ are collinear. Consequently, $\overline{P_1P_2} \cap \overline{P_3P_4} \in C$. In the same way, two other diagonal intersections also lie on C. □

References

1. Berger, M.: Geometry II Springer-Verlag, Berlin Heidelberg. (1987)
2. Jennings, G.: Modern Geometry with Applications. Springer-Verlag, New York. (1994)
3. Silverman, J. H., Tate, J.: Rational Points on Elliptic Curves. Springer-Verlag, New York. (1992)

Nonexistence of 2-Reptile Simplices

Jiří Matoušek

Department of Applied Mathematics,
and Institute of Theoretical Computer Science (ITI),
Charles University, Malostranské nám. 25, 118 00 Praha 1 Czech Republic
matousek@kam.mff.cuni.cz

Abstract. A simplex S is called an *m-reptile* if it can be tiled without overlaps by simplices $S_1, S_2, \ldots, S_m$ that are all congruent and similar to S. The only m-reptile d-simplices that seem to be known for $d \geq 3$ have $m = k^d$, $k \geq 2$. We prove, using eigenvalues, that there are no 2-reptile simplices of dimensions $d \geq 3$. This investigation has been motivated by a probabilistic packet marking problem in theoretical computer science, introduced by Adler in 2002.

1 Introduction

A closed set $A \subset \mathbf{R}^d$ with nonempty interior is called an *m-reptile* (sometimes written "m rep tile" or "m rep-tile") if there are sets $A_1, A_2, \ldots, A_m$ with disjoint interiors and with $A = A_1 \cup A_2 \cup \cdots \cup A_m$ that are all congruent and similar to A. Such sets have been studied in connection with fractals and also with crystallography and tilings of $\mathbf{R}^d$; see, for example, [4], [14], [9], [8], [12]. The connection with tilings stems mainly from the simple observation that any m-reptile, $m \geq 2$, tiles $\mathbf{R}^d$.

Here we consider the following question: For what m and d there exist d-dimensional *simplices* that are m-reptiles? This investigation was motivated by a paper of Adler [1] on probabilistic marking of Internet packets. We will briefly discuss this connection and the quite interesting questions arising there in Section 3 (see [2] for more details). From this point of view, it would be interesting to find d-dimensional m-reptile simplices with m as small as possible.

In the plane, there are several types of m-reptile triangles. The following list exhausts all possible cases [14] (see Fig. 1):

(a) Any triangle is a k^2-reptile for all $k = 2, 3, \ldots$.
(b) A right triangle with acute angles $\frac{\pi}{6}$ and $\frac{\pi}{3}$ is a $3k^2$-reptile for all $k = 1, 2, \ldots$.
(c) A right triangle whose two shorter sides have ratio $k : \ell$ is a $(k^2 + \ell^2)$-reptile, $k, \ell = 1, 2, \ldots$.

Higher-dimensional simplex reptiles seem to be much more rare. At a first encounter with the problem it is tempting to think that the dissection of an equilateral triangle, say, into four equilateral triangles (a special case of (a) above) can be generalized to a dissection of a regular tetrahedron, but once one cuts off the corners of the tetrahedron, one obtains a regular octahedron, which cannot

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 151–160, 2005.

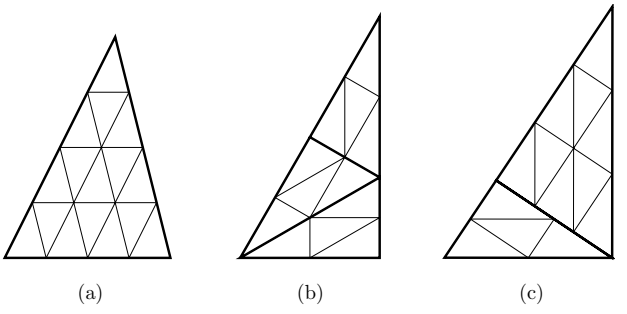


Fig. 1. Planar reptile triangles

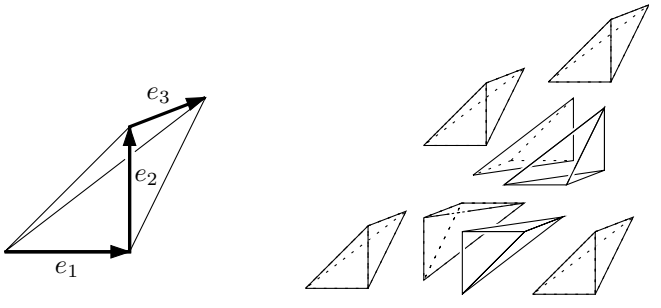


Fig. 2. A 3-dimensional Hill simplex as an 8-reptile

be tiled by regular simplices. The only known construction, at least as far as I could find, of higher-dimensional simplex m-reptiles has $m = k^d$ and is known as the *Hill simplex* (or *Hadwiger-Hill simplex*). A d-dimensional Hill simplex is the convex hull of vectors 0, b_1, $b_1 + b_2$,...,$b_1 + \cdots + b_d$, where $b_1, b_2, \ldots, b_d$ are vectors of equal length such that the angle between any two of them is the same and lies in the interval $(0, \frac{2\pi}{3})$. Fig. 2 shows the decomposition of a 3-dimensional Hill simplex, with $(b_1, b_2, b_3) = (e_1, e_2, e_3)$ the standard orthonormal basis, into 8 congruent pieces similar to it.

In the absence of constructions of m-reptile simplices with $m < 2^d$ one can try to prove nonexistence results, but the general problem seems quite challenging. The only known result on this question I could find is by Hertel [10], who proves that a 3-dimensional simplex is a k^3-reptile using a "standard" way of dissection if and only if it is a Hill simplex. He conjectures that Hill simplices are the only 3-dimensional reptile simplices. Simplex dissection was considered in several other papers (e.g., [6], [15], [3]) but from different points of view.

In this note we establish the following partial result:

Theorem 1. *For $d \geq 3$ no d-dimensional simplex is a 2-reptile.*

It is easy to see that there is just one way of dissecting a a simplex into two simplices. So supposing that a 2-reptile simplex exists, one can quickly derive

a lot of information about the volume of its facets, the edge lengths, and the dihedral angles (Coxeter diagram). It is possible that a simple proof of Theorem 1 can be obtained using this information, but at least it didn't look obvious.

The proof presented here is based on eigenvalues of the linear transformations associated with the similarity maps that send the simplex onto the two tiles. This approach arose naturally while considering a problem associated with the probabilistic packet marking; that problem is more general than the existence of m-reptile simplices and geometric quantities such as facet volume or edge length cannot be directly used in it. Perhaps this approach could be of some interest in tiling problems.

2 Proof of Theorem 1

We proceed by contradiction, assuming that S is a d-dimensional 2-reptile simplex. Let $v_1, v_2, \ldots, v_{d+1} \in \mathbf{R}^d$ be the vertices of S. Let S_1 and S_2 be the two tiles; that is, $S = S_1 \cup S_2$, where S_1 and S_2 are congruent, similar to S, and have disjoint interiors. Here is the observation mentioned in the introduction:

Lemma 1. *The numbering of the vertices of S can be chosen in such a way that S_1 has the vertex set $v_1, v_2, \ldots, v_d, w$, where $w = \frac{1}{2}(v_d + v_{d+1})$, and S_2 has the vertex set $v_1, v_2, \ldots, v_{d-1}, v_{d+1}, w$; see Fig. 3.*

Proof. The interiors of S_1 and of S_2 are disjoint convex sets, and so they can be separated by a hyperplane h. Let V^0, V^+, V^- be the subsets of vertices of S lying on h, on one side of h, and on the other side of h, respectively. Each edge of S connecting a vertex from V^+ to a vertex from V^- gives rise to a vertex of both S_1 and S_2. Hence $|V^+|+|V^0|+|V^+|\cdot|V^-| = d+1$ and $|V^-|+|V^0|+|V^+|\cdot|V^-| = d+1$, and we get $|V^+| = |V^-| = 1$, $|V^0| = d-1$. Hence h contains $d-1$ vertices of S and it has to bisect the edge connecting the two remaining vertices so that S_1 and S_2 have the same volume. □

From now on, we assume that the vertex sets of S_1 and of S_2 are as in the lemma. Let f_k be the similarity map sending S to S_k, $k = 1, 2$. That is, f_k is an isometry followed by scaling in the ratio $\alpha = 2^{-1/d}$ (since the volume of S_k is half of the volume of S).

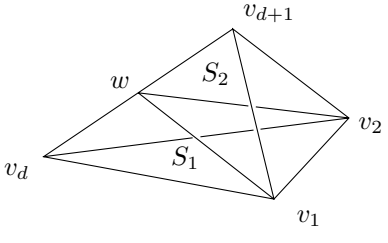


Fig. 3. The dissection of S into S_1 and S_2

Let us write $v'_i = v_i$ for $i = 1, 2, \ldots, d$ and $v'_{d+1} = w$, and let π_1 be the permutation of $\{1, 2, \ldots, d+1\}$ defined by $f_1(v_i) = v'_{\pi_1(i)}$, $i = 1, 2, \ldots, d+1$.

Lemma 2. *The permutation π_1 either has a single cycle (of length $d+1$), or it has d as a single fixed point and $1, 2, \ldots, d-1$, and $d+1$ form a single cycle of length d.*

Proof. Let us suppose that π_1 has at least two cycles, and let U be the union of all cycles that do not contain $d+1$. Then the face F of S spanned by the vertices v_i, $i \in U$, is fixed (setwise) by f_1. Since the diameter of the image of S under the iterated mapping $f_1 \circ f_1 \circ \cdots \circ f_1$ (t-fold composition) tends to 0 as $t \to \infty$, F has to be 0-dimensional, and hence $|U| = 1$. So either π_1 is unicyclic or it has one fixed point, different from $d+1$, and one cycle of length d.

Let j be the fixed point of π_1 and let F_j be the facet of S not containing v_j. If $j \neq d$, then $f_1(F) \subset F$, and again the diameter of the image of S under iterations of f_1 would not shrink to 0. Hence $j = d$ as claimed. □

We define the permutation π_2 analogously using f_2. That is, we set $v''_i = v_i$ for $i \in \{1, 2, \ldots, d-1, d+1\}$ and $v''_d = w$, and we define π_2 by $f_2(v_i) = v''_{\pi_2(i)}$. The analogy of Lemma 2 holds for π_2; i.e., π_2 either is unicyclic or has a fixed point $d+1$ and a cycle of length d.

Let us write $f_k(x) = A_k x + b_k$, $k = 1, 2$, for $d \times d$ matrices A_1 and A_2. Since f_k is an isometry followed by scaling by α, A_k is an orthogonal matrix multiplied by α, and hence all of its eigenvalues have absolute value α. More generally, for any sequence $(k_1, k_2, \ldots, k_{2t})$ of integers, the product matrix $A_1^{k_1} A_2^{k_2} \cdots A_2^{k_{2t}}$ must have all eigenvalues with absolute value $\alpha^{k_1+k_2+\cdots+k_{2t}}$, since the corresponding composed map is a similarity map with the appropriate ratio. We will show that these conditions cannot hold.

The eigenvalues of matrices are preserved under similarity (of matrices, i.e. TAT^{-1} is similar to A; this should not be confused with similarity maps considered elsewhere in this paper). Hence we can choose a basis of $\mathbf{R}^d$ as convenient: we will assume that $v_{d+1} = 0$ (which can be achieved by translation of S) and we will use the basis $(v_1, v_2, \ldots, v_d)$. Expressing f_1 and f_2 with respect to this basis, we obtain explicit and simple matrices $\bar{A}_1$ and $\bar{A}_2$, depending only on the permutations π_1 and π_2, which have to satisfy the eigenvalue condition formulated above for A_1 and A_2. We will now compute the characteristic polynomials. The possibility of unicyclic π_1 can be excluded based on the eigenvalues of $\bar{A}_1$ alone, while for π_1 with a single fixed point we will use the product $\bar{A}_2^{-1}\bar{A}_1$ as well.

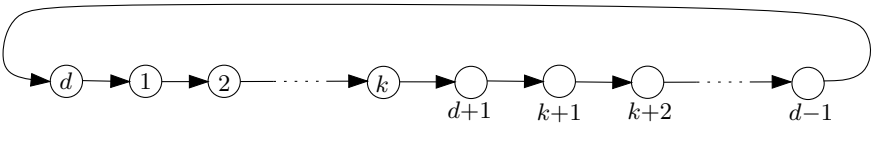


Fig. 4. The permutation π_1 in the unicyclic case

The Unicyclic Case. Here we assume that π_1 has a single cycle.We note that while the vertices v_d and v_{d+1} are distinguished (by the split edge), the roles of the others are completely symmetric, and so the only thing that matters is number k of elements between d and $d+1$ in the single cycle of π_1, $0 \le k \le d-1$. Let us fix the numbering of v_1 through v_{d-1} so that π_1 looks as in Fig. 4. The ith column of $\bar{A}_1$ is $f_1(v_i) - f_1(v_{d+1}) = v'_{\pi_1(i)} - v'_{\pi_1(d+1)}$ expressed in the basis $(v_1, \ldots, v_d)$. Here the coordinate vector of v'_i with respect to this basis is e_i (the standard basis vector) for $1 \le i \le d$ and it is $\frac{1}{2}e_d$ for $i = d$ (we have $v'_{d+1} = w = \frac{1}{2}(v_d + v_{d+1}) = \frac{1}{2}v_d$).

The matrices $\bar{A}_1$ look as follows (they are shown for $d = 5$ and $k = 0, 1, 2$ and 4):

$$\begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & \frac{1}{2} \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 1 & 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & -1 & -1 & \frac{1}{2} & -1 \end{pmatrix}.$$

The characteristic polynomial, easily calculated, is

$$\begin{aligned} p_k(x) &= -\left(x^d + x^{d-1} + \cdots + x^{d-k+1} + \frac{x^{d-k}}{2} + \frac{x^{d-k-1}}{2} + \cdots + \frac{1}{2}\right) \\ &= \frac{2x^{d+1} - x^{d-k} - 1}{2(1-x)}. \end{aligned}$$

Lemma 3. *For $d \ge 3$ and for every $k = 0, 1, \ldots, d-1$, there is at least one root of $p_k(x)$ with absolute value different from $\alpha = 2^{-1/d}$.*

We postpone the proof to the end of this section. This verifies that the case of a unicyclic permutation yields no 2-reptile simplices.

The Case of a Single Fixed Point. Here $\pi_1(d) = d$ and the remaining points form a single cycle. Since $v_1, v_2, \ldots, v_{d-1}$ are completely symmetric, we can choose the notation so that $\pi_1(1) = 2$, $\pi_1(2) = 3$,..., $\pi_1(d-2) = d-1$, $\pi_1(d-1) = d+1$, and $\pi_1(d+1) = 1$. The characteristic polynomial of the matrix $\bar{A}_1$ turns out to be $x^d - \frac{1}{2}$, and it does have all roots with absolute value α. This means, as is easy to check, that one can indeed choose the vertices $v_1, \ldots, v_{d+1}$ so that the mapping f_1 determined by π_1 as above is a similarity with ratio α. We thus have to consider both f_1 and f_2 simultaneously.

Since we have excluded the unicyclic case, π_2 also has to consist of one cycle of length d and a single fixed point, which in this case is $d+1$ (v_{d+1} plays the same

role in S_2 as v_d does in S_1). However, if we consider π_1 and π_2 simultaneously, we can no longer choose the numbering of the vertices along the cycle of π_2, as we did for π_1.

In order to determine the possibilities for π_2, we will consider the volumes of the facets of S, S_1, S_2. Let F_i be the facet of S that does not contain v_i, let V_i be its $(d-1)$-dimensional volume, let F_i' be the facet of S_1 not containing v_i', of volume V_i', and let F_i'' and V_i'' be defined analogously for S_2.

Lemma 4. *If π_1 is as above and π_2 has fixed point $d+1$, then we have $V_i = 2^{i/d} \cdot V_d$, $i = 1, 2, \ldots, d-1$ and $V_{d+1} = V_d$.*

Proof. By the geometry of the slicing of S into S_1 and S_2 (see Fig. 3) we have $V_d' = V_{d+1}''$ (this facet is shared by S_1 and S_2), $V_{d+1}' = V_{d+1}$ (the facet shared by S_1 and S), $V_d'' = V_d$ (the facet shared by S_2 and S), and $V_i' = V_i'' = \frac{1}{2}V_i$ for $i = 1, 2, \ldots, d-1$ (these facets of S are halved by the slicing).

Since f_1 and f_2 change $(d-1)$-dimensional volumes by the factor α^{d-1}, we have $V'_{\pi_1(i)} = V''_{\pi_2(i)} = \alpha^{d-1}V_i$. Since $\pi_1(d) = d$, $\pi_2(d+1) = d+1$, and $V_d' = V_{d+1}''$, we obtain $V_d = V_{d+1}$.

Next, from $V_2' = \frac{1}{2}V_2$ and $V_2' = V'_{\pi_1(1)} = \alpha^{d-1}V_1$ we get $V_2 = 2\alpha^{-(d-1)}V_1 = 2 \cdot 2^{-(d-1)/d}V_1 = 2^{1/d}V_1$. Similarly we calculate $V_3, \ldots, V_{d-1}$ and V_{d+1} and obtain the values in the lemma. □

The lemma shows that the facet volumes are uniquely determined by π_1. Since $V_1, \ldots, V_{d-1}$ are all distinct, π_1 can be reconstructed from the facet volumes (in other words, if the numbering of $v_1, \ldots, v_{d-1}$ is hidden, we can reconstruct it from the facet volumes). By symmetry, π_2 is also uniquely determined by the facet volumes, and we have $\pi_2(1) = 2$, $\pi_2(2) = 3$,..., $\pi_2(d-1) = d$, $\pi(d) = 1$ (and $\pi_2(d+1) = d+1$).

We can now write down both the matrices $\bar{A}_1$ and $\bar{A}_2$ (we show them again for $d = 5$):

$$\bar{A}_1 = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 1 \end{pmatrix}, \quad \bar{A}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 \end{pmatrix}.$$

The matrix $\bar{A}_2^{-1}\bar{A}_1$ has the following particularly simple form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix}.$$

The characteristic polynomial is $(1-x)^{d-2}(x^2-2x+3)$. By the necessary condition on $\bar{A}_1$ and $\bar{A}_2$, all roots should have absolute value 1, but the roots of x^2-2x+3 are $1 \pm i\sqrt{2}$. This concludes the proof of Theorem 1. □

Proof of Lemma 3. We use (the first part of) the following criterion due to Lehmer [11]:

For a polynomial g of the form $g(z) = a_n z^n + a_{n-1}z^{n-1} + \cdots + a_0$ we define the polynomial $T(g)$ by $T(g)(z) = \overline{a}_0 g(z) - a_n z^n \overline{g(z^{-1})}$, where the bar above a symbol denotes complex conjugation. Let g be a polynomial that has no (complex) root on the unit circle $\Gamma = \{z \in \mathbf{C} : |z| = 1\}$ and such that $g(0) \neq 0$. If, for some integer $h > 0$, $T^h(g)(0) < 0$, then g has at least one root inside Γ (T^h denotes an h-fold composition of the operator T). If $T^i(g)(0) > 0$ for $i = 1, 2, \ldots, h$ and $T^h(g)$ is a constant polynomial, then g has no root inside Γ.

We note that $T(g)(0) = |a_0|^2 - |a_n|^2$, which will be useful later.

We apply the criterion to show that $q_k(x) = 2x^{d+1} - x^{d-k} - 1$, the numerator in the second expression for $p_k(x)$, has a root strictly inside the circle $\Gamma' = \{z \in \mathbf{C} : |z| = 2^{-1/d}\}$. To this end, we check by Lehmer's criterion that for some $\beta < 2^{-1/d}$ the polynomial $g_k(z) = q_k(\beta z)$ has a root in the unit circle. We calculate $g_k(z) = 2\beta^{d+1}z^{d+1} - \beta^{d-k}z^{d-k} - 1$, and $T(g_k)(z) = \beta^{d-k}z^{d-k} + 2\beta^{2d+1-k}z^{k+1} - (2\beta^{d+1})^2 + 1$.

For β sufficiently close to $2^{-1/d}$, the absolute term of $T(g_k)(z)$ satisfies $1 - (2\beta^{d+1})^2 \leq 1 - 2^{-2/d} + \varepsilon \leq 1 - 2^{-2/3} + \varepsilon < \frac{1}{2}$. The leading term of $T(g_k)(z)$ is

$$\begin{array}{ll} \beta^{d-k} & \text{for } d > 2k+1, \\ 2\beta^{2d+1-k} & \text{for } d < 2k+1, \\ \beta^{(d+1)/2} + 2\beta^{3d/2+1} & \text{for } d = 2k+1. \end{array}$$

In all three cases the leading term is at least $\frac{1}{2}$, and hence $T^2(g_k)(0) < 0$. Lehmer's criterion shows that $p_k(x)$ has a root with absolute value below $2^{-1/d}$ and the lemma is proved. □

3 Probabilistic Packet Marking

In this section we outline some ideas from [1] and [2] directly related to the present paper.

The *denial-of-service attacks* on the Internet operate by sending an enormous number of packets to the attacked computer through the network. For defense against such attacks it would be very helpful to trace such packets back to their source. Internet packets contain the address of origin, but that can easily be forged. Along its path through the network, each packet passes through several routers, so the source could be traced if each router added its address to each packet passing through it. However, the packets have a standard format which doesn't leave much room for information added by the routers, and even if the packet format could be changed, sending along the addresses of all routers would burden the network significantly. The idea of *probabilistic packet marking* (suggested in [5], with the first actual schemes given in [13] and [7]) is to take

advantage of the large number of packets used in attacks and encode information about the route into the *probability distribution* of the values of a small number of bits in the packets.

We explain a simple and ingenious packet marking scheme from [1] (ignoring some technical issues). For simplicity we assume that just one bit B is available for the marking in each packet, with possible values $B = 0$ and $B = 1$. We assume that in some considered period, all packets reaching some destination pass through routers $R_1, R_2, \ldots, R_n$ in this order. Each R_i wants to "record" a one-bit message r_i, with value 0 or 1, into B (actual addresses of routers have more than 1 bit, of course, but we want to explain the principle). It proceeds according to the following protocol.

- For $r_i = 0$: A received packet with $B = 0$ is sent further with $B = 0$. A received packet with $B = 1$ is sent further with B set at random, to $B = 0$ with probability $\frac{1}{2}$ and to $B = 1$ with probability $\frac{1}{2}$ (the random choice is independent of the choices for other packets and in the other routers).
- For $r_i = 1$: A received packet with $B = 1$ is sent further with $B = 1$. A received packet with $B = 0$ is sent further with B set at random, again to 0 or 1 with probability $\frac{1}{2}$ each.

According to this protocol, if the packets incoming to R_i have probability x of having $B = 1$, then the outgoing packets have probability of $B = 1$ equal to $\frac{1}{2}x$ if $r_i = 0$, and equal to $\frac{1}{2} + \frac{1}{2}x$ if $r_i = 1$. Thus, if we interpret the probability distribution of B as a point $x \in [0, 1]$, then for $r_i = 0$, the router R_i performs the affine map $g_0 \colon x \mapsto \frac{1}{2}x$, while for $r_i = 1$ it performs the affine map $g_1 \colon x \mapsto \frac{1}{2} + \frac{1}{2}x$. If the initial probability distribution of B entering R_1 is given by $y \in [0, 1]$, then distribution "seen" at the destination after R_n is given by $x = 2^{-1}r_n + 2^{-2}r_{n-1} + \cdots + 2^{-n}r_1 + 2^{-n-1}y$, and hence the messages of the routers can be read off as the n most significant bits of x. (The technical issues ignored here include how precisely x can be determined, what are effects of "noisy" packets coming along alternative routes, etc.)

Geometrically, the above packet marking scheme is based on the tiling of $[0, 1]$ by the two intervals $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$. Now one may want to generalize the scheme: First, more than one bit may be available in each packet for the marking; generally one can assume that the marking part of the packet may attain $d + 1$ distinct values. Then the probability distribution is specified by a point in the standard d-dimensional simplex $\Delta_d = \{x \in \mathbf{R}^{d+1} : x_1, \ldots, x_{d+1} \geq 0, x_1 + x_2 + \cdots + x_{d+1} = 1\} \subset \mathbf{R}^{d+1}$, where x_j represents the probability of the jth value among the $d + 1$ possible values. And second, the messages of the routers may attain some number m of distinct values, say $1, 2, \ldots, m$. In such a situation, we would need m affine maps $g_1, g_2, \ldots, g_m \colon \Delta_d \to \Delta_d$, where a router with message i converts an incoming probability distribution $x \in \Delta_d$ into the probability distribution $g_i(x)$ (it is easy to check that any affine map $\Delta_d \to \Delta_d$ can be realized by a suitable protocol, which works with one packet at a time and for which one need not know the incoming probability distribution). We would like to choose the g_i so that the messages of all routers along the path can

be reconstructed from the probability distribution at the destination, allowing for as large error margin in reading off that distribution as possible.

The exact requirements on the g_i can be discussed at great length and are a subject for further research. Here we mention a condition which, if it can be satisfied, yields "asymptotically optimal" marking schemes.

The simplices $g_1(\Delta_d)$, $g_2(\Delta_d)$,..., $g_m(\Delta_d)$ tile Δ_d without overlap, and there exists a constant $c > 0$ such that for any finite sequence $i_1, i_2, \ldots, i_n$, $1 \le i_j \le m$, the composed map $g = g_{i_1} \circ g_{i_2} \circ \cdots \circ g_{i_n}$ doesn't contract distances too much: it satisfies $\|g(x) - g(y)\| \ge c \cdot m^{-n/d} \cdot \|x - y\|$ for every $x, y \in \Delta_d$.

It is easy to check that if S is a d-dimensional simplex that is an m-reptile, $f_1, \ldots, f_n$ are the similarity maps sending S to the tiles $S_1, \ldots, S_m$, and T is an affine isomorphism $S \to \Delta_d$, then the maps $g_1, \ldots, g_m$ given by $g_i = T \circ f_i \circ T^{-1}$ satisfy the above requirement and thus lead to an asymptotically optimal protocol. Unfortunately, so far it seems that simplex m-reptiles with small m, which would be useful in this context, exist very seldom.

Finally, let us remark that the above condition for the g_i implies the eigenvalue condition formulated in the proof of Theorem 1 (in the special case $m = 2$). Thus, the part of that proof using only the eigenvalue condition also applies to the existence of the maps g_i for an asymptotically optimal protocol. However, the part considering the facet volumes does not seem to apply directly in this greater generality.

Acknowledgment

I thank Micah Adler and Uri Zwick for discussions about probabilistic packet marking, which were a starting point of the research reported in this paper. For kind answers to my queries about reptiles, matrices, and similar things I would like to thank Christoph Bandt, Maurice Cochand, David Eppstein, Eike Hertel, Krystyna Kuperberg, Włodek Kuperberg, and Petr Plecháč, and finally, I thank Petr Škovroň for pointing out a gap in an earlier version of the proof.

References

1. M. Adler. Tradeoffs in probabilistic packet marking for IP traceback. In *Proc. 34th Annu. ACM Symposium on Theory of Computing*, pages 407–418, 2002.
2. M. Adler, J. Edmonds, and J. Matoušek. Towards asymptotic optimality in probabilistic packet marking. In *Proc. 37th Annu. ACM Symposium on Theory of Computing*, pages 450–459, 2005.
3. G. L. Alexanderson and J. E. Wetzel. Dissections of a simplex. *Bull. Amer. Math. Soc.*, 79:170–171, 1973.
4. C. Bandt. Self-similar sets. V. Integer matrices and fractal tilings of $\mathbf{R}^n$. *Proc. Amer. Math. Soc.*, 112(2):549–562, 1991.
5. H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. Contribution at the conference Usenix LISA (New Orleans), 2000.
6. H. E. Debrunner. Tiling Euclidean d-space with congruent simplexes. In *Discrete geometry and convexity (New York, 1982)*, volume 440 of *Ann. New York Acad. Sci.*, pages 230–261. New York Acad. Sci., New York, 1985.

7. T. W. Doeppner, P. N. Klein, and A. Koyfman. Using router stamping to identify the source of IP packets. In *Proc. 7th ACM conference on Computer and communications security*, pages 184–189, 2000.
8. G. Gelbrich. Crystallographic reptiles. *Geom. Dedicata*, 51(3):235–256, 1994.
9. G. Gelbrich. Self-affine lattice reptiles with two pieces in $\mathbf{R}^n$. *Math. Nachr.*, 178:129–134, 1996.
10. E. Hertel. Self-similar simplices. *Beiträge Algebra Geom.*, 41(2):589–595, 2000.
11. D. H. Lehmer. A machine method for solving polynomial equations. *J. Assoc. Comput. Mach.*, 8:151–162, 1961.
12. S.-M. Ngai, V. F. Sirvent, J. J. P. Veerman, and Y. Wang. On 2-reptiles in the plane. *Geom. Dedicata*, 82(1-3):325–344, 2000.
13. S. Savage, D. Wetherall, A. Karlin, , and T. Anderson. Practical network support for IP traceback. In *Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM)*, pages 295–306, 2000.
14. S. L. Snover, C. Waiveris, and J. K. Williams. Rep-tiling for triangles. *Discrete Math.*, 91(2):193–200, 1991.
15. T. Zaslavsky. Maximal dissections of a simplex. *J. Combinatorial Theory Ser. A*, 20(2):244–257, 1976.

Single-Vertex Origami and Spherical Expansive Motions

Ileana Streinu[1,⋆] and Walter Whiteley[2,⋆⋆]

[1] Computer Science Department, Smith College, Northampton, MA 01063, USA
streinu@cs.smith.edu, http://cs.smith.edu/~streinu
[2] Department of Mathematics, York University, Toronto, M3J 1P3, Canada
whiteley@mathstat.yorku.ca, http://mathstat.yorku.ca/~whiteley

Abstract. We prove that all single-vertex origami shapes are reachable from the open flat state via simple, non-crossing motions. We also consider *conical* paper, where the total sum of the cone angles centered at the origami vertex is not 2π. For an angle sum less than 2π, the configuration space of origami shapes compatible with the given metric has two components, and within each component, a shape can always be reconfigured via simple (non-crossing) motions. Such a reconfiguration may not always be possible for an angle sum larger than 2π.

The proofs rely on natural extensions to the sphere of planar Euclidean rigidity results regarding the existence and combinatorial characterization of expansive motions. In particular, we extend the concept of a pseudo-triangulation from the Euclidean to the spherical case. As a consequence, we formulate a set of *necessary conditions* that must be satisfied by three-dimensional generalizations of pointed pseudo-triangulations.

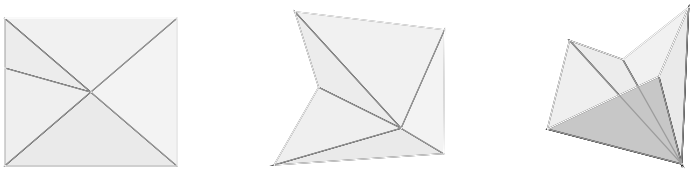
1 Introduction

Imagine making creases in a flat sheet of paper, all of them originating at a single vertex, and then folding along the creases *without tearing, stretching or bending the paper*, to obtain a three dimensional origami shape. Assume that the planar regions bounded by creases behave more like metal sheets than paper, i.e. they move *rigidly*, and do not go through each other during the motion. See Fig. 1. The *single-vertex origami problem* asks: *are there origami shapes which are compatible with the creases and the induced metric of the paper, but which cannot be* folded *by such a process?*

Here, we answer this question and generalize it in several ways. We prove that all single-vertex origami shapes can be folded from the flat state. This implies that the configuration space of all origami shapes with the same crease pattern is *connected*: any shape can be reconfigured into any other shape, via simple (non-self-intersecting) motions. We begin by formulating the problem

⋆ Supported by NSF grants CCR-0105507 and NSF-DARPA CARGO-0310661.
⋆⋆ Supported by grants from NSERC (Canada) and NIH (US).

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 161–173, 2005.



(a) (b) (c)

Fig. 1. A single-vertex origami fold: (a) the creased sheet of paper; (b, c) two of its possible folded shapes

in terms of conical *panel-and-hinge structures*, which have incident hinge-axes and in terms of *spherical polygonal linkages*. If the spherical perimeter (defined below) is no more than 2π, we show that then they can be reconfigured to a spherically convex configuration. But if the perimeter is more than 2π, then the configuration space may be disconnected. We leave open the question of whether simple spherical polygonal *paths* of length between π and 2π, with *short* links (less than π) is connected or not.

Our proofs rely on a generalization to the sphere of the planar *Carpenter's Rule Problem*, which asks whether every simple planar polygon can be unfolded to a convex position, in such a way that the edges maintain their lengths and do not cross throughout the motion. An *expansive motion* never decreases any inter-distance between two points, therefore no collisions may occur. The two solutions of the Carpenter's Rule problem for simple planar polygonal linkages [4,17] rely on expansive motions. In dimension two, the infinitesimal expansive motions are well understood. They form a polyhedral cone [15] whose extreme rays have a combinatorial characterization, given by pointed pseudo-triangulation mechanisms [17]. Their three-dimensional counterparts also form a polyhedral cone, defined by similar linear inequalities, but finding a combinatorial interpretation for its rays has so far remained elusive.

In this paper, we also initiate the study of expansive motions on the sphere and in 3d and give the first provable classes of spherical and 3d expansive mechanisms with one-degree-of-freedom: hemispherical pseudo-triangulations, resp. pointed *cone pseudo-triangulations*. By applying them to the spherical Carpenter's Rule Problem of perimeter $\leq 2\pi$, we obtain the proof that all *simple* folds can be opened to a spherical convex position.

Historical Background. Computational origami is a relatively recent endeavor, see [6] for a survey. The mathematical and computational origami literature addresses questions of feasibility, characterization and NP-hardness of flat-folds, as well as applications, see e.g. [13,1,7]. According to Tom Hull [10], only two published articles deal with the mathematics of rigid origami [9,14].

The topology of the configuration space for spherical linkages and single-vertex origami folds (allowing self-crossings) is studied in [12,11].

Polygonal linkages in the plane have received a lot of attention in recent years. Relevant for our paper are the previously mentioned results on the Carpenter's

Rule Problem using expansive motions [4] and pseudo-triangulations [17]. To the best of our knowledge, there are no other results studying or applying spherical or 3d expansive motions.

Rigidity for non-Euclidean geometries (including spherical and hyperbolic) is the topic of an unpublished paper of the second author [16]. The projective connections between motions of cones, motions on spheres, and motions in plane projections go back to [19]. We achieve the extension of the Carpenter's Rule problem to spherical polygons and single-vertex origami folds via spherical geometry techniques, building on ideas from [19,16] but handling signed motions (expansive and contractive).

2 Definitions and Preliminaries

For rigidity theoretic terminology and concepts, we refer the reader to the classical monograph [8] and the handbook chapter [20]. For pseudo-triangulations, see [17] and [15].

Frameworks in Two and Three Dimensions. A bar-and-joint framework (simply, a framework) $G(p)$ is a graph $G = (V, E)$, $V = [n] := \{1, \cdots, n\}$, embedded on a set of points $p = \{p_1, \cdots, p_n\}$, $p_i \in R^d$. In this paper $d = 2$ or $d = 3$. A pair of indices $(i, j) \in [n]^2$ may be denoted simply as ij. The embedded edges (segments) p_ip_j are also called *bars*, and their endpoints p_i are called *joints*. When the underlying graph is a path or a cycle, the framework is called a *chain* (open, resp. closed).

Infinitesimal rigidity of frameworks. An *infinitesimal motion* of a framework $G(p)$ is a set of velocity vectors $v = \{v_1, \cdots, v_n\}$, $v_i \in R^d$ preserving the lengths of the bars:

$$\langle p_i - p_j, v_i - v_j \rangle = 0, \forall ij \in E$$

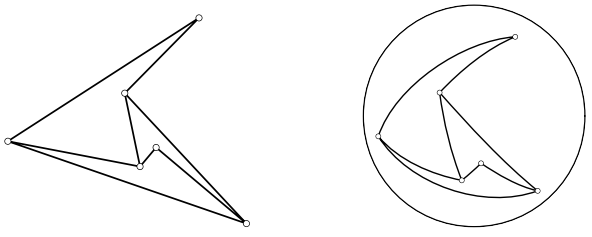
An infimitesimal motion is *trivial* if it is a rigid transformation of the whole space. A framework is *infinitesimally rigid* if it has only trivial infinitesimal motions, and *infinitesimally flexible* otherwise.

A *flex* or motion of a framework is a set of continuous point trajectories

$$p(t) = \{p_1(t), \cdots, p_n(t)\}$$

which preserve the edge lengths $l_{ij}^2 = \langle p_i - p_j, p_i - p_j \rangle, \forall ij \in E$ of the initial framework $G(p) = G(p(0))$ at any moment in time t:

$$\langle p_i(t) - p_j(t), p_i(t) - p_j(t) \rangle = l_{ij}^2, \forall ij \in E$$



(a) (b)

Fig. 2. A pointed pseudo-triangulation mechanism (a) in the plane, and (b) on the sphere

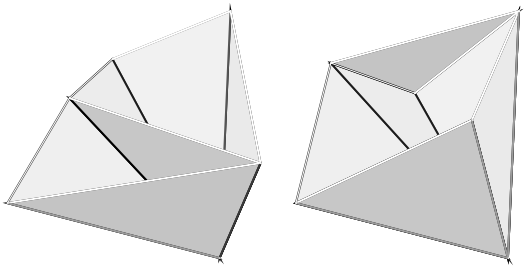
Expansion and Contraction. Given a point set p and infinitesimal velocities v, let us denote by ε_{ij} the quantity $\varepsilon_{ij}(p, v) := \langle p_i - p_j, v_i - v_j \rangle$. For a pair ij of indices (not necessarily an edge of a graph G), we say that the diagonal ij *expands* if $\varepsilon_{ij} > 0$, *contracts* if $\varepsilon_{ij} < 0$ or is *frozen* if $\varepsilon_{ij} = 0$. An infinitesimal motion v of $G(p)$ is *expansive* (resp. *contractive*) if all the non-frozen diagonals expand (resp. contract). A framework is infinitesimally *expansive* if it is infinitesimally flexible and supports a non-trivial infinitesimally expansive motion.

Pointed Pseudo-Triangulations and Mechanisms. A special class of planar frameworks are those with no crossing edges and where every vertex is *pointed*: incident to an angle larger than π. Such frameworks are planar graph embeddings and can have at most $2n - 3$ edges. When they have the maximum number of edges, the outer face is convex and all the internal faces are *pseudo-triangles*: simple polygons with exactly three inner convex angles. As frameworks, pointed pseudo-triangulations are *minimally rigid*. Removing any edge makes then *flexible* mechanisms, with one degree of freedom. If the removed edge is a convex hull edge, the mechanism is *expansive*: the unique motion that increases the length of the removed convex hull edge, never decreases any distance between a pair of points. See [17] and Fig. 2(a).

Panel and Hinge Structures. A bounded *panel* is a simple polygon embedded flat in 3d, and intended to behave like a rigid object, i.e. to remain flat and to maintain its metric properties (edge lengths and angles). The simplest case is a triangle. We will also work with unbounded panels which are wedges defined by two semi-infinite rays.

A *hinge* is a line segment or ray common to at least two panels. In this paper we consider only hinges that appear as complete bounding edges of their incident panels. In particular, a hinge does not run through the interior of a panel and does not extend beyond its boundary. The panels are attached rigidly to hinges: they are allowed to rotate about hinges, but not to slide along them.

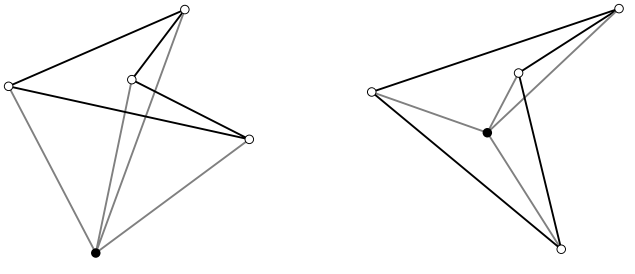
A *panel-and-hinge* structure is a collection of panels connected via hinges. Examples include those in Fig. 1, where all hinges are concurrent, and Fig. 3, where each hinge is incident to two panels.



(a) (b)

Fig. 3. Examples of panel-and-hinge structures. (a) Some hinges may be incident with several panels. (b) A triangulated polyhedral surface is a special case of a panel and hinge structure.

Conical Panel-and-Hinge Structures. In this paper we work only with *conical panel-and-hinge structures*, in which all the hinges are concurrent to a unique vertex called the *cone vertex*, as in Fig. 3(a). In this case, each panel contains at most two hinges. The conical structures are *bounded*, if all panels are bounded polygons and the hinges are line segments, as in Fig. 3(a), or *unbounded*, when the hinges are infinite rays and the panels are wedges. For the purpose of this paper, in the bounded case it suffices to work only with triangular faces. We distinguish two cases, *pointed* and *non-pointed conical structures*, depending on whether the cone vertex is pointed (all incident segments are contained in a half-space defined by a plane through the cone vertex) or not. See Fig. 4.



(a) (b)

Fig. 4. (a) A pointed and (b) a non-pointed conical framework arising from conical panel-and-hinge structures. The cone vertex is black.

From Conical Panel-and-Hinge Structures to Bar-and-Joint Frameworks. To each bounded conical panel-and-hinge structure we associate a bar-and-joint framework $G_0(p)$ in 3d. The vertices of G_0 correspond to those of the conical structure and the edges to sides of the triangular panels. The cone-vertex is always labeled with 0, and is incident with all the other vertices, labeled from 1 to n. G_0 is embedded as $G_0(p)$ on the set of points bounding the triangular

panels, $p_i \in R^3$, $i = 0, 1, \cdots, n$. Such a bar-and-joint framework is called a *cone framework*.

An *infinitesimal motion* of a bounded conical panel-and-hinge structure is an infinitesimal motion of the associated conical bar-and-joint framework.

For unbounded conical panel-and-hinge structures, we first bound the panels by *cutting* them into triangles. Then we define the infinitesimal motion of the bounded structure. The following straightforward lemma shows that the motion can be uniquely extended to all the points of the original unbounded panels.

Lemma 1. *(***Infinitesimal velocities of arbitrary points).** *Any point p' situated on the supporting line of a hinge p_0p_i or on the supporting plane of a panel $p_0p_ip_j$ can be assigned an infinitesimal velocity v' uniquely determined by the velocities of the points bounding the hinge or panel.*

Spherical Frameworks. If we intersect an unbounded conical structure with a sphere centered at the cone vertex, we obtain a *spherical framework*. It consists of *spherical bars* along great-circle arcs on the sphere, and joints which are the intersection of the hinges with the sphere. The incidence structure of a spherical framework is a graph G obtained from G_0 by deleting the cone-vertex 0 and its incident edges. The remaining graph G has the vertex set $[n]$. The *length* of a spherical edge ij is the size of the angle $\angle p_ip_0p_j$ centered at the cone vertex.

A spherical framework fully contained in a hemisphere is called *hemispherical*. It arises from a pointed conical panel-and-hinge structure. See Fig. 5.

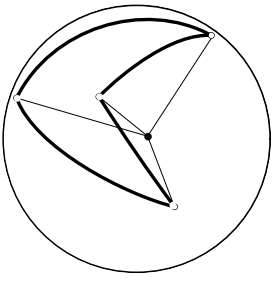


Fig. 5. A hemispherical framework, and the corresponding pointed conical panel-and-hinge structure. The cone-vertex (black) is the center of the sphere.

3 Infinitesimal Rigidity of Conical and Spherical Frameworks

In this section we show how to associate to every *planar framework* a 3d pointed *conical framework*, and therefore also a hemispherical framework and a pointed conical panel-and-hinge structure, and vice-versa. The association preserves infinitesimal rigidity and maintains the expansion pattern.

The Cone of a Planar Framework. Let $G(p)$ be a planar framework embedded in R^2, viewed as the affine plane $z = 1$ in R^3. The *cone* over G is the graph $G_0 = (V_0, E_0)$, $V_0 = V \cup \{0\}$ (0 is the *cone vertex*) and $E_0 = E \cup \{0i\}_{i \in V}$. The *canonical conical framework* $G_0(p)$ over $G(p)$ with center $p_0 \in R^3$ is an embedding of the cone G_0 over G so that it extends $G(p)$ and embeds the cone vertex at the center p_0. In this paper we take $p_0 = 0$. More generally, a cone framework $G_0(q)$ over $G(p)$ will have the cone vertex at q_0 and the ith vertex q_i on the line $p_0 p_i$.

Infinitesimal Motions of Planar and Conical Frameworks. The association we just described induces a *projection map* carrying a conical framework[1] $G_0(q)$ with no points on the plane $z = 0$ into a framework $G(p)$ in R^2 embedded in the plane $z = 1$ (and vice-versa). We now turn our attention to establishing and proving the connection between infinitesimal motions of a planar framework and the associated conical framework.

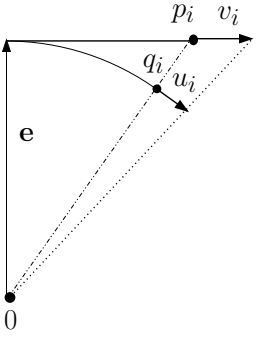


Fig. 6. The projection map, shown here in a section along the plane spanned by a point q_i on the sphere and the z-axis. It takes a point q_i on the sphere to a point p_i in the $z = 1$ plane, and an infinitesimal velocity u_i of q_i to an infinitesimal velocity v_i of p_i.

Let $G_0(q)$ be a conical framework with the cone vertex at the origin, $q = (q_0, q_1, \cdots, q_n)$, $q_0 = 0$ and $q_i = (x_i, y_i, z_i), z_i \neq 0$. Let $u = (u_0, u_1, \cdots, u_n)$, $u_0 = 0$ be an infinitesimal motion of the framework $G_0(q)$ in R^3, with the cone vertex pinned down. Let $\mathbf{e}$ the unit vector along the z-axis. See Fig. 6.

Proposition 1. *Consider the map $R^3 \mapsto R^2$ taking the points $q_i = (x_i, y_i, z_i)$ on the sphere to points $p_i = (\frac{x_i}{z_i}, \frac{y_i}{z_i}, 1)$ in the $z = 1$ plane and velocities u_i to:*

$$v_i = \frac{1}{z_i} \left(u_i - \langle u_i, \boldsymbol{e} \rangle \boldsymbol{e} \right)$$

[1] In this section, we denote by p a planar point set and by q a 3d point set.

Then $u = (0, u_1, \cdots, u_n)$ is an infinitesimal motion of the conical framework $G_0(q)$ in R^3 iff $v = (v_1, \cdots, v_n)$ is an infinitesimal motion in R^2 of the framework $G(p)$.

If two points q_i and q_j are in the same hemisphere ($z > 0$ or $z < 0$), then u is expansive for the pair q_i, q_j iff v is expansive on p_i, p_j in the plane. If q_i and q_j are on opposite hemispheres, then u is expansive for the pair q_i, q_j iff v is contractive on p_i, p_j in the plane.

Proof. The bars from the cone vertex give the equations $\langle q_i - q_0, u_i - u_0\rangle = \langle q_i, u_i\rangle = 0$. Applying them on i and j, the expansion values become:

$$\langle p_i - p_j, v_i - v_j\rangle = \langle \frac{q_i}{z_i} - \frac{q_j}{z_j}, \frac{1}{z_i}(u_i - \langle u_i, \mathbf{e}\rangle \mathbf{e}) - \frac{1}{z_j}(u_j - \langle u_j, \mathbf{e}\rangle \mathbf{e}) =$$

$$= -\frac{\langle q_i, u_j\rangle + \langle q_j, u_i\rangle}{z_i z_j} + [\frac{\langle u_i, \mathbf{e}\rangle}{z_i} - \frac{\langle u_j, \mathbf{e}\rangle}{z_j} - \frac{\langle u_i, \mathbf{e}\rangle}{z_i} + \frac{\langle u_j, \mathbf{e}\rangle}{z_j}] =$$

$$= -\frac{\langle q_i, u_j\rangle + \langle q_j, u_i\rangle}{\langle \mathbf{e}, q_i\rangle\langle \mathbf{e}, q_j\rangle} + \frac{1}{z_i z_j}[\langle q_i, u_i\rangle - \langle q_i, u_j\rangle - \langle q_j, u_i\rangle\langle q_j, u_j\rangle] = \frac{1}{z_i z_j}\langle q_i - q_j, u_i - u_j\rangle$$

Thus the sign (positive, negative or zero) of $\langle p_i - p_j, v_i - v_j\rangle$ is the same as the sign of $\langle q_i - q_j, u_i - u_j\rangle$ when the two points q_i and q_j are in the same hemisphere ($z_i z_j > 0$) and opposite when in different hemispheres ($z_i z_j < 0$). □

As a simple corollary we obtain:

Lemma 2. *Let $G(p)$ be a planar framework and let $G_0(p)$ be an associated 3-dimensional cone framework. Then:*

1. *If $G(p)$ is infinitesimally rigid, then so is $G_0(p)$.*
2. *If $G(p)$ is infinitesimally expansive, and the cone vertex is pointed, then $G_0(p)$ is also expansive.*

If the planar framework $G(p)$ is a pseudo-triangulation mechanism, then the spherical framework associated to the cone framework $G_0(p)$ will be called a *hemispherical pointed pseudo-triangulation mechanism.* See Fig. 2(b). Interpreting Lemma 2 in this case leads to:

Lemma 3. *A hemispherical pseudo-triangulation mechanism is expansive.*

As another corollary we also obtain the following simple necessary condition for 3d expansive motions. Given a graph G and a vertex i, let V_i be the vertex i together with the set of neighbors of i. The ith *star* G_i of G is the subgraph induced on V_i.

Lemma 4. *Let $G(q)$ be a spatial flexible framework and let $G_i(q)$ be the conical framework induced on the star G_i. If a conical framework $G_i(q)$ is not infinitesimally expansive, then $G(q)$ is not infinitesimally expansive. In other words, the following two local pointedness conditions are necessary for infinitesimal expansiveness:*

1. *The cone vertex of each star $G_i(q)$ is pointed (in $3d$), and*
2. *Each induced cone framework projects to a* pointed *and* non-crossing $2d$ *framework.*

Part 2 of the previous lemma relies on the existence of expansive motions for planar pointed graphs (which are subgraphs of pointed pseudo-triangulations), see [17,15]. Lemma 4 gives a necessary condition for a 3d framework to be expansive, and thus precludes any rigidity-theoretic generalization of planar pointed pseudo-triangulation mechanisms to 3d.

4 Convexifying Spherical Polygonal Linkages

A spherical framework is a graph G embedded on the surface of a sphere, with edges going along arcs of great circles. The *length* of an edge q_iq_j is the angle between the two line segments joining the center of the sphere to the two endpoints of the edge. To simplify the presentation, we will assume that edges have lengths different from 0 and π, i.e the endpoints of every edge are distinct and not-antipodal (the results hold even without this assumption).

A great-circle cuts the sphere into two hemispheres. With respect to a hemisphere, the defining great-circle is called its *equator*, and the point where the normal to the equator plane crosses the hemisphere is called the *pole* of the hemisphere.

A hemispherical framework is one lying on a hemisphere. A *proper* framework has all its edge lengths at most π. Of special interest are spherical and hemispherical polygons and polygonal paths. The perimeter of a polygon and the length of a polygonal path is the total sum of its edge lengths. A hemispherical polygon is *convex* if the projection on the plane going through the pole and parallel to the plane of the equator is a convex polygon.

Theorem 1. *Every simple hemispherical polygon of perimeter $\leq 2\pi$ and every simple hemispherical polygonal path of length $\leq \pi$ can be unfolded to a convex polygon using expansive motions.*

Proof. Note that any polygon of perimeter at most 2π must lie inside a closed hemisphere. If it lies inside an open hemisphere, we proceed with the following case. We return below to the extreme case, when there are antipodal points.

Project the polygon in an open hemisphere to a plane tangent to the pole of the hemisphere. Find an expansive infinitesimal motion of the planar polygon, e.g. one induced by a pointed pseudo-triangulation mechanism as in [17], or an arbitrary one obtained by a linear program as in [4]. Lemmas 2 and 3 imply that the induced cone framework is also moving expansively.

Move the spherical mechanism until an alignment event occurs, as in [17]. Notice that the alignment of two incident edges in a spherical pseudo-triangulation mechanism happens exactly when the projection on the $z = 1$ plane has the corresponding edges aligned (but beware, the projection does not move as a planar rigid framework). The combinatorial structure of the spherical and planar

pseudo-triangulations is the same. Therefore, as long as the polygon stays inside a hemisphere, there is always an expanding motion given by a pseudo-triangulation mechanism, and the convexification algorithm of [17] applies identically.

There is a concern that the motion might leave an open hemisphere. This occurs when two antipodal points appear. With two antipodal points in a polygon, there must be two paths joining the points, each of length at least π. This implies that the polygon must have length exactly 2π, and the two paths must each be great circle segments joining the antipodal points. In this situation, we may have the entire polygon on a single great circle, and we are finished with a flat spherical polygon. The alternative is that we have a 'wedge' formed by two intersecting great circles. Freezing one, we can rotate the other to make them flat, on opposite segments of a great circle. This motion is obviously expansive. Finally, we note that if we did not start with antipodal points, we can only achieve this configuration at an alignment event, so our previous process is complete and only requires a check for a wedge after each alignment event.

Finally, we note that if the algorithm terminates with a convex polygon which is not flat, then we have a *convex cone* from the center of the sphere which has a total angle of less than 2π. This means the spherical polygon had perimeter less than 2π. Therefore, any polygon of perimeter 2π will terminate with a flat configuration on a single great circle.

The polygonal path of length not more than 2π can be reduced to the closed polygon case by joining the endpoints of the path via a geodesic spherical path (which stays inside the hemisphere and bends at vertices and edges of the polygonal path). The total length of the added geodesic edges is at most the length of the polygon, so we have reduced to the previous case. Moreover, for any path less than π we could choose the additional edges to make the total length exactly 2π and terminate with a flat (collinear) polygonal path. If the length is exactly π, a limiting argument guarantees we can also achieve flatness. □

Let's emphasize as separate corollaries two ideas from the previous proof.

Lemma 5. *A spherical polygon of perimeter* 2π *can be unfolded to a great circle using expansive motions.*

Corollary 1. *If the geodesic line added between the endpoints of a spherical polygonal path doesn't increase the total length to more than* 2π*, then the polygonal path can be made flat on a great circle using expansive motions.*

The case not covered by the previous corollary, the hemispherical polygonal path (even when its total length is under 2π) may need contractive (or partially contractive) motions to convexify. We conjecture that the convexification is always possible.

Conjecture 1. Every spherical polygonal path of length at most 2π can be flattened without collisions onto a great circle.

Just as in the planar case, we obtain:

Corollary 2. *Two similarly oriented configurations of a simple spherical polygon with fixed edge lengths and perimeter* $\leq 2\pi$ *lie in the same component of the configuration space.*

Indeed, one can move each one into convex position and then reverse one unfolding path to achieve the reconfiguring motion.

What happens for spherical polygons and paths longer than 2π? They obviously cannot be convexified, since they do not fit on a great circle. The question is whether they can always be reconfigured to any other position. The following simple example shows that this may not be always possible.

Example 1. Consider a spherical quadrilateral with all edge lengths equal to $\frac{2\pi}{3} - \epsilon_i$, for four distinct values ϵ_i. Its perimeter is $\frac{8\pi}{3} - \sum \epsilon_i > 2\pi$, and any consecutive pair of edges adds to over π in length. The framework fits into a hemisphere in two ways, which cannot be reconfigured one to another without self-intersections.

Moreover, even when a reconfiguration os such spherical linkages is possible, a combination of expansive and contractive motions may have to be used.

5 Unfolding Single-Vertex Origami

A single-vertex origami is a creased piece of paper with all the creases incident to one vertex. Assume first that the vertex lies in the interior of the piece of paper. For simplicity, assume that the paper has a polygonal boundary with one vertex on each crease line. The total angular length of the corresponding spherical polygon or path framework is 2π in this case. We also consider the case when the polygonal piece of paper has the fold-vertex situated on a boundary edge or at a corner of the paper. The total angular length of the corresponding spherical polygon or path framework is π in the first case, and it can be either strictly less than or larger than 2π in the second case, depending on the corner angle being convex or reflex. See Fig. 7. Theorem 1 implies:

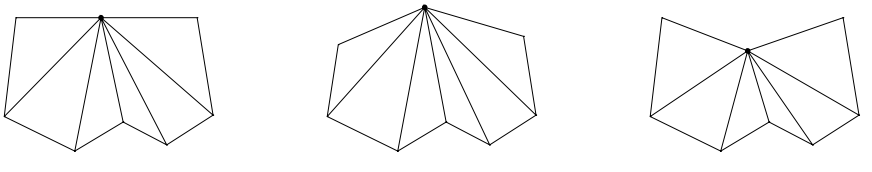


Fig. 7. Single-vertex origami with the fold-vertex on the boundary of the piece of paper: (a) on an edge, (b) at a convex corner and (c) at a reflex corner

Corollary 3. *Every simple single-vertex origami fold with the fold-vertex interior to the paper or interior to a boundary edge or situated at a convex vertex can be unfolded with expansive motions.*

Corollary 4. *The configuration space of simple single-vertex origamis with the fold-vertex interior to the paper, or to a boundary edge or situated at a convex vertex is connected. Two shapes can be reconfigured one into the other with simple, non-self-intersecting motions.*

Conjecture 1 extends to single-vertex origamis with the fold-vertex situated on a reflex corner of the paper.

References

1. M. Bern and B. Hayes. The complexity of flat origami. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 175–183, Atlanta, January 1996.
2. T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. Toussaint and S. Whitesides. Locked and unlocked polygonal chains in three dimensions. *Discrete & Computational Geometry*, 26(3):269–281, 2001.
3. J. Cantarella and H. Johnston. Nontrivial embeddings of polygonal intervals and unknots in 3-space. *J. Knot Theory Ramifications*, 7(8):1027–1039, 1998.
4. R. Connelly, E. Demaine and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(5):205–239, 2003.
5. H. Crapo and W. Whiteley. Statics of frameworks and motions of panel structures: a projective geometric introduction. *Structural Topology*, 6:43–82, 1982.
6. E. Demaine and M. Demaine. Recent results in computational origami. In *Proc. 3rd Int. Meeting of Origami Science, Math, and Education (OSME 2001)*, pages 3–16, Monterey, California, March 9–11 2001.
7. E. Demaine, M. Demaine, and J. Mitchell. Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. In *Proc. 15th ACM Symp. on Computational Geometry (SoCG'99)*, pages 105–114, Miami Beach, Florida, June 13–16 1999.
8. J. Graver, B. Servatius and H. Servatius. *Combinatorial Rigidity.* Graduate Studies in Mathematics vol. 2. American Mathematical Society, 1993.
9. D. Huffman. Curvature and creases: a primer on paper. *IEEE Transactions on Computers*, C-25(10):1010–1019, Oct. 1976.
10. T. Hull. Rigid origami. http://www.merrimack.edu/~thull/rigid/rigid.html, 2003.
11. M. Kapovich and J. Millson. Hodge theory and the art of paper-folding. *Publications of RIMS, Kyoto*, 33(1):1–33, 1997.
12. M. Kapovich and J. Millson. On the moduli space of a spherical polygonal linkage. *Canad. Math. Bull.*, 42(3):307–320, 1999.
13. R. Lang. A computational algorithm for origami design. In *Proc. 12th ACM Symposium on Computational Geometry*, pages 98–105, 1996.
14. K. Miura. A note on intrinsic geometry of origami. *Proc. First International Meeting of Origami Science and Technology (H. Huzita ed.)*, pages 239–249, 1989.

15. G. Rote, F. Santos and I. Streinu. Expansive motions and the polytope of pointed pseudo-triangulations. In J. Pach, B. Aronov, S. Basu and M. Sharir, editors, *Discrete and Computational Geometry - The Goodman-Pollack Festschrift*, Algorithms and Combinatorics, pages 699–736. Springer Verlag, Berlin, 2003.
16. F. Saliola and W. Whiteley. Notes on the equivalence of first-order rigidity in various geometries. Preprint, 2002.
17. I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proc. ACM-IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 443–453, 2000.
18. T.S. Tay and W. Whiteley. Generating isostatic graphs. *Structural Topology*, 11:21–68, 1985.
19. W. Whiteley. Cones, infinity and one-story buildings. *Structural Topology*, 8:53–70, 1983.
20. W. Whiteley. Rigidity and scene analysis. In J.E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 893–916. CRC Press, Boca Raton New York, first edition, 1997.

An Optimal Algorithm for the 1-Searchability of Polygonal Rooms

Xuehou Tan

Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@wing.ncc.u-tokai.ac.jp

Abstract. The *1-searcher* is a mobile guard who can see only along a ray emanating from his position and can continuously change the direction of the ray with bounded speed. A polygonal region P with a specified point d on its boundary is called a *room*, and denoted by (P, d). The room (P, d) is said to be *1-searchable* if the searcher, starting at the point d, can eventually see a mobile intruder who moves arbitrarily fast inside P, without allowing the intruder to touch d. We present an optimal $O(n)$ time algorithm to determine whether there is a point x on the boundary of P such that the room (P, x) is 1-searchable. This improves upon the previous $O(n \log n)$ time bound, which was established for determining whether or not a room (P, d) is 1-searchable, where d is a given point on the boundary of P.

1 Introduction

Recently, much attention has been devoted to the problem of searching for a mobile intruder in a polygonal region P by a mobile searcher [6,8,9,10,11,12,13,14,15]. Both the searcher and the intruder are modeled by points that can continuously move in P. The *1-searcher* is a mobile guard who can see only along a ray emanating from his position and can change the direction of the ray with bounded speed. A polygonal region P with a specified point d (called the *door*) on its boundary is called a *room*, and denoted by (P, d). The room (P, d) is said to be *1-searchable* if the searcher, starting at the point d, can eventually see a mobile intruder who moves arbitrarily fast inside P, without allowing the intruder to touch d.

The problem of searching a polygonal room by a single 1-searcher was first studied by Lee et al. [10]. By characterizing the class of 1-searchable rooms, they described an $O(n \log n)$ time algorithm to determine if a specifed room is 1-searchable. An optimal algorithm for generating a search schedule was later given in [14]. In this paper, we present an optimal $O(n)$ time and space algorithm to determine whether there is a point x on the boundary of P such that the room (P, x) is 1-searchable. Combining with result of [14], we thus obtain an optimal solution to the problem of searching a polygonal room by a 1-searcher. Moreover, our algorithm is simple and does not require a triangulation of P. This simplicity is important as many linear-time geometric algorithms depend on the triangulation algorithm of Chazelle [3], which is too complicated to be suitable in practice.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 174–183, 2005.

2 Preliminary

Let P denote a simple polygon, i.e., it has neither self-intersections nor holes. Two points $x, y \in P$ are said to be mutually *visible* if the line segment connecting them, denoted by $\overline{xy}$, is entirely contained in P. For two regions $Q_1, Q_2 \subseteq P$, we say that Q_1 is *weakly visible* from Q_2 if every point in Q_1 is visible from some point in Q_2. For a vertex x of the polygon P, let $Succ(x)$ denote the vertex immediately succeeding x clockwise, and $Pred(x)$ the vertex immediately preceding x clockwise. A vertex of P is *reflex* if its interior angle is strictly greater than 180°; otherwise, it is *convex*. An important definition for reflex vertices is that of *ray shots*: the backward ray shot from a reflex vertex r, denoted by *Backw(r)*, is the first point of P hit by a "bullet" shot at r in the direction from $Succ(r)$ to r, and the forward ray shot *Forw(r)* is the first point hit by the bullet shot at r in the direction from $Pred(r)$ to r. See Fig. 1.

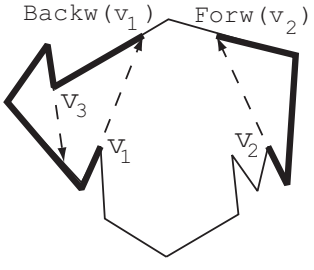


Fig. 1. Forward, backward ray shots and components

Let u, v denote two boundary points of P, and let $P[u, v]$ (resp. $P(u, v)$) denote the closed (resp. open) *clockwise* chain of P from u to v. We define the chain $P[r, Backw(r)]$ (resp. $P[Forw(r), r]$) as the *backward component* (resp. *forward component*) of the reflex vertex r. The point r is referred to as the *defining vertex* of the component. See Fig. 1 for an example, where two different components of v_1 and v_2 are shown in bold line. A backward (resp. forward) component is said to be *non-redundant* if it does not contain any other backward (resp. forward) component. A reflex vertex is *critical* if its backward or forward component is non-redundant. For example, the vertices v_1, v_2 and v_3 in Fig. 1 are critical.

A polygon P is said to be *LR-visible* if there is a pair of boundary points u and v such that $P[u, v]$ and $P[v, u]$ are weakly visible from each other. Clearly, P is *LR*-visible with respect to the point pair (u, v) if and only if each non-redundant component of P contains either u or v. Das et al. have developed a linear-time algorithm to determine whether a polygon P is *LR*-visible or not [4]. Later, Bhattacharya and Ghosh [1] simplified the algorithm such that it uses only simple data structures and does not require a triangulation of the polygon. The algorithm also allows one to compute the shortest paths from an arbitrary vertex to all other vertices of P. If P is *LR*-visible, then all of its

non-redundant components can be computed in linear time [1,4]. (Actually, the containment relation between forward components and backward components is further considered in the definition of non-redundant components given by Das et al. [4]. But, the main part of their algorithm is to compute the set of non-redundant forward or backward components.)

Lemma 1. *[1,4] It takes $O(n)$ time to determine whether or not P is LR-visible. Also, all non-redundant forward (resp. backward) components of an LR-visible polygon can be computed in $O(n)$ time.*

A pair of reflex vertices x, y is said to give a *d-deadlock*, where d is a boundary point of P, if both components $P(x, Backw(x)]$ and $P[Forw(y), y)$ do not contain d, and the points v_1, $Forw(v_2)$, $Backw(v_1)$ and v_2 are in clockwise order. (Note that the point d may be identical to x or y.) See an example in Fig. 2(a). In the case that P is LR-visible with respect to some point pairs (x, y), all the x-deadlocks and y-deadlocks in P can be reported in linear time.

Lemma 2. *[2] Suppose that P is LR-visible with respect to some point pairs (x, y). It takes $O(n)$ time to report all the x-deadlocks and y-deadlocks in P.*[1]

3 The Main Result

The characterization of 1-searchable rooms was originally given by Lee et al. [10]. To obtain the optimality of the algorithm, we make use of the following alternate characterization, which is given in terms of components and deadlocks (see also [14]).

Lemma 3. *[10,14] A polygonal room (P, d) is not 1-searchable if and only if one of the following conditions is true.*

*(**A1**) A d-deadlock occurs (Fig. 2(a)), or there are two disjoint components such that both of them do not contain d (Figs. 2(b)-(e)).*

*(**A2**) There are three reflex vertices v_1, v_2 and v_3, which are in clockwise order, such that the pair (v_1, v_3) gives both the v_2-deadlock and the $Forw(v_2)$-deadlock or $Backw(v_2)$-deadlock (Fig. 2(f)).*

*(**A3**) There are two vertices a_2 and b_2 such that both components $P[a_2, Backw(a_2)]$ and $P[Forw(b_2), b_2]$ do not contain d, and all vertices of the chain $P[a_2, b_2]$ have their deadlocks (Fig. 2(g)).*

Notice first that the condition **A2** is independent of d, which implies that if **A2** is true, then P is not 1-searchable for any room (P, d), where d is an arbitrary point on the boundary of P. Actually, if **A2** is true, then the condition **A1** is true for all the rooms (P, x), $x \in P[Forw(v_1), Backw(v_3)]$. Note also that

[1] This result, together with Lemma 1, gives an optimal algorithm for the *two-guard walkability* of simple polygons [2]. In the appendix, we give a polygon that has a 1-searchable room, but is not walkable by two guards [7]. Thus, our result is stronger than the result obtained in [2].

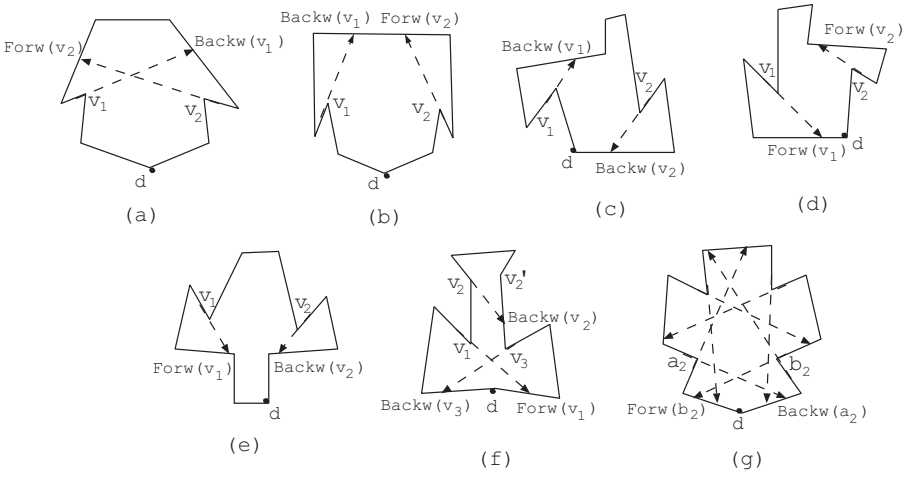


Fig. 2. The conditions A1, A2 and A3

if P is not LR-visible, then **A1** is true for every point d on the boundary of P, and thus no rooms in P are 1-searchable.

We will present an $O(n)$ time algorithm to determine whether there is a 1-searchable room in a simple polygon. Our algorithm is based on the following observations, which immediately follow from the definition of critical vertices.

Observation 1. *If there are two disjoint components such that* **A1** *is true, then we can assume that these two components are non-redundant, or equally, two defining vertices of these components are critical.*

Observation 2. *If* **A2** *is true, then we can assume that the vertex v_2 for* **A2** *is critical.*

Observation 3. *If* **A3** *is true, then we can assume that two vertices a_2 and b_2 for* **A3** *are critical.*

For simplicity, we consider below the ray shot from a critical vertex as two different vertices of P; one slightly preceding it and one slightly succeeding it. Following from Lemma 3 and the observations made above, it suffices to verify **A1**, **A2** and **A3** for all *vertex-door rooms* (P, d), where d denotes a vertex of P. Our algorithm can be summarized as follows.

Algorithm *searchability*

1. Run the linear-time algorithm of [1,4] to determine whether the given polygon P is LR-visible. If P is not LR-visible, report "no rooms in P are 1-searchable". (It means that no room (P, d), where d is an arbitrary point on the boundary of P, is 1-searchable.) Otherwise, compute all non-redundant components (i.e., critical vertices) of P, and then mark the ray shots from critical vertices as the vertices of P.

2. Verify the condition **A1** for all vertex-door rooms of P. If **A1** is true for all vertex-door rooms, report "no rooms in P are 1-searchable".
3. Check whether the condition **A2** is true or not. If *yes*, report "no rooms in P are 1-searchable".
4. For the vertex-door rooms (P, d) for which the condition **A1** is not true, we further verify whether the condition **A3** is true for them. If **A1** or **A3** holds for every vertex-door room, report "no rooms in P are 1-searchable". Otherwise, a 1-searchable room exists and we report it.

Theorem 1. *The algorithm* **searchability** *takes $O(n)$ time to determine whether there is a point x on the boundary of P such that the room (P, x) is 1-searchable.*

Proof. First, run the linear-time algorithm of Das et al. [1,4] to check if the polygon P is LR-visible. If P is not LR-visible, report "no rooms in P are 1-searchable", and we are done. Otherwise, all non-redundant components as well as their corresponding ray shots are computed. An order of the polygon vertices, including the ray shots from critical vertices, on the boundary of P is then obtained.

The step 2 of the algorithm *searchability* is to check if **A1** is true for every vertex-door room (P, d). The condition **A1** for (P, d), except for the d-deadlock case, can be verified as follows. Let v_1 denote the critical vertex of P such that it is closest to d counterclockwise and the component $P[v_1, Backw(v_1)]$ does not contain d, and v_2 the critical vertex such that it is closest to d clockwise and the component $P[Forw(v_2), v_2]$ does not contain d. If the points v_1, $Backw(v_1)$, $Forw(v_2)$ and v_2 are in clockwise order, the configuration shown in Fig. 2(b) occurs, and thus **A1** is true for (P, d). Otherwise, the configuration shown in Fig. 2(b) never occurs for (P, d). This is because $P[Backw(v_1), Forw(v_2)]$ contains all chains $P[Backw(v_1'), Forw(v_2')]$, where v_1', v_2' are critical and the points v_1', $Backw(v_1')$, $Forw(v_2')$ and v_2' are in clockwise order. For each vertex d, the corresponding vertices v_1 and v_2 as well as the order of v_1, $Backw(v_1)$, $Forw(v_2)$ and v_2 can be found in (amortized) constant time. Thus, we can determine in $O(1)$ amortized time if the configuration shown in Fig. 2(b) occurs. Other situations shown in Figs. 2(c)-2(e) can be dealt with analogously.

Consider now the deadlock case for the condition **A1**. Suppose that there are no two disjoint components in P which make the condition **A1** be true for (P, d), but there are two vertices u_1 and u_2 which give the d-deadlock. (Note that u_1 or u_2 may not be critical.) Then, $P(Backw(u_1), d]$ (resp. $P[d, Forw(u_2)))$ does not contain any other component; otherwise, the defining vertex of the contained component and u_1 (resp. u_2) give some configuration of **A1** shown in Figs. 2(b)-2(e), a contradiction. For the same reason, there are no two disjoint components in $P[u_1, u_2]$. Hence, there is at least one point $d' \in P[Forw(u_2), Backw(u_1)]$ such that P is LR-visible with respect to the point pair (d, d'). We can then use Bhattacharya et al.'s algorithm [2] to determine if a d-deadlock occurs. It follows from Lemma 2 that all the v-deadlocks can be reported in $O(n)$ time, provided that the configurations of **A1** shown in Figs. 2(b)-2(e) do not occur for the rooms (P, v).

Turn to the step 3 of the algorithm *searchability*. Suppose that (P, d) is a vertex-door room, for which **A1** is not true. Let v_2 be the critical vertex such that it is closest to d counetrclockwise and the component $P[v_2, Backw(v_2)]$ does not contain d (if it exists). Let P' denote the portion of P obtained by cutting off the region bounded by $P[v_2, Backw(v_2)]$ and the line segment $\overline{v_2 Backw(v_2)}$. None of the configurations shown in Figs. 2(b)-2(e) occurs for two rooms (P', v_2) and $(P', Backw(v_2))$ simultaneously; otherwise, there are three disjoint components in P and thus P is not LR-visible [4], a contradiction. As discussed above, we can then determine in $O(n)$ time if there is a v_2-deadlock or a $Backw(v_2)$-deadlock in the polygon P'. If *yes*, two vertices giving the deadlock and v_2 make the condition **A2** be true, and thus no rooms in P are 1-searchable. Otherwise, we further find the critical vertex v_2' such that it is closest to d clockwise and the component $P[Forw(v_2'), v_2']$ does not contain d, and perform the same procedure for v_2' (if it exists). If **A2** is not ever satisfied, it can never be true for the polygon P, as we have assumed that the condition **A1** is not true for the room (P, d).

Finally, consider the step 4 of *searchability*. Again, let (P, d) denote a vertex-door room, for which **A1** is not true. Let $l_1, \ldots, l_i$ be the sequence of critical vertices on P such that l_1 is closest to d counterclockwise and all the components $P[l_k, Backw(l_k)]$ $(1 \le k \le i)$ do not contain d. The points $Backw(l_1)$, $Backw(l_2)$, $\ldots$, $Backw(l_i)$ are then in clockwise order. See Fig. 3. Similarly, let $r_1, \ldots, r_j$ be the sequence of critical vertices on the boundary of P such that r_j is closest to d clockwise and all the components $P[Forw(r_k), r_k]$ $(1 \le k \le j)$ do not contain d. Also, the points $Forw(r_1)$, $Forw(r_2)$, $\ldots$, $Forw(r_j)$ are in clockwise order. Assume that both l_i and r_1 exist (otherwise, the room (P, d) is 1-searchable and we are done), and that the points d, l_i and r_1 are in clockwise order (otherwise, the d-deadlock occurs, a contradiction). To verify the condition **A3** for (P, d), we first determine if P is LR-visible with respect to both point pairs (d, l_i) and (d, r_1) [1,4]. If *yes*, then P is LR-visible with respect to any point pair (d, d'), $d' \in P[l_i, r_1]$. So we can verify whether all vertices of $P[l_i, r_1]$ have their deadlocks (Lemma 2). If there is a vertex in $P[l_i, r_1]$ that does not have the deadlock, then the room (P, d) is 1-searchable and we are done. Otherwise, **A3** is true for (P, d) as well as the rooms (P, v), $v \in P(Backw(l_i), Forw(r_1))$.

Suppose that **A3** is true for the rooms (P, v), $v \in P(Backw(l_i), Forw(r_1))$. We need to further check whether the condition **A3** is true for the vertex-door

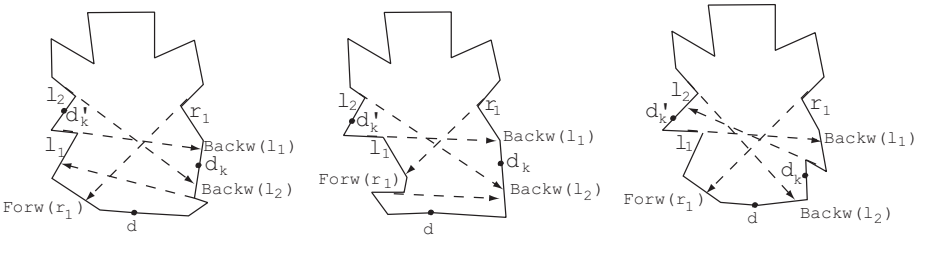


Fig. 3. The polygon P is LR-visible with respect to both point pairs (d, l_i) and (d, r_1)

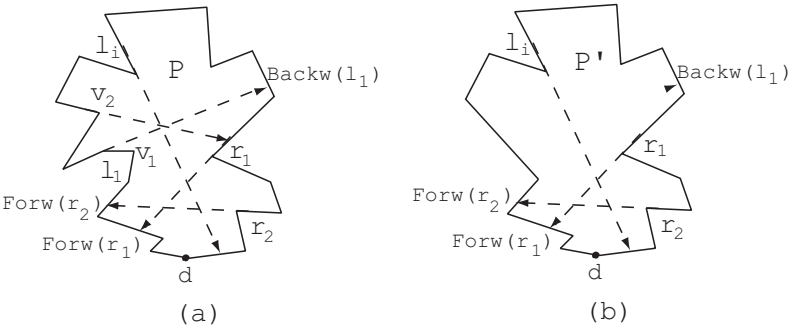


Fig. 4. The polygon P is LR-visible only with respect to the point pair (d, l_i)

rooms (P, d), $d \in P[Backw(l_1), Backw(l_i)] \cup P[Forw(r_1), Forw(r_j)]$. Since the condition **A1** has previously been verified, by a scan of the polygon boundary, we can find all the vertex-door rooms (P, d_k), $d_k \in P[Backw(l_{i-k}), Backw(l_{i-k-1})]$ and $0 \leq k \leq i-2$, for which **A1** is not true. Assume that **A1** is not true for a room (P, d_k), $d_k \in P[Backw(l_{i-k}), Backw(l_{i-k-1})]$, and d_k is contained in $P[r_1, d]$ (it can easily be verified, too). In this case, two chains $P[d_k, d'_k]$ and $P[d'_k, d_k]$, for any point $d'_k \in P[l_{i-k-1}, l_{i-k}]$, are mutually weakly visible; otherwise, **A1** is true for (P, d_k) or some vertices of $l_1, \ldots, l_i$ are not critical, a contradiction in either case. See Fig. 3 for some examples, where the vertex l_{i-k-1} and the vertex destroying the weak visibility (the component of that vertex does not contain d_k nor d'_k) make **A1** be true for (P, d_k). Thus, we can determine if all vertices of $P[l_{i-k-1}, l_{i-k}]$ have their deadlocks (Lemma 2), and if so the condition **A3** is true for the room (P, d_k). If **A3** is not true for some room (P, d_k), then it is 1-searchable and we are done. Otherwise, we perform a symmetric procedure for the sequence of vertices $r_1, \ldots, r_j$. In this way, we can determine in $O(n)$ time if there is a 1-searchable room in P, and if so report such a room. (Note that the algorithm of Bhattacharya et al. [2] needs to run only once for the polygon P, although its outputs (i.e., the deadlocks reported) are used several times in our algorithm.)

Let us turn to the situation in which the polygon P is LR-visible with respect to only one point pair, say, (d, l_i). In this case, r_1 (as well as d) is not contained in the component $P[l_1, Bcakw(l_1)]$. See Fig. 4(a). Following from the discussion made above, the work of verifying the condition **A3** is to compute the deadlocks for the vertices of $P[l_1, r_j]$. Since P is LR-visible with respect to both point pairs (d, l_i) and $(d, Backw(l_1))$ in this case, we can simply determine if all the vertices of $P[l_1, Backw(l_1)]$ have their deadlocks (Lemma 2). But, a new method for reporting the vertices of $P[Backw(l_1), r_j]$ having their deadlocks has to be developed. Let v_1 and v_2 denote two vertices such that their backward components $(P[v_l, Backw(v_l)]$, $l = 1, 2)$ do not contain r_1 and all such vertices are contained in $P[v_1, v_2]$. See Fig. 4(a). Clearly, $P[v_1, v_2] \subset P[d, l_i]$ holds. For any vertex $v \in P[v_1, v_2]$, no backward shot $Backw(v)$ can contribute to an x-deadlock, $x \in P[Backw(l_1), r_j]$; otherwise, the d-deadlock occurs, a contradiction. However, the shot $Forw(v)$ may contribute to an x-deadlock, $x \in P[Backw(l_1), r_j]$.

Let v' denote the vertex such that two shots $Backw(v')$ and $Forw(v)$ give the x-deadlock. Then, the vertex v' is contained in any component $P(Backw(v''), v'')$, $v'' \in P(v, v_2]$; otherwise, three vertices v, v' and v'' make the condition **A2** be true, a contradiction. This implies that the vertex l_i is contained in $P[v, v']$. Since the polygon P is weakly visible with respect to the point pair (d, l_i), these x-deadlocks with one defining vertex belonging to $P[v_1, v_2]$ can thus be found using Lemma 2. Clearly, when we compute other deadlocks, all vertices of $P[v_1, v_2]$ can be ignored. Note that the vertices v_1 and v_2 can be found by computing the shortest paths from r_1 to all vertices of $P[d, l_i]$ [5], and marking the vertices v such that the shortest path from r_1 to $Succ(v)$ turns left at v (as viewed from r_1). Let P' denote the polygon obtained after the chain $P[v_1, v_2]$ is deleted (i.e., connnecting $Pred(v_1)$ and $Succ(v_2)$ by a line segment). See Fig. 4(b) for an example. The polygon P' is now LR-visible with respect to both point pairs (d, r_j) and (d, l_i) (or $(d, Backw(l_1)$ if l_i is deleted). As discussed above, we can find the vertices of $P'[Backw(l_1), d]$, which have their deadlocks in the polygon P'. Since any pair of reflex vertices giving a deadlock in P' corresponds to a unique pair of reflex vertices of P, the same deadlock also occurs in P. In conclusion, we can determine in $O(n)$ time whether there is a 1-searchable room in P.

The situation in which the polygon P is LR-visible with respect to only the point pair (d, r_1) can be dealt with analogously. Note that the polygon P is LR-visible with respect to at least one pair of $(d, l_i))$ and (d, r_1); otherwise, the condition **A1** is true for (P, d), contradicting our assumption. This completes the proof. □

4 Conclusion

We have proposed an optimal $O(n)$ time algorithm to determine whether there is a point d on the boundary of P such that the room (P, d) is 1-searchable. Our result improves upon the previous $O(n \log n)$ time bound, which was established for determining whether a specified room is 1-searchable. A further work is to give a linear time algorithm to determine whether a simple polygon is 1-searchable, without considering any door [8,13,15].

Acknowledgements

This research is partially supported by the Grant-in-Aid of the Ministry of Education, Science, Sports and Culture of Japan.

References

1. B.K.Bhattacharya and S.K Ghosh, Characterizing LR-visibility polygons and related problems, *Comput. Geom. The. Appl.* **18** (2001) 19-36.
2. B.K.Bhattacharya, A. Mukhopadhyay and G.Narasimhan, Optimal algorithms for two-guard walkability of simple polygons, *Lect. Notes Comput. Sci.* **2125** (2001) 438-449.

3. B.Chazelle, Triangulating a simple polygon in linear time, *Discrte Comput. Geometry* **6** (1991) 485-524.
4. G.Das, P.J.Heffernan and G.Narasimhan, LR-visibility in polygons, *Comput. Geom. Theory Appl.* **7** (1997) 37-57.
5. L.J.Guibas, J.Hershberger, D.Leven, M.Sharir and R.E.Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica*, **2** (1987) 209-233.
6. L.J.Guibas, J.C.Latombe, S.M.Lavalle, D.Lin and R.Motwani, Visibility-based pursuit-evasion in a polygonal environment, *Int. J. Comput. Geom. & Appl.* **9**, (1999) 471-493.
7. C. Icking and R. Klein, The two guards problem, *Int. J. Comput. Geom. & Appl.* **2** (1992) 257-285.
8. S.M.LaValle, B.Simov and G.Slutzki, An algorithm for searching a polygonal region with a flashlight, in *Int. J. Comput. Geom. & Appl.* **12** (2002) 87-113.
9. J.H. Lee, S.Y.Shin and K.Y.Chwa, Visibility-based pursuit-evasion in a polygonal room with a door, *Proc. 15th Annu. ACM Symp. Comput. Geom.* (1999) 281-290.
10. J.H.Lee, S.M.Park and K.Y.Chwa, Searching a polygonal room with one door by a 1-searcher, *Int. J. Comput. Geom. & Appl.* **10** (2000) 201-220.
11. J.H.Lee, S.M.Park and K.Y.Chwa, Simple algorithms for searchng a polygon with flashlights, *Inform. Process. Lett.* **81** (2002) 265-270.
12. I.Suzuki and M.Yamashita, Searching for mobile intruders in a polygonal region, *SIAM J. Comp.* **21** (1992) 863-888.
13. I.Suzuki, Y.Tazoe, M.Yamashita and T.Kameda, Searching a polygonal region from the boundary, *Int. J. Comput. Geom. & Appl.* **11** (2001) 529-553.
14. X.Tan, Efficient algorithms for searching a polygonal room with a door, *Lect. Notes Comput. Sci.* **2098** (2001) 339-350.
15. X.Tan, A characterization of polygonal regions searchable from the boundary, *Lect. Notes Comput. Sci.* **3330** (*Proc. of IJCCGGT 2003*) 200-215.

Appendix

A simple polygon P with two marked points s, t on its boundary is called a *corridor*, and denoted by (P, s, t). The 1-searcher is termed as *two guards* [7], if we require that the movement of the endpoint of the ray (as well as the 1-searcher) be continuous on the polygon boundary. The corridor (P, s, t) is said to be *walkable* by two guards if two guards starting at s can force the mobile intruder out of P through t, without allowing the intruder to touch s [7]. It has been shown that (P, s, t) is walkable by two guards if and only if $P[s, t]$ and $P[t, s]$ are weakly visible from each other and neither s-deadlocks nor t-deadlocks occur [7]. An $O(n)$ time algorithm has been given to determine if there is a point pair (s, t) on the boundary of P such that (P, s, t) is walkable by two guards [2].

It is clear that if the polygon P is walkable by two guards, then there is a 1-searchable room in P. However, the converse is not true. The room (P, d) shown in Fig. 5 is 1-searchable, but P is not walkable by two guards. This is because all points of $P[a, b]$ have their deadlocks, and any two chains $P[u, v]$ and $P[v, u]$, $u, v \in P[b, a)$, are not weakly visible from each other.

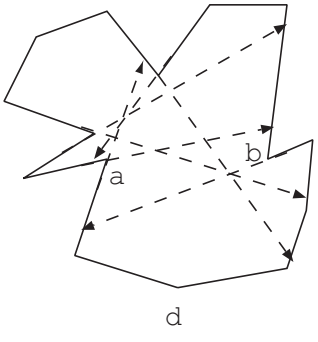


Fig. 5. A polygon has a 1-searchable room, but it is not walkable by two guards

Crossing Stars in Topological Graphs

Gábor Tardos and Géza Tóth

Rényi Institute, Hungarian Academy of Sciences, Budapest, Hungary,
{tardos, geza}@renyi.hu

Abstract. Let G be a graph without loops or multiple edges drawn in the plane. It is shown that, for any k, if G has at least $C_k n$ edges and n vertices, then it contains three sets of k edges, such that every edge in any of the sets crosses all edges in the other two sets. Furthermore, two of the three sets can be chosen such that all k edges in the set have a common vertex.

1 Introduction

A *topological graph* is a graph drawn in the plane with no loops or multiple edges so that its vertices are represented by points, and its edges by Jordan curves connecting the corresponding points. We do not distinguish these points and curves of the topological graph from the vertices and edges of the underlying abstract graph they represent. We assume that (i) the edges of a topological graph do not pass through any vertex, (ii) two edges share a finite number of interior points and they properly cross at each and (iii) no three edges cross at the same point. A topological graph is called *simple* if any pair of its edges have at most one point in common (either a common endpoint or a crossing).

It is well known that every planar graph with n vertices has at most $3n-6$ edges. Equivalently, every topological graph G with n vertices and more than $3n-6$ edges has a pair of crossing edges. This simple statement was generalized in several directions.

Pach et al. [8], [7] proved that a topological graph of n vertices and more than $(r+2)(n-2)$ edges must have r edges that cross the same edge. This bound is tight for $r = 1, 2, 3$, but can be substantially improved for large values of r.

Agarwal et al. [1] (for simple topological graphs) and then, with a shorter and more general argument, Pach et al. [3] proved that for some $c > 0$, every topological graph with n vertices and more than cn edges has *three* pairwise crossing edges. In [4], this result was further strengthened: for every integer $r > 0$, there exists a constant $c_r > 0$, such that every topological graph with n vertices and more than $c_r n$ edges has $r + 2$ edges such that the first two cross each other and both of them cross the remaining r edges (see Fig. 1a).

In [5] another generalization was shown. For any k and l there is a constant $c_{k,l}$ with the following property. Every topological graph with n vertices and more than $c_{k,l} n$ edges has $k + l$ edges such that the first k have a common vertex, and each of them crosses all of the remaining l edges (see Fig. 1b).

In this note we prove a common generalization of the above results.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 184–197, 2005.

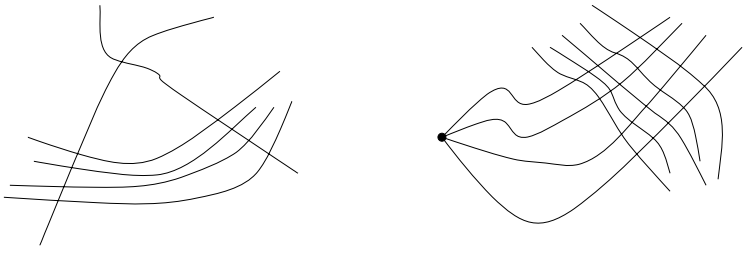


Fig. 1. A topological graph without either configuration has only a linear number of edges

Let k be a positive integer. The edges $A \cup B \cup Z$ of a topological graph form a *k-star grid* if A is a set of k edges incident to a common endpoint x, B is a set of k edges incident to a common endpoint y and any edge from A crosses any edge from B; furthermore, Z also contains k edges and any edge in Z crosses all edges in $A \cup B$. See Figure 2. In this definition we allow for the case $x = y$ and we also allow the edges of Z to be incident to x or y. These pathological cases are not possible in a simple topological graph.

Theorem 1. *For any $k \geq 1$, there is a constant C_k such that every topological graph with n vertices and at least $C_k n$ edges contains a k-star grid.*

If we could guarantee any further edge-crossing in a k-star grid, then we would have *four* pairwise crossing edges. Therefore, k-star grids seem to represent the natural last configuration before one attacks the following well-established conjecture:

Conjecture 1. There is a $C > 0$ such that every topological graph with n vertices and Cn edges contains four pairwise crossing edges.

2 Proof of the Theorem

The proof of Theorem 1 is rather technical and consists of several steps. First we give an overview and indicate which steps of the proofs can be eliminated if we only consider simple topological graphs. Note that we do not strive for absolute preciseness in this overview. The reader finds the precise definitions later in the proof.

Fix k and take an arbitrary topological graph F. We let $C = |E(F)|/|V(F)|$. Our goal is to prove that if C is large enough (as a function of k), then we find a k-star grid in F. This clearly establishes Theorem 1.

First we take a *densest subgraph* F_0 of F and concentrate on F_0 only.

Next we *redraw* F_0, i. e., we take another topological graph G_0 which has the same underlying abstract graph as F_0 but eliminates certain unnecessary

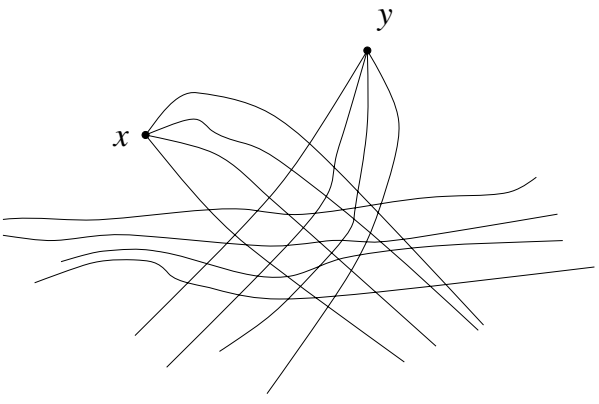


Fig. 2. A 4-star grid

crossings. This step of the proof is not needed if F is a simple topological graph, i. e., we may take $G_0 = F_0$.

We then use *subdivisions*, i. e., we introduce vertices at certain edge-crossings. We obtain a subdivision G_1 of G_0 with a crossing-free spanning tree T. This step is taken from [3] and [5].

We further subdivide G_1 to obtain G_2 and its crossing-free spanning subgraph H with no *proper cut*. This means that any two consecutive crossing points of any edge e in $G_2 \setminus H$ with H are with "close-by" edges of H. This step is taken from [5]. In this and the previous step we make sure that the size (number of vertices) of the graph increases by a constant factor only. Note also that subdivisions in these two steps can create k-star grids. This does not happen for simple topological graphs.

The next step represents the new idea in this paper. For many vertices we find a large number of edges emanating from that vertex with the property that they go "parallel" (with respect to H) for a long time and then one by one they "depart" from the rest of the edges. All these "departures" take place in separate cells of H. We call these sets of edges *bundles*.

Using that C is large enough we find a *cross-track configuration* in G_2, i. e., k edges of a bundle, another k edges of a (perhaps different) bundle such that these $2k$ edges go parallel through $l - 1$ cells of H but still, eventually the first k edges cross the second k edges. Here l is an exponential function of k. Note that for simple topological graphs we can choose $l = k$ and the proof ends here. Indeed, the $2k$ edges in the cross-track configuration plus k edges of H form a k-star grid. In the general case however, some of the edges of H crossed by the edges in the cross-track configuration may coincide or may be parts of the same edge of G_0 separated only by our subdivision process.

The final step of the proof takes care of the technical difficulties mentioned above. We use a result of Schaefer and Stefankovič [9] to show that if l is large enough, then out of the l edges of H crossed by the parallel track of the edges of the cross-track configuration at least k must come from distinct edges of G_0.

We continue with the detailed execution of the above plan.

Let $k \geq 1$ fixed, and let F be a topological graph with n' vertices and Cn' edges. Our goal is to prove that F contains a k-star grid if C is large enough. The bound on C may depend on k but not on n'. This will establish the validity of Theorem 1.

Let F_0 be the densest non-empty connected subgraph of F that is, $F_0 \subseteq F$ connected and $|E(F_0)|/|V(F_0)|$ is maximal. Clearly, the requirement that F_0 has to be connected does not change the value of the maximum, so we have $|E(F_0)|/|V(F_0)| \geq |E(F)|/|V(F)| = C$. Removing a vertex of F_0 of degree d increases the ratio if $d < C$, therefore each vertex in F_0 has degree at least C. Let n denote the number of vertices of F_0. Clearly, $n > C$ so we may assume $n \geq 5$.

Redraw F_0 so that the resulting topological graph G_0 satisfies the following two conditions:

(i) If two edges of G_0 cross each other, then the corresponding edges also cross in F_0;
(ii) G_0 has the minimum number of crossings among all drawings with property (i).

It is enough to find a k-star grid in G_0 as property (i) shows that the corresponding edges form a k-star grid in F_0 and thus in F too.

We will apply a *subdivision* to G_0, i. e., we declare a certain intersection point of two edges as a *new vertex* and replace each of the two edges by their two segments up to and from that new vertex. Notice that this way we may create two edges connecting the same pair of vertices, thus we have to extend our definition of topological graph to allow for this. No pair of vertices will ever be connected by more than two edges. The graph obtained from G_0 by several subdivisions is called a *subdivision of* G_0. To distinguish from the new vertices of the subdivision, vertices of G_0 are called *old vertices*.

Notice that subdivision does not introduce a k-star grid in a simple topological graph, so if G_0 is simple it is enough to find a k-star grid in a subdivision of G_0. The situation is somewhat more complex if G_0 is not simple. If G_0 contains two k-edge stars A and B such that each edge of A is crossed by each edge of B and another edge e_0 crosses every edge in $A \cup B$ k times, then the repeated subdivision of e_0 may result in a k-star grid.

Obviously, no edge of G_0 intersects itself, otherwise we could reduce the number of crossings by removing the loop. Suppose that G_0 has two distinct edges, e and f, that meet at least twice (including their common endpoints, in the case they have). A simply connected region whose boundary is composed of an arc of e and an arc of f is called a *lens*.

Claim 1. *Every lens in G_0 has a vertex in its interior.*

Proof. Suppose, for a contradiction, that there is a lens ℓ that contains no vertex of G in its interior. Consider a *minimal* lens $\ell' \subseteq \ell$, by containment. Notice that by swapping the two sides of ℓ', we could reduce the number of crossings without

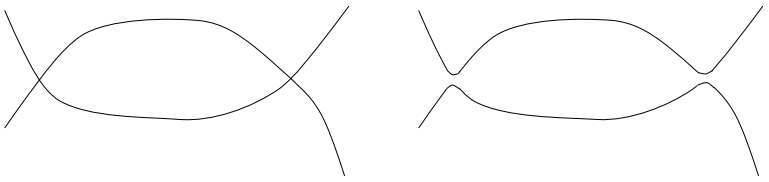


Fig. 3. Swapping the two sides of a lens

creating any new pair of crossing edges, contradicting property (ii) above. See Fig. 3. □

Clearly, subdivisions of G_0 inherit the property of having no self-intersecting edge and the property stated in Claim 1.

Let G be a topological graph, H a subgraph of G. Let e be an edge of G not contained in H. We always consider e with an orientation. Each edge can be considered with either orientation. The edge e has a finite number of intersection points with edges of H, these points split the Jordan curve e into a finite number of shorter curves. We call these shorter curves the *segments* of the edge e determined by H and denote them by $s_1(e), s_2(e), \ldots$ in the order they appear on e. The dependence on H is not explicit in the notation but H will always be clear from the context. If e does not cross the edges of H the entire edge is a single segment.

We consider a crossing-free subgraph H of a topological graph G that is connected and contains all vertices. Such a graph H subdivides the plane into *cells*. The boundary of a cell is a closed walk in H that may visit vertices several times and may even pass through an edge twice. The *size* of a cell is the length of the corresponding walk. A segment s of an edge e not in H inherits its orientation from e. It is contained in single cell α, the endpoints of s are on the boundary of α. See Fig. 4. We call the cell α and the vertex or edge of the boundary walk of α where s starts the *origin* of s. Similarly, α and the vertex or edge of this walk where s ends is the *destination* of s. Notice that in case the boundary of α visits the relevant vertex or edge more than once the origin or destination of e contains more information than the vertex or edge itself, it tells us "which side" of the vertex or edge is involved. If two segments have the same origin and the same destination we call them *parallel* and say that their *type* is the same. If two segments s and s' have the same origin but different destinations, then they are contained in the same cell. We say that s *turns left from* s' if the common origin, the destination of s, and the destination of s' appear in this order in the clockwise tour of the boundary of the cell. Notice that the common origin must differ from either of the destinations. A segment with equal origin and destination would define an "empty lens" contradicting Claim 1. As a consequence, for segments s and s' with a common origin, either s and s' are parallel, or s turns left from s', or s' turns left from s.

As in [3] and [5], first we construct a subdivision G_1 of G_0 that contains a crossing-free spanning tree T.

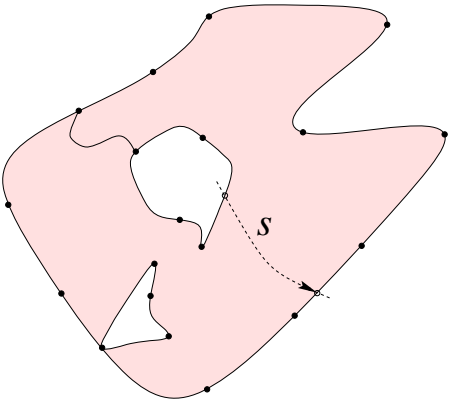


Fig. 4. A cell of H and a segment of an edge

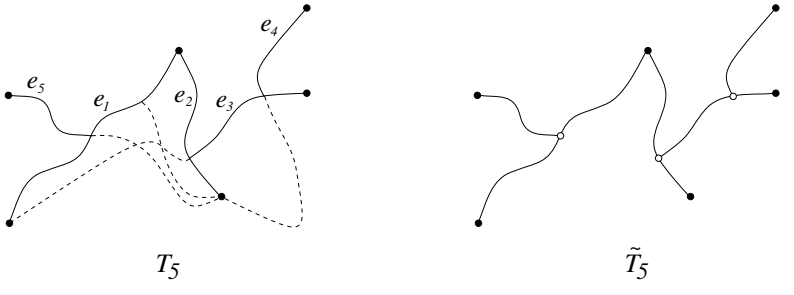


Fig. 5. Constructing $\tilde{T}_5$ from T_5

Since the abstract underlying graph of G_0 is connected, we can choose a sequence of edges $e_1, e_2, \ldots, e_{n-1} \in E(G_0)$ such that $e_1, e_2, \ldots, e_i$ form a tree T_i, for every $1 \leq i \leq n-1$. In particular, $e_1, e_2, \ldots, e_{n-1}$ form a spanning tree T_{n-1} of G.

Construct the crossing-free topological graphs $\tilde{T}_1$, $\tilde{T}_2$, ..., $\tilde{T}_{n-1}$, as follows. Each is a subtree of a subdivision of G_0. Let $\tilde{T}_1$ be defined as a topological graph of two vertices, consisting of the single edge e_1. Suppose that $\tilde{T}_i$ has already been defined for some $1 \leq i < n-1$, and let v denote the endpoint of e_{i+1} that does not belong to T_i. Then we define $\tilde{T}_{i+1}$ as follows. Add to $\tilde{T}_i$ the piece of e_{i+1} between v and its first crossing with $\tilde{T}_i$. More precisely, follow the edge e_{i+1} from v up to the point v' where it hits $\tilde{T}_i$ for the first time. If this is a vertex of $\tilde{T}_i$ simply add e_{i+1} to $\tilde{T}_i$ to get $\tilde{T}_{i+1}$. If v' is in the interior of an edge e then we apply subdivision: we introduce v' as a new vertex. We replace the edge e of $\tilde{T}_i$ with the two resulting parts and add the segment of e_{i+1} between v and v' to obtain $\tilde{T}_{i+1}$.

We let $T = \tilde{T}_{n-1}$ and G_1 be the subdivision of G_0 obtained in the process. Note that G_1 has n old and at most $n-2$ new vertices. See Fig. 5.

Next, just like in [5], we further subdivide G_1 to obtain G_2 and a crossing-free subgraph H of G_2.

Start with $H_0 = T$ and $\tilde{G}_0 = G_1$. Define $H_1, \ldots, H_u$ and $\tilde{G}_1, \ldots, \tilde{G}_u$ recursively, maintaining that H_i is a crossing-free subgraph of a subdivision $\tilde{G}_i$ of G_0. Furthermore H_i is connected, it contains all vertices of $\tilde{G}_i$ and all the cells of H_i are of size at least 8. This clearly holds for H_0 and $\tilde{G}_0$ if $n \geq 5$.

Having defined H_i and $\tilde{G}_i$ consider the segments of the edges of $\tilde{G}_i$ as determined by H_i. Let s be such a segment. By *adding s to H_i* we mean constructing a subdivision of $\tilde{G}_i$ by inserting new vertices for the endpoints of s if necessary and defining a subgraph H_i^s of it by adding s to H_i. More precisely, we also have to replace any edge of H_i that contains in its interior an endpoint of s by the two new edges resulting from the subdivision. Notice that s itself is an edge after the subdivision. The resulting graph H_i^s is a crossing-free connected spanning subgraph of the resulting subdivision of $\tilde{G}_i$. The cell of H_i containing s is now subdivided into two cells, the other cells remain intact (but their size may increase). We call s a *proper cut of H_i* if both new cells of H_i^s are of size at least 8. See Fig. 6.

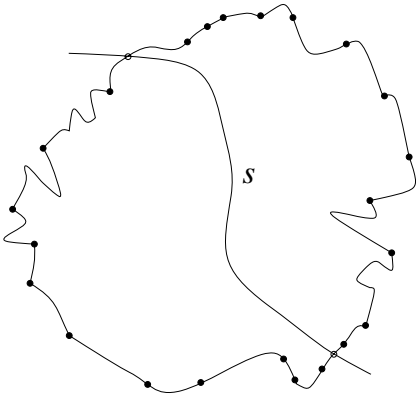


Fig. 6. A proper cut

If there exist a proper cut of H_i, then we choose one such segment s and set $H_{i+1} = H_i^s$ and let $\tilde{G}_{i+1}$ be the resulting subdivision of $\tilde{G}_i$. If there is no proper cut of H_i we set $u = i$, $H = H_u$ and $G_2 = \tilde{G}_u$.

The number of cells starts at one cell, at $H_0 = T$, and increases by one in every step, so H_i contains $i+1$ cells. Each of these cells is of size at least 8, so we have at least $4i + 4$ edges in H_i. From the Euler formula, the number of vertices v_i of H_i is at least $3i + 5$. As $H_0 = T$ contains at most $2n - 2$ vertices and we introduce at most 2 new vertices in every step, so we also have $v_i \leq 2i + 2n - 2$. The upper and lower bounds on v_i imply $i \leq 2n-7$. Therefore, the above process terminates in $u \leq 2n - 7$ steps. This proves the following

Claim 2. *G_2 is a subdivision of G_0 with at most $6n-16$ vertices. H is a connected, spanning, crossing-free subgraph of G_2 with no proper cut. H has at most $8n-24$ edges.*

We call an old vertex of G_2 *important* if its degree in H is less than 32. By Claim 2 H has less than $n/2$ vertices of degree 32 or more. Out of the n old vertices we must have more than $n/2$ important vertices.

Let $l = 2^{k+1}k^2+1$. Consider an edge e of G_2 not in H. Call any l consecutive segments $s_i(e), \ldots, s_{i+l-1}(e)$ a *track* of e. The *type* of a track is simply the sequence of the types of its l segments. Tracks (of possibly different edges) of the same type are called *parallel*. Consider two edges e and f of G_2 that are not in H. Let $d(e,f)$ be the largest index $i \geq 1$ such that for all $1 \leq j < i$ the segments $s_j(e)$ and $s_j(f)$ exist and are parallel. For example, if e and f start at different vertices or in different cells we have $d(e,f) = 1$.

Notice that for any origin of a segment at most 24 destinations are possible. For large cells of H more choices would be possible, but they yield proper cuts of H which do not exist by Claim 2. By the same claim there are less than $32n$ possible origins and therefore less than $768n$ types of segments. The destination of a segment determines the origin of the next segment, therefore there are less than $32 \cdot 24^l n$ different types of tracks.

Let $m = 300k \cdot 24^l$. We call the sequence $e_1, \ldots, e_{2m}$ of $2m$ edges of G_2 not in H a *bundle* if $l < d(e_1, e_{2m}) < d(e_2, e_{2m}) < \ldots < d(e_{2m-1}, e_{2m})$. Notice that the edges of a bundle start at a common vertex. We say that the bundle *emanates* from this common starting vertex.

Claim 3. *If $C > 31 \cdot 24^{2m+l} + 31$, then there exist an bundle emanating from every important vertex.*

Proof. Consider an important vertex x. Let S_0 be the set of edges of G_2 not in H that start at x. The vertex x has degree at least C in G_0 and it has the same degree in its subdivision G_2. Its degree in H is at most 31, so $|S_0| \geq C - 31$. For $i \geq 1$ we define S_i to be a subset of maximal size of S_{i-1} with $s_i(e)$ existing and having equal type for each $e \in S_i$. The number of possible origins for the type of segment $s_1(e)$ of an edge $e \in S_0$ is the degree of x in H. Since x is important, at most 31 origins and at most 744 types of $s_1(e)$ may exist for $e \in S_0$. Thus, $|S_1| \geq |S_0|/744$. Notice that the type of $s_i(e)$ determines if e ends with the segment $s_i(e)$ and if so, then it determines the ending vertex. So if one of the edges $e \in S_i$ ends with its ith segment, then all do, and they all connect the same pair of vertices. Thus, as long as $|S_i| > 2$, $s_{i+1}(e)$ exists for all $e \in S_i$. Furthermore, the type of $s_i(e)$ determines the origin of $s_{i+1}(e)$. So if $|S_i| > 2$ then $|S_{i+1}| \geq |S_i|/24$.

The finiteness of the entire topological graph G_2 implies that $S_i = \emptyset$ for large enough i. Let $l \leq d_1 < d_2 < \ldots < d_v$ be all the indices $d \geq l$ such that $|S_{d+1}| < |S_d|$. The above calculations yield $|S_{d_1}| \geq 24^{2m}$ and $S_{d_i+1} = S_{d_{i+1}} \geq 24^{2m-i}$ for $i \leq 2m$. We choose e_i to be an arbitrary element of $S_{d_i} \setminus S_{d_i+1}$. We have $d(e_i, e_{2m}) = d_i + 1$ for $i < 2m$. This establishes that $(e_1, \ldots, e_{2m})$ form a bundle. □

Fix a bundle $B^x = \{e_1^x, \dots, e_{2m}^x\}$ from every important vertex x. The existence is given by Claim 3. These will be all the bundles, and in fact all the edges of $G_2 \setminus H$ we consider from now on.

The segments $s_1(e_{2m}^x), s_2(e_{2m}^x) \dots, s_{d^x}(e_{2m}^x)$ for $d^x = d(e_m^x, e_{2m}^x)$ form the *backbone* of the bundle B^x. The tracks of e_{2m}^x contained in the backbone are called the *vertebrae*. We denote the vertebra starting with the segment $s_i(e_{2m}^x)$ by t_i^x. Notice that the vertebrae interleave: the last $l-1$ segments of a vertebra is the first $l-1$ segments of the next vertebra. With a vertebra t_i^x there are $m-1$ parallel tracks: the tracks starting with the segments $s_i(e_{m+1}^x), \dots, s_i(e_{2m-1}^x)$.

Let $e = t_i^x$ and $f = t_j^y$ be two distinct parallel vertebrae. Notice that $i > 1$ and $j > 1$ must hold, since we only consider a single bundle from any (important) vertex. Let e' and f' be the inverse orientation of the "previous" segments $s_{i-1}(e_{2m}^x)$ and $s_{j-1}(e_{2m}^y)$, respectively. Notice that e' and f' have the same origin. We say that $e < f$ if e' turns left from f'. We also say that $e < f$ if t_{i-1}^x and t_{j-1}^y are parallel, and $t_{i-1}^x < t_{j-1}^y$. Notice that the recursive definition is well founded and it defines a linear order among parallel vertebrae. We call a vertebra *extremal* if it is smallest or largest among the vertebrae of its type. If e is a non-extremal vertebra we let e^+ stand for the next larger vertebra of the same type, while e^- stands for the next smaller vertebra. We say that a vertebra e is *special* if it is either extremal or one of e^+ or e^- is the last vertebra in a backbone.

Claim 4. *The number of special vertebrae is at most* $65 \cdot 24^l n$.

Proof. We have at most two extremal vertebrae for every type, that is at most $64 \cdot 24^l n$ extremal vertebrae. We have one last vertebra in every backbone, that is at most n last vertebrae. Each last vertebra makes its two (or less) neighbors special, so the claimed bound holds. □

We define a *cross-track configuration* as two sets of k edges such that every edge from the first set crosses every edge from the second set, and all $2k$ edges go parallel for a long time. More precisely, let A and B both be a set of k edges. We say that $A \cup B$ is a *cross-track configuration* if the following conditions hold.

(i) Every $a \in A$ crosses every $b \in B$.

(ii) Every $a \in A$ is incident to an old vertex x and every $b \in B$ is incident to an old vertex y.

(iii) There is $\alpha, \beta > 0$ such that for every $a \in A$, $b \in B$, and $0 \le i < l-1$, $s_{\alpha+i}(a)$ and $s_{\beta+i}(b)$ exist and are parallel.

Notice that for simple topological graphs a cross-track configuration $A \cup B$ can be appended with the set $Z \subseteq E(H)$ consisting of k of the origins of the segments in the parallel tracks of the edges in $A \cup B$. These edges cross every edge in $A \cup B$, therefore $A \cup B \cup Z$ form a k-star grid. Unfortunately, if G_2 is not simple, then Z may contain fewer than k edges, in extreme situations Z might consist of a single edge (the edges in $A \cup B$ cross this single edge many times). Also, finding a k-star grid in G_2 is not enough in this case.

Our immediate goal is to find a cross-track configuration in G_2, see Claim 6. As explained above this leads immediately to a k-star grid in G_2, and also in G_0 if G_2 is simple. For non-simple topological graphs we will also use the cross-track configuration to find k-star grids in G_0, but the argument is more involved.

The following claim is based on a similar observation in [1].

Claim 5. *Let e and f be two consecutive vertebrae of the bundle B^x, neither special. Then e^+ and f^+ are also consecutive vertebrae of a backbone or there exists a cross-track configuration in G_2. The same holds for e^- and f^-.*

Proof. Assume f follows e in B^x and let $e^+ = t_i^y$. We have to show that $f^+ = t_{i+1}^y$. Suppose that $f^+ = t_j^z$.

Since e is not special, $e^* = s_{i+l}(e_{2m}^y)$ is still in the backbone of B^y. Let f^* be the last segment of f. These two segments have a common origin. We distinguish three cases. See Fig. 7.

Case 1: e^* and f^* are parallel. Then, by the definition of the order of vertebrae t_{i+1}^y must be f^+.

Case 2: f^* turns left from e^*. In this case all edges e_a^x intersect all edges e_b^y for $m < a, b \leq 2m$. This provides a cross-track configuration. See Fig 7 (a).

Case 3: e^* turns left from f^*. Now the edges e_a^y and e_b^z must cross for $m < a, b \leq 2m$, and this also provides a cross-track configuration. See Fig 7 (b).

The proof for e^- and f^- is similar. □

We considered at least $n/2$ bundles. By Claim 4 we have at most $65 \cdot 24^l n$ special vertebrae, so the pigeonhole principle gives the existence of a bundle B^x with at most $130 \cdot 24^l$ special vertebrae. We fix such a bundle B^x and let e_i stand for the ith segment in the backbone of B^x: $e_i = s_i(e_{2m}^x)$ for $1 \leq i \leq d(e_m^x, e_{2m}^x)$. We call e_i a *departure point* if $i = d(e_j^x, e_{2m}^x)$ for some $1 \leq j \leq m$. We look for an interval of the backbone of B^x without special vertebrae but with the most departure points. There are m departure points, so at least $\lfloor m/(130 \cdot 24^l + 1) \rfloor$ of them are in an interval that has no special vertebra. Formally, we have $1 \leq i < j \leq d(e_m^x, e_{2m}^x) - l + 1$, such that none of the vertebrae $t_i^x, t_{i+1}^x, \ldots, t_j^x$ are special, but for some indices $1 \leq i' < j' \leq m$ we have $i + l \leq d(e_{i'}^x, e_{2m}^x) < d(e_{j'}^x, e_{2m}^x) \leq j + l - 1$ and $j' - i' + 1 \geq \lfloor m/(130 \cdot 24^l + 1) \rfloor$.

By Claim 5 we either have a cross-track configuration or the vertebrae $(t_i^x)^+$, $(t_{i+1}^x)^+, \ldots, (t_j^x)^+$ are consecutive tracks of some bundle B^y, while $(t_i^x)^-$, $(t_{i+1}^x)^-$, $\ldots, (t_j^x)^-$ are also consecutive tracks of some bundle B^z. In the latter case for any $i' \leq v \leq j'$ the edge e_v^x crosses all edges e_w^y with $m < w \leq 2m$ or it crosses all edges e_w^z with $m < w \leq 2m$. One of the options must occur with at least $\lfloor m/(260 \cdot 24^l + 2) \rfloor \geq k$ edges. This provides a set A of k edges of the bundle B^x, another set B of k edges of a bundle such that the properties of cross-track configuration are satisfied. Thus, a cross-track configuration must exist. See Figure 8. This proves the following

Claim 6. *For $C > 31 \cdot 24^{2m+l} + 31$, there exists a cross-track configuration in G_2.*

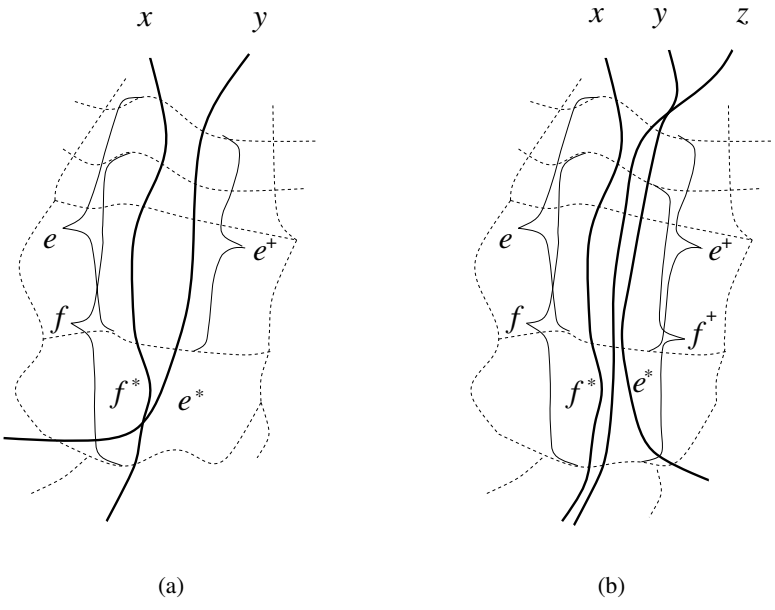


Fig. 7. e^+ and f^+ are consecutive vertebrae

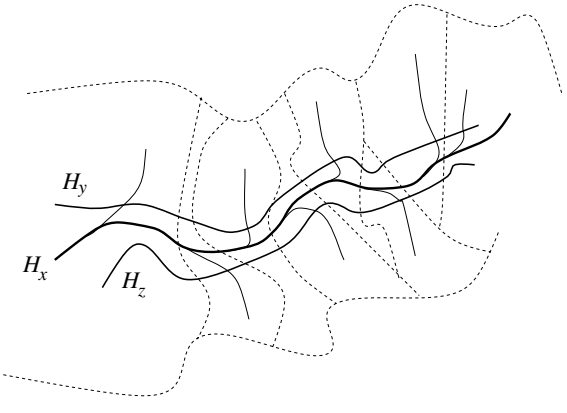


Fig. 8. H_y and H_z envelope a vertebra of H_x

Let $A \cup B$ be a cross-track configuration in G_2. We use it to find a k-star grid in G_0.

There is $\alpha, \beta > 0$ such that for every $a \in A$, $b \in B$ and $0 \leq i < l-1$, the segments $s_{\alpha+i}(a)$ and $s_{\beta+i}(b)$ are parallel. Let $s_i^*(e) = s_{\alpha+i}(e)$ for $e \in A$ and $s_i^*(e) = s_{\beta+i}(e)$ for $e \in B$. We say that $0 \leq i < l-1$ is *bad* if two distinct segments from the set $\{s_i^*(e) \mid e \in A \cup B\}$ intersect.

Observe that we counted at most one crossing for each pair of edges in $A \cup B$, otherwise we would get an empty lens contradicting Claim 1. Therefore, there

are at most $\binom{2k}{2}$ bad values of i. So there are $0 \leq i_0 < i_1 \leq l-1$, $i_1 - i_0 + 2 > l/(\binom{2k}{2}+1) > 2^k + 1$ such that there is no bad i with $i_0 \leq i \leq i_1$. For $i_0 \leq i \leq i_1$, let h_i be the edge of H that is the common origin of the segments $s_i^*(e)$ for $e \in A \cup B$. Order the edges $e \in A \cup B$ according to the order the starting points of $s_i^*(e)$ appear on h_i. Notice that we get the same order for each i. Let a and b be the first and last edge in this order. Let p_i and q_i be the starting point of $s_i^*(a)$ and $s_i^*(b)$, respectively. Let a^* be "relevant" part of a that is, a^* is the interval of a between p_{i_0} and p_{i_1}.

At this point we shift attention from G_2 and H to the original graph G_0 and modify its drawing in the plane. Let S be the set of edges of G_0 containing the edges $A \cup B$ of G_2. Note that S contains $2k$ distinct edges, as edges in A are incident to the same old vertex, therefore they cannot be different segments of an edge of G_0, the same holds for edges of B, while an edge of A and an edge of B intersect, therefore they are not different segments of the same edge. We do not redraw the edge containing a but redraw some segments of other edges making sure that conditions (i) and (ii) defining G_0 are satisfied and furthermore every edge that intersects a^* intersects also all edges in S.

Let $i_0 \leq i < i_1$ and consider the interval of h_i between p_i and q_i, $s_i^*(a)$, the interval of h_{i+1} between p_{i+1} and q_{i+1}, and $s_i^*(b)$. These segments bound a quadrilateral shaped region R_i, with "vertices" p_i, q_i, p_{i+1} and q_{i+1}. See Figure 9. We cannot rule out that some of the regions R_i are not disjoint and, in fact, we cannot even rule out that $h_i = h_{i+1}$ in which case the shape of R_i is more complicated but it does not effect the argument to be presented. The region R_i does not contain vertices, therefore no edge of G_0 entering R_i through $s_i^*(a)$ may leave R_i through $s_i^*(a)$ again, as that would contradict Claim 1. We distinguish three types of edges of G_0 entering R_i through $s_i^*(a)$. Note that an edge can cross $s_i^*(a)$ several times, in this case we consider all the segments of e inside R_i separately.

Type 1: Edge e enters R_i through $s_i^*(a)$ and leaves R_i through $s_i^*(b)$. In this case, e crosses each edge in S.

Type 2: Edge e enters R_i through $s_i^*(a)$ and leaves R_i through h_i.

Type 3: Edge e enters R_i through $s_i^*(a)$ and leaves R_i through h_{i+1}.

We describe procedure Redraw. If there exists $i_0 \leq i < i_1$ with an edge of type 2 crossing $s_i^*(a)$, then we choose an arbitrary such i and the edge e of type 2 crossing $s_i^*(a)$ closest to p_i. Let e_a be the point of e where it enters R_i and e_h be the point where it leaves R_i. Let e'_a and e'_h be points on e outside R_i but close to e_a and e_h, respectively. Replace the interval $e'_a e'_h$ of e by a curve outside R_i, which follows very closely the interval of a between e_a and p_i, and then the interval of h_i between p_i and e_h. In case $h_i = h_{i+1}$ the new curve is drawn similarly, but it does not go outside the region R_i. It is easy to verify that if the new segment of e follows the boundary of R_i close enough, then no new crossings are created and therefore the modified topological graph satisfies properties (i) and (ii). See Figure 9.

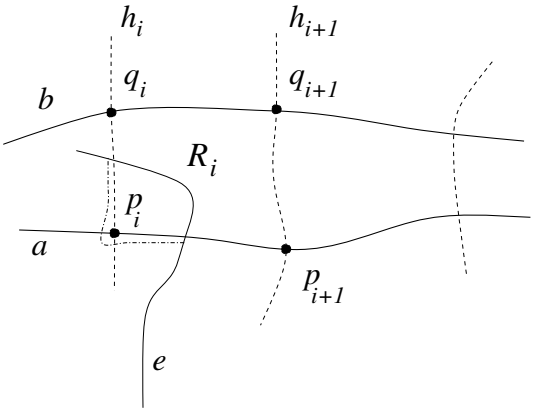


Fig. 9. Procedure REDRAW

If there exists $i_0 \le i < i_1$ and an edge of type 3 crossing $s_i^*(a)$, then we proceed analogously. We choose such an i arbitrarily, we choose a type 3 edge that crosses $s_i^*(a)$ closest to p_{i+1} and redraw the segment of the edge in R_i taking a detour around p_{i+1}.

As long as there is an i, $i_0 \le i < i_1$ with a type 2 or type 3 edge, execute REDRAW.

If a^* enters the region R_i (we cannot rule out this possibility), then REDRAW choosing this i effects other regions R_j. In the extreme case when p_{i+1} is on h_i between p_i and q_i redrawing edges of type 2 we create another crossing with $s_i^*(a)$ itself, possibly another type 2 crossing. Nevertheless, it can be shown that the procedure terminates after finitely many steps. To see this, consider an edge e. The set $\cup_{i=i_0}^{i_1-1} R_i$ divides e into several intervals. Let e^* be one of them. For each crossing p of e^* and a^* let $r(p) = i$ if and only if p is on $s_i^*(a)$. Let $r(e^*, a^*)$ be the sum of all $r(p)$ over all crossings. This sum will either always decrease or always increase when we execute REDRAW involving e^*, therefore e^* is involved in finitely many steps only. To see this "monotonicity condition" notice that each segment of a^* entering R_i has the "same orientation", that is, it enters R_i through h_i and leaves through h_{i+1}.

Let G_0' be the topological graph obtained in the process. All edges of G_0' crossing the curve a^* cross all edges in S. We did not create any additional crossing, so the graph G_0' satisfies properties (i) and (ii) in the definition of G_0. These properties and a result of Schaefer and Stefankovič [9] imply the following.

Claim 7. *For any edge e of G_0' and for any $i > 0$, any 2^i consecutive crossings on e arise from at least i different edges.*

The interval a^* of a crosses H at least 2^k times and we did not "redraw" these segments of edges of G_0. Therefore, we can take 2^k consecutive crossings of a^* in G_0' and by Claim 7 they are from at least k edges. Let Z be a set of k edges of G_0' crossing a^*. Clearly, $S \cup Z$ is a k-star grid in G_0'.

Clearly, the corresponding edges form a k-star grid in F too. This finishes our proof of Theorem 1. □

References

1. P. K. Agarwal, B. Aronov, J. Pach, R. Pollack, and M. Sharir, Quasi-planar graphs have a linear number of edges, *Combinatorica* **17** (1997), 1–9.
2. J. Pach, Geometric graph theory, in: *Surveys in Combinatorics, 1999* (J. D. Lamb and D. A. Preece, eds.), London Mathematical Society Lecture Notes **267**, Cambridge University Press, Cambridge, 1999, 167–200.
3. J. Pach, R. Radoičić, and G. Tóth, Relaxing planarity for topological graphs, in: *Discrete and Computational Geometry* (J. Akiyama, M. Kano, eds.), Lecture Notes in Computer Science **2866**, Springer-Verlag, Berlin, 2003, 221–232.
4. J. Pach, R. Radoičić, and G. Tóth: A generalization of quasi-planarity in: *Towards a Theory of Geometric Graphs*, (J. Pach, ed.), Contemporary Mathematics **342**, AMS, 2004, 177-183.
5. J. Pach, R. Pinchasi, M. Sharir, and G. Tóth, Topological graphs with no large grids, *Graphs and Combinatorics* Special Issue dedicated to Victor Neumann-Lara.
6. J. Pach, R. Pinchasi, G. Tardos, and G. Tóth, Geometric graphs with no self-intersecting path of length three, in: *Graph Drawing* (M. T. Goodrich, S. G. Kobourov, eds.), Lecture Notes in Computer Science **2528**, Springer-Verlag, Berlin, 2002, 295–311.
7. J. Pach, R. Radoičić, G. Tardos, and G. Tóth, Improving the Crossing Lemma by finding more crossings in sparse graphs, *Proceedings of the 20th Annual Symposium on Computational Geometry (SoCG 2004)*, 2004, 76–85.
8. J. Pach and G. Tóth, Graphs drawn with few crossings per edge, *Combinatorica* **17** (1997), 427–439.
9. M. Schaefer and D. Stefankovič, Decidability of string graphs, in: *Proceedings of the 33rd Annual Symposium on the Theory of Computing (STOC 2001)*, 2001, 241–246.

The Geometry of Musical Rhythm

Godfried Toussaint*

School of Computer Science,
McGill University Montréal, Québec, Canada
godfried@cs.mcgill.ca

Dedicated to János Pach on the occasion of his 50th birthday.

Abstract. Musical rhythm is considered from the point of view of geometry. The interaction between the two fields yields new insights into rhythm and music theory, as well as new problems for research in mathematics and computer science. Recent results are reviewed, and new open problems are proposed.

1 Introduction

Imagine a clock which has 16 hours marked on its face instead of the usual 12. Assume that the hour and minute hands have been broken off so that only the second-hand remains. Furthermore assume that this clock is running fast so that the second-hand makes a full turn in about 2 seconds. Such a clock is illustrated in Figure 1. Now start the clock ticking at "noon" (16 O'clock) and let it keep running for ever. Finally, strike a bell at positions 16, 3, 6, 10 and 12, for a total of five strikes per clock cycle. These times are marked with a bell in Figure 1. The resulting pattern rings out a seductive rhythm which, in a short span of fifty years during the last half of the 20th century, has managed to conquer our planet. It is known around the world (mostly) as the *Clave Son* from Cuba. However, it is common in Africa, and probably travelled from Africa to Cuba with the slaves [65]. In Africa it is traditionally played with an iron bell. In Cuba it is played with two sticks made of hard wood also called *claves* [43]. More relevant to this paper, there exist purely geometric properties that may explain the world-wide popularity of this *clave* rhythm [58].

The *Clave Son* rhythm is usually notated for musicians using standard music notation which affords many ways of expressing a rhythm. Four such examples are given in the top four lines of Figure 2. The fourth line displays the rhythm using the smallest convenient durations of notes and rests. Western music notation is not ideally suited to represent African music [3], [18]. The fifth and sixth lines show two popular ways of representing rhythms that avoid Western notation. The representation on line five is called the *Box Notation Method* developed by Philip Harland at the University of California in Los Angeles in 1962 and is also known as TUBS (Time Unit Box System). The TUBS representation is popular among ethnomusicologists [18], and invaluable to percussionists not familiar

* This research was partially supported by NSERC and FCAR.

J. Akiyama et al. (Eds.): JCDCG 2004, LNCS 3742, pp. 198–212, 2005.

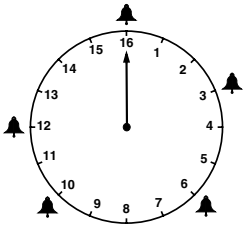


Fig. 1. A clock divided into sixteen equal intervals of time

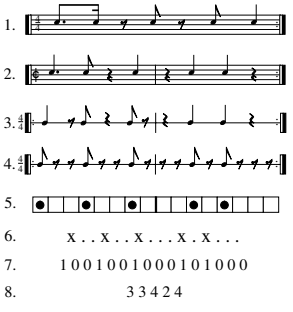


Fig. 2. Eight common ways of representing the *clave Son* rhythm

with Western notation. It is also convenient for experiments in the psychology of rhythm perception, where a common variant of this method is simply to use one symbol for the note and another for the pause [17], as illustrated in line six. In computer science the clave *Son* might be represented as the 16-bit binary sequence shown on line seven. Finally, line eight depicts the interval length representation of the clave *Son*, where the numbers denote the lengths (in shortest convenient units) of the durations between consecutive onsets (beginning points in time of notes). The compactness and ease of use in text, of this numerical interval-length representation, are two obvious advantages, but its iconic value is minimal. For a description of additional (more geometric) methods used to represent rhythms see [61].

In this paper several geometric properties of musical rhythm are analysed from the musicological and mathematical points of view. Several connecting bridges between music theory, mathematics, and computer science are illuminated. Furthermore, new open problems at the interface are proposed.

2 Measures of Rhythmic Evenness

Consider the following three 12/8 time rhythms expressed in box-like notation: [x . x . x . x . x . x .], [x . x . x x . x . x . x] and [x . . . x x . . x x x .]. It is intuitively clear that the first rhythm is more *even* (well spaced) than the second,

and the second is more even than the third. In passing we note that the second rhythm is internationally the most well known of all the African timelines. It is traditionally played on an iron bell, and is known on the world scene mainly by its Cuban name *Bembé* [60]. Traditional rhythms have a tendency to exhibit such properties of evenness to one degree or another. Therefore mathematical measures of evenness, as well as other geometric properties, find application in the new field of mathematical ethnomusicology [9], [62], where they may help to identify, if not explain, cultural preferences of rhythms in traditional music.

2.1 Maximally Even Rhythms

In music theory much attention has been devoted to the study of intervals used in pitch scales [24], but relatively little work has been devoted to the analysis of time duration intervals of rhythm. Two notable exceptions are the books by Simha Arom [3] and Justin London [36]. Clough and Duthett [12] introduced the notion of *maximally even sets* with respect to scales represented on a circle. According to Block and Douthett [5], Douthet and Entringer went further by constructing several mathematical measures of the amount of *evenness* contained in a scale (see the discussion on p. 41 of [5]). One of their measures simply adds all the interval arc-lengths (geodesics along the circle) determined by all pairs of pitches in the scale. This definition may be readily transferred to durations in time, of cyclic rhythms represented on a unit circle, as illustrated in Figure 1. However, the measure is too coarse to be useful for comparing rhythm timelines such as those studied in [58] and [60]. Admitedly, the measure does differentiate between rhythms that differ widely from each other. For example, the two four-onset rhythms [x . . . x . . . x . . . x . . .] and [x . x . x . . x . . . . . .] yield evenness values of 32 and 23, respectively, reflecting clearly that the first rhythm is more evenly spaced than the second. However, *all* six fundamental 4/4 time clave/bell patterns illustrated in Figure 3, and discussed in [58], have an equal evenness value of 48, and yet the *Rumba* clave is clearly more uneven than the *Bossa-Nova* clave. The use of interval chord-lengths (as opposed to geodesic distances), proposed by Block and Douthet [5], yields a more discriminating measure, and is therefore the preferred measure of evenness.

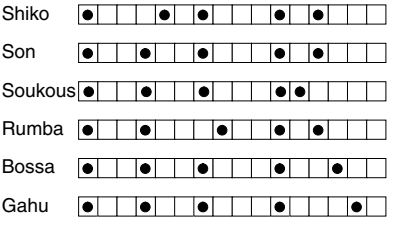


Fig. 3. The six fundamental 4/4 time clave and bell patterns in box notation

2.2 Maximizing the Sum of Distances

The evenness measure of Block and Douthet [5], which sums all the pairwise straight-line distances of a set of points on the circular lattice, brings up the question of which configurations of points (rhythms) achieve maximum evenness. In fact, this problem had been investigated by the Hungarian mathematician Fejes Tóth [55] some forty years earlier without the restriction of placing the points on the circular lattice. He showed that the sum of the pairwise distances determined by n points in a circle is maximized when the points are the vertices of a regular n-gon inscribed on the circle.

The discrete version of this problem, of interest in music theory [5], is a special case of several problems studied in computer science and operations research. In graph theory it is a special case of the maximum-weight clique problem [22]. In operations research it is studied under the umbrella of obnoxious facility location theory. In particular, it is one of the *dispersion* problems called the discrete p-maxian location problem [20], [21]. Because these problems are computationally difficult, researchers have proposed approximation algorithms [30], and heuristics [21], [66], for the general problem, and have sought efficient solutions for simpler special cases of the problem [49], [54].

Fejes Tóth also showed in [55] that in three dimensions four points on the sphere maximize the sum of their pairwise distances when they are the vertices of a regular tetrahedron. The problem remains open for more than four points on the sphere. Some upper and lower bounds on the maximum value that the sum may attain are known. Alexander [1] proved an upper bound of $(2/3)n^2 - (1/2)$. It has also been shown that the points must be well spaced in some sense. Stolarsky [53] proved that if n points are placed on the sphere so that the sum of their distances is maximized, then the distance between the closest pair is at least $2/3n$. Additional bounds and references may be found in the survey paper by Chakerian and Klamkin [8].

In 1959 Fejes Tóth [56] asked a more difficult question by relaxing the spherical constraint. He asked for the maximum sum of distances of n points in the plane under the constraint that the diameter of the set is at most one. Pillichshammer [46] found upper bounds on this sum but gave exact solutions only for $n = 3, 4$, and 5. For $n = 3$ the points form the vertices of an equilateral triangle of unit side lengths. For $n = 5$ the points form the vertices of a regular pentagon with unit length diagonals. For $n = 4$ the solution may be obtained by placing three points on the vertices of a Reuleaux unit-diameter triangle, and the fourth point at a midpoint of one of the Reuleaux triangle arcs. The problem remains open for more than five points in the plane. In the mathematics literature such problems have also been investigated with the Euclidean distance replaced by the squared Euclidean distance [45], [47], [67].

3 Interval Spectra of Rhythms

Rather than focusing on the *sum* of all the inter-onset duration intervals of a rhythm, as in the preceeding section, here we examine the shape of the *spectrum*

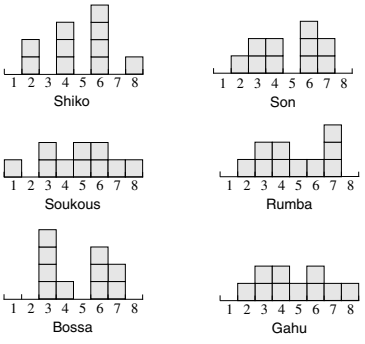


Fig. 4. The full-interval histograms of the 4/4 time clave-bell patterns

of the frequencies with which all the inter-onset durations are present in a rhythm. Again we assume rhythms are represented as points on a circle as in Figure 1. In music theory this spectrum is called the *interval vector* (or full-interval vector) [39]. For example, the interval vector for the clave *Son* pattern of Figure 1 is given by [0,1,2,2,0,3,2,0]. It is an 8-dimensional vector because there are eight different possible duration intervals (geodesics on the circle) between pairs of onsets defined on a 16-unit circular lattice. For the clave *Son* there are 5 onsets (10 pairs of onsets), and therefore the sum of all the vector elements is equal to ten. A more compelling and useful visualization of an interval vector is as a histogram. Figure 4 shows the histograms (interval vectors) of the full-interval sets of all six (4/4)-time clave-bell patterns pictured in Figure 3.

Examination of the six histograms leads to questions of interest in a variety of fields of enquiry: musicology, geometry, combinatorics, and number theory. For example, David Locke [35] has given musicological explanations for the characterization of the *Gahu* bell pattern (shown at the bottom of Figure 3) as "rhythmically potent", exhibiting a "tricky" quality, creating a "spiralling effect", causing "ambiguity of phrasing" leading to "aural illusions." Comparing the full-interval histogram of the *Gahu* pattern with the five other histograms in Figure 4 leads to the observation that the *Gahu* is the only pattern that has a histogram with a maximum height of 2, and consisting of a single connected component of occupied histogram cells. The only other rhythm with a single connected component is the *Rumba*, but it has 3 intervals of length 7. The only other rhythm with maximum height 2 is the *Soukous*, but it has two connected components because there is no interval of length 2. Only *Soukous* and *Gahu* use seven out of the eight possible interval durations.

The preceeding observations suggest that perhaps other rhythms with relatively uniform (flat) histograms, and few, if any, gaps may be interesting from the musicological point of view as well. Does the histogram shape of the *Gahu* rhythm play a significant role in the rhythm's special musicological properties? If so, this geometric property could provide a heuristic for the discovery and automatic generation of other "good" rhythms. Such a tool could be used for

music composition by computer. With this in mind one may wonder if rhythms exist with the most extreme values possible for these properties. Let us denote the family of all rhythms consisting of k onsets in a time span cycle of n units by $R[k, n]$. In other words $R[k, n]$ consists of all n-bit cyclic binary sequences with k 1's. Thus all the 4/4 time clave-bell patterns in Figure 3 belong to $R[5, 16]$.

The first natural question that arizes is whether there exist any rhythms whose inter-onset intervals have perfectly flat histograms of height one with no gaps. This is clearly not possible with $R[5, 16]$. Since there are only 8 possible different interval lengths and 10 distance pairs, there must exist at least one histogram cell with height greater than one. The second natural question is whether there exists an $R[5, 16]$ rhythm that uses all eight intervals. The answer is yes; one such pattern is [x x . . . x . x . . . . . x . .] with interval vector given by [1,1,1,2,1,2,1,1]. However, the rhythm [x x . . x . x . . . . .] belonging to the family $R[4, 12]$ depicted on the circle in Figure 5 (a) does have a perfectly flat histogram: every one of the inter-onset intervals occurs exactly once; its interval vector is [1,1,1,1,1,1].

For a rhythm to have "drive" it should not contain silent intervals that are too long, such as the silent interval of length six in Figure 5 (a). One may wonder if there are other rhythms in $R[4, 12]$ with interval vectors equal to [1,1,1,1,1,1], and if they exist, are there any with shorter silent gaps. It turns out that the answer is yes. The rhythm [x x . x . . . x . . . .] shown in Figure 5 (b) satisfies all these properties; its longest silent gap is five units.

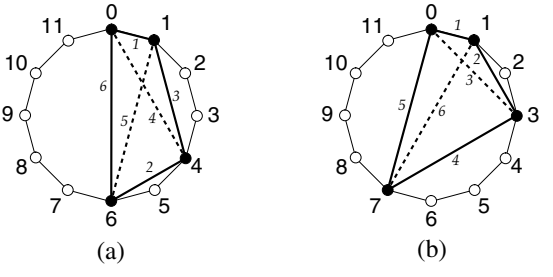


Fig. 5. Two flat-histogram rhythms

A cyclic sequence such as [x x . . x . x . . . . .] is an instance of a *necklace* with "beads" of two colors [33]; it is also an instance of a *bracelet*. Two necklaces are considered the same if one can be rotated so that the colors of its beads correspond, one-to-one, with the colors of the other. Two bracelets are considered the same if one can be rotated or turned over (mirror image) so that the colors of their beads are brought into one-to-one correspondence. The rhythms in Figure 5 clearly maintain the same interval vector (histogram) if they are rotated, although this rotation may yield rhythms that sound quite different. Therefore it is useful to distinguish between rhythm-necklaces, and just plain rhythms (necklace *instances* in a fixed rotational position with respect to the underlying beat).

The number of onsets in a rhythm is called the *density* in combinatorics, and efficient algorithms exist for generating all the necklaces with a specified fixed density [51].

3.1 Rhythms with Specified Duration Multiplicities

In 1989 Paul Erdős [19] asked whether one could find n points in the plane (no three on a line and no four on a circle) so that for every i, $i = 1, ...n-1$ there is a distance determined by these points that occurs exactly i times. Solutions have been found for $2 \leq n \leq 8$. Palásti [44] considered a variant of this problem with further restrictions: no three form a regular triangle, and no one is equidistant from three others. A musical scale whose pitch intervals are determined by points drawn on a circle, and that has the property asked for by Erdős is known in music theory as a *deep* scale [31]. We will transfer this terminoly from the pitch domain to the time domain and refer to cyclic rhythms with this property as *deep* rhythms. Deep scales have been studied as early as 1967 by Carlton Gamer [26], [27], and it turns out that the ubiquitous Western *diatonic* scale is a deep scale. Also, the *Bembé* rhythm mentioned in the preceeding is a deep rhythm since it is isomorphic to the diatonic scale.

The question posed by Erdős is closely related to the general problem of reconstructing sets from interpoint distances: given a distance multiset, construct all point sets that realize the distance multiset. This problem has a long history in crystallography [34], and more recently in DNA sequencing [52]. Two noncongruent sets of points, such as the two different necklaces of Figures 5, are called *homometric* if the multisets of their pairwise distances are the same [50]. For an extensive survey and bibliography of this problem see [34]. The special cases relevant to the theory of rhythm, when points lie on a line or circle, have received some attention, and are called the *turnpike* problem and the *beltway* problem, respectively [34].

Some existing results on homometric sets on the circular lattice are most relevant to the theory of rhythm. For example many drumming patterns have two sounds (such as the high and low congas) that together tile the lattice. It is known that every n-point subset of the regular $2n$-gon is homometric to its

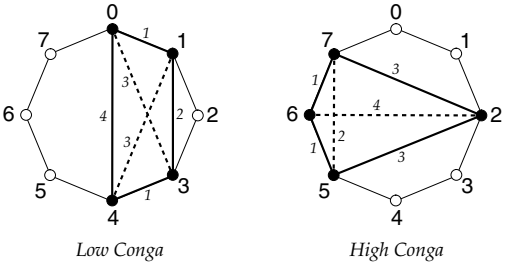


Fig. 6. Two complementary homometric rhythms

complement [34]. This leads immediately to a simple method for the generation of two-tone tiling rhythms in which each of the two parts is homometric. One example is illustrated in Figure 6. It is also known that two rhythms are homometric if, and only if, their complements are [10]. This concept provides another tool that may find use in music composition by computer.

4 Measuring the Similarity of Rhythms

At the heart of any algorithm for comparing, recognizing or classifying rhythms, lies a measure of the similarity between a pair of rhythms. The type of similarity measure chosen is in part predetermined by the manner in which the rhythm is represented. Furthermore, the design of a measure of similarity is guided by at least two fundamental ideas: *what* should be measured, and *how* should it be measured.

There exists a wide variety of methods for measuring the similarity of two rhythms represented by strings of symbols [59]. Indeed the resulting approximate pattern matching problem is a classical problem in pattern recognition and computer science in general [16]. Traditionally similarity between two patterns is measured by a simple template matching operation. More recently similarity has been measured with more powerful and complex functions such as the *earth mover's distance* [7], [64], weighted geometric matching functions [37], the *swap* distance [61], and the *directed-swap* distance [15], [13].

4.1 Swap Distance

A well known distance measure between two n-bit binary sequences is the Hamming distance trivially computed in $O(n)$ time. However, this distance measure is not appropriate for rhythm similarity, when used with a binary-string representation, because although it measures the existence of an onset missmatch, it does not measure how far the missmatch occurs. Furthermore, if a note onset of a rhythm is displaced a large distance, the resulting modified rhythm will in general sound more different than if the onset is moved a small distance.

To combat this inherent weakness of the Hamming distance, variants and generalizations of the Hamming distance have been proposed over the years. One early generalization is the *edit* distance which allows for insertions and deletions of notes. Discussions of the application of the *edit-distance* to the measurement of similarity in music can be found in Mongeau and Sankoff [40] and Orpen and Huron [42]. A noteworthy more recent generalization is the *fuzzy* Hamming distance [6] which allows *shifting* of notes as well as insertions and deletions. Using *dynamic programming* these distances may be computed in $O(n^2)$ time.

The problem of comparing two binary strings of the same length with the same number of one's suggests an extremely simple edit operation called a *swap*. A swap is an interchange of a one and a zero that are adjacent to each other in the binary string. Interchanging the position of elements in strings of numbers is a fundamental operation in many sorting algorithms [14]. However, in the sorting literature a swap may interchange non-adjacent elements. When the elements

are required to be adjacent, the swap is called a *mini-swap* or *primitive-swap* [4]. Here we use the shorter term *swap* to mean the interchange of two adjacent elements. The swap distance between two rhythms is the *minimum* number of swaps required to convert one rhythm to the other. The *swap* distance may be viewed as a simplified version of the *generalized* Hamming distance [6], where only the shift operation is used, and the cost of the shift is equal to its length. Such a measure of dissimilarity appears to be more appropriate than the Hamming distance between the binary vectors in the context of rhythm similarity [58], [60]. The swap distance may also be viewed as a special case of the more general *earth mover's distance* (also called *transportation distance*) used by Typke et al. [64] to measure melodic similarity. Given two sets of points called supply points and demand points, each assigned a weight of material, the earth movers distance measures the minimum amount of work (weight times distance) required to transport material from the supply points to the demand points. No supply point can supply more weight than it has and no demand point receives more weight than it needs. Typke et al. [64] solve this problem using linear programming, a relatively costly computational method. The swap distance is a one dimensional version of the earth mover's distance with all weights equal to one. Furthermore, in the case where both binary sequences have the same number of "one's" (or onsets), there is a one-to-one correspondence between the indices of the ordered onsets of the sequences [32].

The swap distance may of course be computed by actually performing the swaps, but this is inefficient. If X has one's in the first $n/2$ positions and zero's elsewhere, and if Y has one's in the last $n/2$ positions and zero's elsewhere, then a quadratic number of swaps would be required. On the other hand, if we compare the distances instead, a much more efficient algorithm results. First scan the binary sequence and store a vector of the x-coordinates at which the k onsets occur. Then the swap distance between the two onset-coordinate vectors U and V with k onsets is given by:

$$d_{SWAP}(U, V) = \sum_{i=1}^{k} |u_i - v_i| \tag{1}$$

Computing U and V from X and Y is done trivially in $O(n)$ time with a simple scan. Therefore $O(n)$ time suffices to compute $d_{SWAP}(U, V)$, resulting in a large gain over the linear or dynamic programming algorithms.

5 Introducing Melody into Rhythm

ÓMaidín [41] proposed a geometric measure of the distance between two melodies modelled as monotonic pitch-duration rectilinear functions of time as depicted in Fig. 7. ÓMaidín measures the distance between the two melodies by the area between the two polygonal chains (shown shaded in Fig. 7). Note that if the area under each melody contour is equal to *one*, the functions can be viewed as probability distributions, and in this case ÓMaidín's measure is identical to

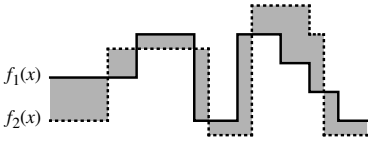


Fig. 7. Two melodies as rectilinear pitch-duration functions of time

the classical Kolmogorov *variational distance* used to measure the difference between two probability distributions [57]. If the number of vertices (vertical and horizontal segments) of the two polygonal chains is n then it is trivial to compute ÓMaidín's distance in $O(n)$ time using a line-sweep algorithm.

In a more general setting, such as music information retrieval systems, we are given a short query segment of music, denoted by the polygonal chain $Q = (q_1, q_2, ..., q_m)$, and a longer stored segment $S = (s_1, s_2, ..., s_n)$, where $m < n$. Furthermore, the query segment may be presented in a different *key* (transposed in the vertical direction) and in a different *tempo* (scaled linearly in the horizontal direction). Note that the number of keys (horizontal levels) is a small finite constant. Time is also quantized into fixed intervals (such as eighth or sixteenth notes). In this context it is desired to compute the minimum area between the two contours under vertical translations and horizontal scaling of the query. Francu and Nevill-Manning [25] claim that this distance measure can be computed in $O(mn)$ time but they do not describe their algorithm in detail.

6 New Open Problems

Let us assume that we are given a circular lattice with n points (evenly spaced), and we would like to create a rhythm consisting of k onsets by choosing k of these n lattice points. For example, perhaps $n = 16$ and $k = 5$ as in Figure 1. Furthermore we would like to select the k onsets that maximize the sum of the lengths of all pairwise chords between these onsets. Evaluating all n-choose-k subsets may in general be too costly. However, interesting rhythms often have additional musicological constraints that may be couched in a geometric setting [3], and may permit simpler solutions. One may also consider an approximation method using the following *snap* heuristic: construct a regular k-gon with one vertex coincident with one lattice point, and then move the remaining onset points to their nearest lattice points. One would expect such a rhythm to have high evenness value. How close to optimal is this procedure?

The two sequences shown in Figure 5 are the only possible rhythm bracelets with flat histograms, for any values of k greater than three [48]. Therefore in order to be able to generate additional rhythms with near-flat histograms the constraints outlined in the preceeding need to be relaxed. We may proceed in several directions. For example, it is desirable for timelines that can be played fast, and that "roll along" (such as the *Gahu* already discussed), that the rhythm contain silent gaps that are neither too short nor too long. Therefore it would be

desirable to be able to efficiently generate rhythms that either contain completely prescribed histogram shapes, or have geometric constraints on their shapes, and to find good approximations when such rhythms do not exist.

The analysis of cyclic rhythms suggests another variant of the question asked by Erdős. First note that if a rhythm R$[k, n]$ has $k \leq n/2$, then a solution to Erdős' problem always exists: simply place points at positions $0, 1, 2, ..., k$. However, as mentioned in the preceeding, from the musicological point of view it may be desirable sometimes not to allow empty semicircles. These constraints suggest the following problem. Is it possible to have k points on a circular lattice of n points so that for every i, $i = k_s, k_{s+1}, ..., k_f$ (s and f are pre-specified) there is a *geodesic* distance that occurs exactly i times, with (or without) the further restriction that there is no empty semicircle?

The preceeding discussion on the swap distance was restricted to comparing two linear strings. However, many rhythms such as the timelines considered here are cyclic, and there are applications, such as music information retrieval, where it is desired to compute the best alignement of two cyclic rhythms over all possible rotations. In other words, it is of interest to compute the distance between two rhythms minimized over all possible rotations of one with respect to the other. Some work has been done with cyclic string matching for several definitions of string similarity [28], [38], [11]. Consider two binary sequences of length n and density k (k ones and $(n - k)$ zeros). It is desired to compute the minimum swap distance between the two strings under all possible alignments. I call this distance the *cyclic swap-distance* or also the *necklace swap-distance*, since it is the swap distance between two necklaces. From the preceeding discussion it follows that the cyclic swap distance may be computed in $O(n^2)$ time by using the linear-time algorithm in each of the n possible alignment positions of the two rhythms. Note that swaps may be performed in whatever direction (clockwise or counter-clockwise) yields the fewest swaps. Can the cyclic swap distance be computed in $o(n^2)$ time? In contrast, if the swap distance is replaced with the Hamming distance, then the cyclic (or necklace) Hamming distance may be computed in $O(n \log n)$ time with the Fast Fourier Transform [23], [29].

The work of ÓMaidín [41] and Francu and Nevill-Manning [25] suggests several interesting open problems. In the acoustic signal domain the vertical transposition is continuous rather than discrete. The same can be said for the time axis. What is the complexity of computing the minimum area between a query $Q = (q_1, q_2, ..., q_m)$ and a longer stored segment $S = (s_1, s_2, ..., s_n)$ under these more general conditions?

A simpler variant of the melody similarity problem concerns acoustic *rhythmic* melodies, i.e., cyclic rhythms with notes that have pitch as a continuous variable. Here we assume two rhythmic melodies of the same length are to be compared. Since the melodies are cyclic rhythms they can be represented as closed curves on the surface of a cylinder. What is the complexity of computing the minimum area between the two rectilinear polygonal chains under rotations around the cylinder and translations along the length of the cylinder? Aloupis et al. [2] present an $O(n)$ time algorithm to compute this measure if rotations

are not allowed, and an $O(n^2 \log n)$ time algorithm for unrestricted motions (rotations around the cylinder and translations along the length of the cylinder). Can this complexity be improved?

In the preceeding sections several tools were pointed out that can be used for computer composition. We close the paper by mentioning one additional tool for automatically selecting rhythm timelines that can be used for generating new music. In [63] it is shown that the *Euclidean* algorithm for finding the greatest common divisor of two numbers can be used to generate very good rhythm timelines when the two numbers that serve as input to the *Euclidean* algorithm are the number of onsets (k) and the time-span (n), respectively, of the desired rhythm. The resulting rhythms, called *Euclidean* rhythms, are particularly attractive when k and n are relatively prime. What is the relation between *Euclidean* rhythms and maximally even rhythms under the different definitions of *even* considered in the preceeding?

References

1. R. Alexander. On the sum of distances between n points on a sphere. *Acta. Math. Acad. Sci. Hungar.*, 23:443–448, 1972.
2. Greg Aloupis, Thomas Fevens, Stefan Langerman, Tomomi Matsui, Antonio Mesa, Yurai Nuñez, David Rappaport, and Godfried Toussaint. Computing a geometric measure of the similarity between two melodies. In *Proc. 15th Canadian Conf. Computational Geometry*, pages 81–84, Dalhousie University, Halifax, Nova Scotia, Canada, August 11-13 2003.
3. Simha Arom. *African Polyphony and Polyrhythm.* Cambridge University Press, Cambridge, England, 1991.
4. Therese Biedl, Timothy Chan, Erik D. Demaine, Rudolf Fleischer, Mordecai Golin, James A. King, and Ian Munro. Fun-sort – or the chaos of unordered binary search. *Discrete Applied Mathematics*, 144(Issue 3):231–236, December 2004.
5. Steven Block and Jack Douthett. Vector products and intervallic weighting. *Journal of Music Theory*, 38:21–41, 1994.
6. Abraham Bookstein, Vladimir A. Kulyukin, and Timo Raita. Generalized Hamming distance. *Information Retrieval*, 5(4):353–375, 2002.
7. Sung-Hyuk Cha and Sargur N. Srihari. On measuring the distance between histograms. *Pattern Recognition*, 35:1355–1370, 2002.
8. G. D. Chakerian and M. S. Klamkin. Inequalities for sums of distances. *The American Mathematical Monthly*, 80(9):1009–1017, November 1973.
9. M. Chemillier. Ethnomusicology, ethnomathematics. The logic underlying orally transmitted artistic practices. In G. Assayag, H. G. Feichtinger, and J. F. Rodrigues, editors, *Mathematics and Music*, pages 161–183. Springer, 2002.
10. C. Chieh. Analysis of cyclotomic sets. *Zeitschrift Kristallographie*, 150:261–277, 1979.
11. Kuo-Liang Chung. An improved algorithm for solving the banded cyclic string-to-string correction problem. *Theoretical Computer Science*, 201:275–279, 1998.
12. J. Clough and J. Douthett. Maximally even sets. *Journal of Music Theory*, 35:93–173, 1991.
13. Justin Colannino and Godfried Toussaint. An algorithm for computing the restriction scaffold assignment problem in computational biology. *Information Processing Letters*, 2005. in press.

14. N. G. de Bruijn. Sorting by means of swapping. *Discrete Mathematics*, 9:333–339, 1974.
15. Miguel Díaz-Bañez, Giovanna Farigu, Francisco Gómez, David Rappaport, and Godfried T. Toussaint. El compás flamenco: a phylogenetic analysis. In *Proc. BRIDGES: Mathematical Connections in Art, Music and Science*, Southwestern College, Kansas, July 30 - August 1 2004.
16. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., New York, 2001.
17. Douglas Eck. A positive-evidence model for classifying rhythmical patterns. Technical Report IDSIA-09-00, Instituto Dalle Molle di studi sull'intelligenza artificiale, Manno, Switzerland, 2000.
18. Laz E. N. Ekwueme. Concepts in African musical theory. *Journal of Black Studies*, 5(1):35–64, September 1974.
19. Paul Erdős. Distances with specified multiplicities. *American Math. Monthly*, 96:447, 1989.
20. E. Erkut, T. Baptie, and B. von Hohenbalken. The discrete p-maxian location problem. *Computers in Operations Research*, 17(1):51–61, 1990.
21. E. Erkut, T. Ulkusal, and O. Yenicerioglu. A comparison of p-dispersion heuristics. *Computers in Operations Research*, 21(10):1103–1113, 1994.
22. Sándor P. Fekete and Henk Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38:501–511, 2004.
23. M. J. Fisher and M. S. Patterson. String matching and other products. In Richard M. Karp, editor, *Complexity of Computation*, volume 7, pages 113–125. SIAM-AMS, 1974.
24. A. Forte. *The Structure of Atonal Music*. Yale Univ. Press, New Haven, 1973.
25. Cristian Francu and Craig G. Nevill-Manning. Distance metrics and indexing strategies for a digital library of popular music. In *Proceedings of the IEEE International Conference on Multimedia and EXPO (II)*, 2000.
26. Carlton Gamer. Deep scales and difference sets in equal-tempered systems. In *Proceedings of the Second Annual Conference of the American Society of University Composers*, pages 113–122, 1967.
27. Carlton Gamer. Some combinational resources of equal-tempered systems. *Journal of Music Theory*, 11:32–59, 1967.
28. J. Gregor and M. G. Thomason. Efficient dynamic programming alignment of cyclic strings by shift elimination. *Pattern Recognition*, 29:1179–1185, 1996.
29. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, 1997.
30. R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21:133–137, 1997.
31. Timothy A. Johnson. *Foundations of Diatonic Theory: A Mathematically Based Approach to Music Fundamentals*. Key College Publishing, Emeryville, California, 2003.
32. Richard M. Karp and Shou-Yen R. Li. Two special cases of the assignment problem. *Discrete Mathematics*, 13:129–142, 1975.
33. Michael Keith. *From Polychords to Pólya: Adventures in Musical Combinatorics*. Vinculum Press, Princeton, 1991.
34. Paul Lemke, Steven S. Skiena, and Warren D. Smith. Reconstructing sets from interpoint distances. Tech. Rept. DIMACS-2002-37, 2002.
35. David Locke. *Drum Gahu: An Introduction to African Rhythm*. White Cliffs Media, Gilsum, New Hampshire, 1998.

36. Justin London. *Hearing in Time*. Oxford University Press, Oxford, England, 2004.
37. Anna Lubiw and Luke Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *Proceedings of the Fifth International Symposium on Music Information Retrieval*, pages 289–296, Barcelona, Spain, October 2004.
38. Maurice Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35:73–78, 1990.
39. Brian J. McCartin. Prelude to musical geometry. *The College Mathematics Journal*, 29(5):354–370, 1998.
40. M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.
41. D. ÓMaidín. A geometrical algorithm for melodic difference. *Computing in Musicology*, 11:65–72, 1998.
42. Keith S. Orpen and David Huron. Measurement of similarity in music: A quantitative approach for non-parametric representations. In *Computers in Music Research*, volume 4, pages 1–44. 1992.
43. Fernando Ortiz. *La Clave*. Editorial Letras Cubanas, La Habana, Cuba, 1995.
44. Ilona Palásti. A distance problem of Paul Erdős with some further restrictions. *Discrete Mathematics*, 76:155–156, 1989.
45. F. Pillichshammer. On the sum of squared distances in the Euclidean plane. *Arch. Math.*, 74:472–480, 2000.
46. F. Pillichshammer. A note on the sum of distances in the Euclidean plane. *Arch. Math.*, 77:195–199, 2001.
47. F. Pillichshammer. On extremal point distributions in the Euclidean plane. *Acta. Math. Acad. Sci. Hungar.*, 98(4):311–321, 2003.
48. John Rahn. *Basic Atonal Theory*. Schirmer, 1980.
49. S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, March-April 1994.
50. Joseph Rosenblatt and Paul Seymour. The structure of homometric sets. *SIAM Journal of Algebraic and Discrete Methods*, 3:343–350, 1982.
51. Frank Ruskey and Joe Sawada. An efficient algorithm for generating necklaces with fixed density. *SIAM Journal of Computing*, 29(2):671–684, 1999.
52. S. S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56:275–294, 1994.
53. Keneth B. Stolarsky. Spherical distributions of N points with maximal distance sums are well spaced. *Proceedings of the American Mathematical Society*, 48(1):203–206, March 1975.
54. Arie Tamir. Comments on the paper: "Heuristic and special case algorithms for dispersion problems" by S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. *Operations Research*, 46:157–158, 1998.
55. L. Fejes Tóth. On the sum of distances determined by a pointset. *Acta. Math. Acad. Sci. Hungar.*, 7:397–401, 1956.
56. L. Fejes Tóth. Über eine Punktverteilung auf der Kugel. *Acta. Math. Acad. Sci. Hungar.*, 10:13–19, 1959.
57. Godfried T. Toussaint. Sharper lower bounds for discrimination information in terms of variation. *IEEE Transactions on Information Theory*, pages 99–100, January 1975.
58. Godfried T. Toussaint. A mathematical analysis of African, Brazilian, and Cuban *clave* rhythms. In *Proc. of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 157–168, Towson University, MD, July 27-29 2002.

59. Godfried T. Toussaint. Algorithmic, geometric, and combinatorial problems in computational music theory. In *Proceedings of X Encuentros de Geometria Computacional*, pages 101–107, University of Sevilla, Sevilla, Spain, June 16-17 2003.
60. Godfried T. Toussaint. Classification and phylogenetic analysis of African ternary rhythm timelines. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 25–36, Granada, Spain, July 23-27 2003.
61. Godfried T. Toussaint. A comparison of rhythmic similarity measures. In *Proc. 5th International Conference on Music Information Retrieval*, pages 242–245, Barcelona, Spain, October 10-14 2004. Universitat Pompeu Fabra.
62. Godfried T. Toussaint. A mathematical measure of preference in African rhythm. In *Abstracts of Papers Presented to the American Mathematical Society*, volume 25, page 248, Phoenix, January 7-10 2004. American Mathematical Society.
63. Godfried T. Toussaint. The Euclidean algorithm generates traditional musical rhythms. In *Proc. of BRIDGES: Mathematical Connections in Art, Music and Science*, Banff, Canada, July 31 - August 3 2005.
64. Rainer Typke, Panos Giannopoulos, Remco C. Veltkamp, Frans Wiering, and René van Oostrum. Using transportation distances for measuring melodic similarity. In Holger H. Hoos and David Bainbridge, editors, *Proceedings of the Fourth International Symposium on Music Information Retrieval*, pages 107–114, Johns Hopkins University, Baltimore, 2003.
65. Chris Washburne. Clave: The African roots of salsa. In *Kalinda!: Newsletter for the Center for Black Music Research*. Columbia University, New York, 1995. Fall-Issue.
66. Douglas J. White. The maximal dispersion problem and the "first point outside the neighbourhood heuristic. *Computers in Operations Research*, 18(1):43–50, 1991.
67. H. S. Witsenhausen. On the maximum of the sum of squared distances under a diameter constraint. *The American Mathematical Monthly*, 81(10):1100–1101, December 1974.

Author Index