Freie Universität Berlin

34th EuroCG 18

Extended Abstracts

Preface

The 34th European Workshop on Computational Geometry (EuroCG '18), was held at Freie Universität Berlin, Berlin, Germany, on March 21—23, 2018. EuroCG is an annual workshop that combines a strong scientific tradition with a friendly and informal atmosphere. The workshop is a forum where researchers can meet, discuss their work, present their results, and establish scientific collaborations, in order to promote research in the field of Computational Geometry, within Europe and beyond.

We received 78 submissions, which underwent a limited refereeing process by the program committee in order to ensure some minimal standards and to check for plausibility. We selected 75 submissions for presentation at the workshop. One submission was later withdrawn. EuroCG does not have formally published proceedings; therefore, we expect most of the results presented here to be also submitted to peer-reviewed conferences and/or journals. This book of abstracts, available through the EuroCG '18 web site, should be regarded as a collection of preprints. In addition to the 74 contributed talks, this book also contains abstracts of the three invited lectures, given by Nina Amenta, Prosenjit Bose, and Raúl Rojas.

Many thanks to all authors, speakers, and invited speakers for their participation, and to the members of the program committee and all external reviewers for their insightful comments. We gratefully thank the supporters of EuroCG '18 for making this event possible and helping to keep the registration fees low: Freie Universität Berlin, keylight GmbH, and the German Research Foundation (DFG grant MU 3501/4-1). Special thanks to all members of the organizing committee and members of the administration at Freie Universität Berlin, for their work that made EuroCG '18 possible.

March 2018

Matias Korman and Wolfgang Mulzer
EuroCG '18 chairs

Organizing Committee

- Helmut Alt
- Bahareh Banyassady
- Jonas Cleve
- Claudia Dieckmann
- Heike Eckart
- Frank Hoffmann
- Boris Klemz
- Katharina Klost
- Tamara Knoll
- Klaus Kriegel
- Wolfgang Mulzer (chair)
- Günter Rote
- Nadja Scharf
- Max Willert

Program Committee

- Mikkel Abrahamsen, University of Copenhagen
- Helmut Alt, Freie Universität Berlin

- Katharina Klost
- Jakob Knudsen
- Sylvain Lazard
- Chih-Hung Liu
- Ioannis Mantas
- Till Miltzow
- Majid Mirzanezhad
- Fabrizio Montecchiani
- Martin Nöllenburg
- Peter Palfrader
- Alexander Pilz
- Nadja Scharf
- Don Sheehy
- Rodrigo Silveira
- Martin Suderland
- Javier Tejel
- Jorge Urrutia
- Jordi Vermeulen
- Manuel Wettstein
- Max Willert
- Lionov Wiratma
- Binhai Zhu

Supported by

Freie Universität Berlin

DFG Deutsche Forschungsgemeinschaft

>keylight/

Table of Contents

Rigidity and Deformation

Nina Amenta[1]

1 UC Davis

Abstract

Download a triangle-mesh model of a 3D bunny, cut a stick for every edge, and attach them together with a flexible joint at each vertex to re-create the model's one-skeleton. Would it stand up or collapse? Bet on "stand up" — Herman Gluck proved in 1975 that almost all such triangulated one-skeletons are rigid. There are a few examples of non-rigid polyhedra; that is, there is a motion under which the edge lengths remain fixed but the dihedral angles change.

What if, instead of fixing the edge lengths, we fixed the dihedrals? Are there motions which fix the dihedrals but allow the lengths to change? We show an analog of Gluck's theorem, that almost all polyhedra are "dihedral-rigid".

Who cares? Well, deformation is the opposite of rigidity. What can rigidity—and the examples of non-rigidity—tell us about how we can parameterize, measure and control the deformations of a mesh? Parameterizing deformations by edge length turns out to be a bad idea, but we demonstrate that there is reason to be much more hopeful about parameterizing meshes by their dihedrals.

This is work with Carlos Rojas.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Online Competitive Routing on Delaunay Triangulations and their Variants

Prosenjit Bose[1]

1 Carleton University

Abstract

A fundamental problem in computer science is that of finding a path in a graph. When the whole graph is available, standard path-finding algorithms can be applied such as Depth-First Search or Dijkstra's Algorithm. However, the problem of finding a path is more challenging in an online setting when at every step of the computation, only local information is available to the routing algorithm (such as the neighbourhood of the current vertex in the path). The difficulty is in deciding which edge to follow next in a path with only this local information. It is even more challenging to find a path with constant spanning ratio.

We will highlight different techniques for finding a short path in various types of Delaunay graphs in the online setting. We will highlight some of the difficulties involved with routing, review some of the currently best-known routing algorithms and mention a few open problems.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Geometric Issues for Self-Driving Cars

Raúl Rojas[1]

1 Freie Universität Berlin

Abstract

In my talk, I will give an overview of a new iteration of the architecture of the autonomous cars which have been developed at the Dahlem Center for Machine Learning and Robotics, Freie Universität Berlin. I will explain how we mix reactive with deliberative control. I will explain how we have experimented with geometry-based localization and the ideas we have for localization and driving under tough weather conditions. In one project we are investigating swarm behavior in traffic. At the end, I will present some ideas about the evolution of the commercial introduction of autonomous vehicles in the near future.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Maximal Two-Guard Walks in a Polygon

Franz Aurenhammer[1], Michael Steinkogler[2], and Rolf Klein[3]

1 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria,** auren@igi.tugraz.at
2 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria,** steinkogler@igi.tugraz.at
3 **Universität Bonn, Institut für Informatik, Bonn, Germany,** rolf.klein@uni-bonn.de

Abstract

Deciding two-guard walkability of an n-sided polygon is a well-solved problem. We study the related question of how far two guards can reach from a given source vertex, in the (more realistic) case that the polygon is not entirely walkable. There can be $\Theta(n)$ such maximal walks, and we show how to find all of them in $O(n \log n)$ time.

1 Introduction

We address a structural question on polygons, which is motivated by optimizing so-called triangulation axes [1], but is also interesting in its own right: How many adjacent 'ear triangles' can be cut off from a polygon W, starting from a given vertex s? Equivalent is the following question: How far can two guards reach when they are to walk on W's boundary, starting from s in different directions and keeping mutually visible?

Visibility problems of this kind have been studied already in the 1990s, where Icking and Klein [5] gave an $O(n \log n)$ time algorithm for deciding two-guard walkability of an n-sided polygon W, from a source vertex s to a target vertex t. A few years later, Tseng et al. [7] showed that one can find, within the same runtime, all vertex pairs (s, t) such that W is two-guard walkable from s to t. Their result was improved to optimal $O(n)$ time by Bhattacharya et al. [2]. The algorithm in [5] actually provides a walk for W in case of its existence but, on the other hand, only a negative message is returned in the (quite likely) case that the polygon is not entirely walkable.

The present note elaborates on 'how far' in the latter case a polygon W is two-guard walkable – a natural question that has not been considered in the literature to the best of our knowledge. Such *maximal walks* are not unique, in general, which complicates matters. We present a strategy that finds, in $O(n \log n)$ time, all possible maximal walks that initiate at a given source vertex s of W. A more detailed description of the results is given in [6].

2 Preliminaries

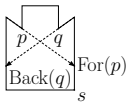
Throughout, we let W denote a simple polygon in the plane with n vertices, one of them being tagged as a source vertex, s. For two points x and y on the boundary, ∂W, of W, we write $x < y$ if x is reached before y when walking on ∂W from s in clockwise (CW) direction. For a vertex p of W, p^+ (repectively, p^-) is the CW successor (predecessor) vertex of p on ∂W. When p is a reflex vertex (that is, a vertex where the interior angle in W is greater than π), then the two 'ray shooting points' for p in W can be defined, namely, For(p) as the first intersection point with ∂W in the direction from p^- to p, and Back(p) as the first intersection point with ∂W in the direction from p^+ to p.

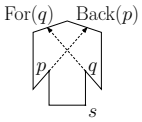
According to the aforementioned relation between walks and triangulations, we are only interested in *discrete* and *straight* walks. That is, the guards when moving on ∂W directly

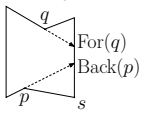
34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

'jump' from a vertex to the respective neighboring vertex (only one guard is allowed to move at a time), and they never backtrack. A *walk in* W is now defined as a diagonal (l, r) of W, $l < r$, such that the first guard can move CW from s to l, and the second guard can move CCW from s to r, while keeping visible to each other at each step. An obvious condition for W to be walkable till (l, r) is that the two boundary chains from s to l and to r, respectively (call them L and R), are *co-visible* in W. That is, each vertex on L is visible from some vertex on R, and vice versa.

To characterize walkability, we will need a few more concepts, first introduced in [5]. We say that at a pair (p, q) of its vertices, W forms a:

- *Forward deadlock* if
$$\text{Back}(q) < p < q < \text{For}(p).$$


- *Backward deadlock* if
$$p < (\text{For}(q), \text{Back}(p)) < q.$$


- *CW wegde* if $p < q$ and there exists no vertex x of W with
$$q < \text{For}(q) < x < \text{Back}(p).$$


(A *CCW wegde* is defined in a symmetric way.)

It is not hard to see that the two guards cannot pass beyond deadlocks and wedges without losing visibility. This will be made specific in Section 4. Moreover, in [5] it has been shown that these obstacles to walkability are indeed the only ones. By adapting their result to our setting we get:

▶ **Theorem 2.1.** *Let (l, r), $l < r$, be a diagonal of W, and denote with Q the polygon bounded by (l, r) and the chains L and R. Then (l, r) is a walk in W iff (1) L and R are co-visible in Q, (2) Q forms no forward and no backward deadlock, (3) Q forms no CW wedge on L, and no CCW wedge on R.*

3 Extremal walks and obstacles

A walk (l, r) in W is termed *maximal* if it cannot be extended by a single guard move. More precisely, neither (l^+, r) nor (l, r^-) is a walk in W. For finding maximal walks, we will apply Theorem 2.1, but we have to do so with care since conditions (1) to (3) refer to a (yet unknown) polygon Q, rather than to the input polygon W as in [5].

To this end, for (1) we observe that the chains L and R are co-visible in Q iff they are co-visible in W: The line segment $\overline{lr}$ lies entirely within W, so the part of ∂W different from ∂Q does not obstruct the view within Q.

Concerning (2), we notice that forward deadlocks formed by Q do not depend on the shape of $\partial W \setminus \partial Q$, and thus trivially are also forward deadlocks formed by W. By contrast, for a backward deadlock (p, q) formed by Q, the points $\text{For}(q)$ and $\text{Back}(p)$ in Q may not be the same as in W. (Namely, if at least one of them lies on $\overline{lr}$). But since these points are larger than p and smaller than q, (p, q) is also a backward deadlock in W.

No such property holds for the wedges in (3), however. A wedge (p, q) formed by Q is not neccessarily also formed by W: The segment $\overline{lr}$ can obstruct the view to vertices x on $\partial W \setminus \partial Q$ that prevent (p, q) from being a wedge in W. Fortunately though, such 'induced' wedges cannot occur once the co-visibility condition is satisfied; see [6].

In conclusion, it suffices to consider obstacles formed by W rather than by Q.

For maximal walks, obstacles with extremal positions are relevant (in case of the presence of obstacles at all, which we will assume in the sequel). A *minimal CW wedge* on the chain L is a wedge (p, q) on L where the vertex q is smallest possible. For a *minimal CCW wedge* (p, q) on R, in turn, the vertex p has to be largest possible. Such extremal wedges need not be unique. A representative can be found in $O(n \log n)$ time by adapting an algorithm in [7].

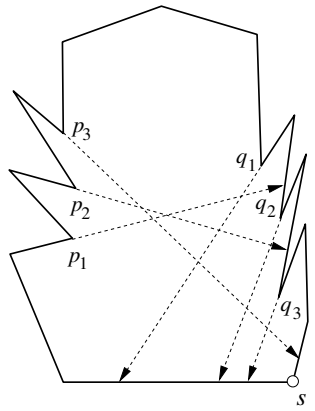
A deadlock (p, q) (either forward or backward) is called minimal if there is no other such deadlock (p', q') with $p' \leq p$ and $q' \geq q$. The *minimal backward deadlock* is unique, by the following property: If (p, q) and (p', q') are two backward deadlocks with $p < p'$ and $q < q'$, then (p, q') is a backward deadlock as well.

To find this minimal deadlock, we simply let p and q run through the reflex vertices of W, starting from s in CW and CCW direction, respecticely, until the deadlock inequalities for p as well as for q are fulfilled at the same time. This can be done in $O(n)$ time, if W has been preprocessed accordingly in $O(n \log n)$ time using ray shooting; see Chazelle et al. [3].

Minimal forward deadlocks, on the other hand, are not unique in general. This is one of the reasons why maximal walks need not be unique. In fact, W can contain $\Theta(n)$ minimal forward deadlocks (p_i, q_i); see the figure below for $i = 1, 2, 3$. The following algorithm reports all of them. The points on ∂W relevant for this task are the reflex vertices p of W plus their ray shooting points For(p). We assume their availability in cyclic order around W.

Algorithm MFD

Trace relevant points x in CCW order from s:
if $x =$ For(p) and $p < x$ **then**
  Put p to a CW sorted list F
else if x is a reflex vertex q **then**
  Search F for the smallest p with Back$(q) < p$
  if p exists and is unmarked **then**
    Mark p
    Report the forward deadlock (p, q)
  end if
end if
$x =$ next relevant point
Delete from F all vertices p with $p \geq x$



The proof of correctness of Algorithm MFD is omitted due to lack of space. The algorithm can be implemented in $O(n \log n)$ time. It scans $O(n)$ relevant points, each being processed in constant time apart from the actions on F, which take $O(n \log n)$ time in total when a balanced search tree for F is used.

4 Constraints from obstacles

Minimal wedges and deadlocks, and also the required co-visibility, give rise to constraints on l and r for a maximal walk (l, r) in the polygon W. We will discuss the constraints on l in some detail. The situation for r is symmetric.

We have to distinguish between *absolute* and *conditional* constraints. Among the former is the list below. The first two constraints stem from the co-visibility of L and R; see [5].

(1) For each reflex p with $p >$ For(p): $l \leq p$.

(2) For each reflex p with $p <$ Back(p): $l \leq$ Back(p).

(3) For the minimal CW wedge (p, q) on L: $l \leq q$.

(4) For the minimal backward deadlock (p, q): $l \leq p$.

The conditional constraints read as follows:

(I) For each p in (1): If $r > p$ then $l < p^-$.

(II) For each p in (2): If $r > \text{Back}(p)$ then $l \leq p$.

(III) For (p, q) in (3): If $r > q$ then $l < q$.

For convenience, we subsume the absolute constraints (1) - (4) into a single one, $l \leq x$ (where x is the smallest right-hand side value), and turn it into a conditional constraint:

(IV) If $r \geq s$ then $l \leq x$.

Finally, the minimal forward deadlocks lead to absolute constraints which deserve special attention. Whereas in the case of a backward deadlock (p, q), neither guard can walk beyond these vertices, we have the following observation for the avoidance of a forward deadlock (p, q): Only one of the bounds $l \leq p$ and $r \geq q$ needs to hold. Assume now that k minimal forward deadlocks $(p_1, q_1), \ldots, (p_k, q_k)$ exist, and let the p_i's be sorted in CW order.

▶ **Lemma 4.1.** *Each of the following $k + 1$ pairs of bounds for (l, r) avoids all minimal forward deadlocks:* $(p_1, s), (p_2, q_1), \ldots, (p_k, q_{k-1}), (s^-, q_k)$.

Proof. By minimality of the considered deadlocks, we know that the q_i's have to be sorted in CW order. So, for each index $i \geq 2$, the observation above tells us that the constraint $l \leq p_i$ avoids the deadlocks $(p_i, q_i), \ldots, (p_k, q_k)$, and the constraint $r \geq q_{i-1}$ avoids the remaining deadlocks $(p_1, q_1), \ldots, (p_{i-1}, q_{i-1})$. Moreover, the constraint $l \leq p_1$ suffices to avoid all k deadlocks, and $r \geq s$ is trivially fulfilled. The same is true for $r \geq q_k$ and $l \leq s^-$, respectively. ◀

In summary, there are $O(n)$ constraints in total, which can be found in $O(n \log n)$ time by the results in Section 3.

5 Computing all maximal walks

Section 4 tells us that the goal is to fulfill the constraints in (I) - (IV) simultaneously, though for each of the bounding pairs in Lemma 4.1 separately. This gives all possible maximal walks—granted the visibility of the reported vertex pairs. But let us come back to the issue of visibility later in this section.

For a *fixed* bounding pair (a, b), the constraint satisfaction problem can be transformed into the following standard form: For two variables l and r, with bounds a and b, respectively, there are two sets C_L and C_R of conditional constraints, of the form

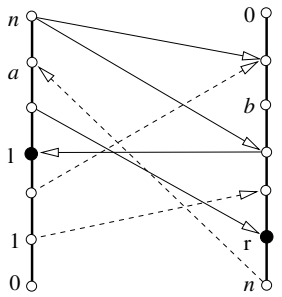
$$r \geq y_i \implies l \leq x_i \quad \text{and} \quad l \leq x_j \implies r \geq y_j$$

respectively, with all values in $\{0, 1, \ldots, n\}$. (That is, the vertices $w_0, w_1, \ldots, w_n$ of W, $w_0 = w_n = s$, are identified with their indices.) We want to compute the maximal pair (l, r) such that

$$l \leq a, \ r \geq b, \text{ and all } c \in C_L \cup C_R \text{ are fulfilled.}$$

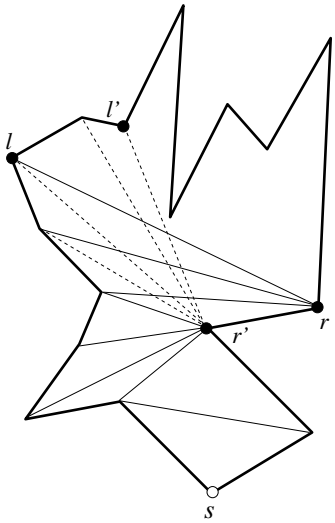
We say that $c_i \in C_L$ is *active* at a value r if $r \geq y_i$ holds. Similarly, $c_j \in C_R$ is *active* at l if we have $l \leq x_j$. The constraint fulfilling algorithm now simply alternates in scanning through the sorted sets C_L and C_R (in ascending order of y_i-values, and in descending order of x_j-values, respectively), and adjusts the values of l and r according to the constraints that get active. (Active/inactive constraints are indicated with full/dashed arrows below.)

Algorithm CFF(a, b, C_L, C_R)
$l = a$, $r = b$
repeat
$x = \min\{x_i \mid c_i \in C_L \text{ is active at } r\}$
$l = \min\{l, x\}$
$y = \max\{y_j \mid c_j \in C_R \text{ is active at } l\}$
$r = \max\{r, y\}$
until $r = y$ or $r = b$
Return the pair (l, r)



Suppose that a function VIS(l, r) is available which returns the first vertex $r' \geq r$ such that (l, r') is visible in the polygon W. (If r' does not exist then $n+1$ is returned.) We now present an algorithm that uses CFF and VIS as subroutines, and is capable of computing, in $O(n \log n)$ time, all maximal walks that exist in W. Let $P = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ be the given set of bounding pairs. We assume $a_1, \ldots, a_m$ (and thus $b_1, \ldots, b_m$) in increasing order. In the polygon below, (l, r) and (l', r') are the two possible maximal walks.

Algorithm MAXWALKS(P, C_L, C_R)
$l = a_m$, $r = b_1$
$r_{\text{rep}} = n + 1$
while $l \geq 0$ and $r < r_{\text{rep}}$ **do**
$(l, r) = \text{CFF}(l, r, C_L, C_R)$
$i = \min\{\lambda \mid a_\lambda \geq l\}$
$r_{\text{cand}} = \max\{b_i, r\}$
$r_{\text{vis}} = \text{VIS}(l, r_{\text{cand}})$
$y = \max\{\varrho \mid \text{all } c \in C_L \text{ active at } \varrho \text{ admit } l\}$
if $r_{\text{vis}} \leq \min\{n, y\}$ and $r_{\text{vis}} < r_{\text{rep}}$ **then**
Report (l, r_{vis})
$r_{\text{rep}} = r_{\text{vis}}$
end if
$l = l - 1$
end while



▶ **Lemma 5.1.** *Algorithm* MAXWALKS *is correct.*

Proof. The value of r changes only when Algorithm CFF is called, and thus r cannot decrease. The first call of CFF is with the bounding pair (a_m, b_1), and the subsequent calls are with (l, r) for $l < a_m$. As soon as we have $r > b_1$, some constraint in C_R is responsible for this. So putting the bound r for the next call means no additional restriction. This implies that, for all l, we have the equality $\text{CFF}(l, r, C_L, C_R) = \text{CFF}(l, b_1, C_L, C_R)$.

We now look at one iteration of the while loop, under the assumption that Algorithm MAXWALKS worked correctly so far. That is, all maximal walks (l', r') with $l' \geq l$ have been reported, and no other walks. Let l_{old} be the value of l before the iteration. Then $(l, r) = \text{CFF}(l_{\text{old}} - 1, b_1, C_L, C_R)$ holds by the former equality. So we have $(l, r) = \text{CFF}(l', b_1, C_L, C_R)$ for $l_{\text{old}} > l' > l$, implying that there is no walk (l', r') for these l'-values.

There also is no walk (l, r') with $r' < r_{\text{cand}}$, because the bounding pair (a_i, b_i) as well as the constraints in C_R need to be respected. Concerning r_{vis}, if $r_{\text{vis}} > n$ then no pair (l, r')

with $r' \geq r_{\text{cand}}$ is visible, and thus no such pair can be a walk. Further, if $r_{\text{vis}} > y$ then some constraint in C_L is active at r_{vis} but does not admit l, so (l, r_{vis}) is not a walk either. On the other hand, if $r_{\text{vis}} \leq \min\{n, y\}$ then (l, r_{vis}) is a walk, because the pair is visible and fulfills all the constraints. The pair gets reported unless $r_{\text{vis}} \geq r_{\text{rep}}$, in which case (l, r_{vis}) is not maximal because a larger pair has been reported already. ◀

CFF can be implemented such that the bounding pair of the last call is remembered. This way each constraint in $C_L \cup C_R$ is handled only once: If a call has been with (l, r), the next call will be with (l', r') where $l' < l$ (and thus $r' \geq r$). Thus only $O(n)$ time is spent in total for all calls to CFF from Algorithm MAXWALKS.

Computing the thresholds y can also be done in total $O(n)$ time. We remember the previous value of y, and scan down from this value as long as all active constraints of C_L are fulfilled by l. The first violating constraint then gives the new value for y.

The function VIS can be performed in logarithmic time, using the techniques in Guibas and Hershberger [4]. Clearly, the while loop is executed only $O(n)$ times, which gives a runtime of $O(n \log n)$ for this part, and thus for Algorithm MAXWALKS overall.

6 Concluding remarks

We conclude the paper with a few brief comments.

The polygon example above shows that maximal walks differ in (combinatorial) length, in general. It is also revealed that minimum forward deadlocks are not the only reason why maximal walks are not unique.

The number of maximal walks is trivially bounded by n, because no two of them can have the same l-vertex (or the same r-vertex).

Algorithm MAXWALKS provides each maximal walk in the form of a target pair (l, r), but not the way how the guards actually move on ∂W. Such a movement can be computed in $O(n)$ time, by applying the algorithm in [5] to the (already preprocessed) subpolygon of W defined by s and (l, r). Notice, however, that a fixed target pair (l, r) may leave to the guards different ways to perform the walk. Different ways to triangulate W from s to (l, r) then result, though the dual of any such triangulation has to be a path.

References

1 W. Aigner, F. Aurenhammer, and B. Jüttler. On triangulation axes of polygons. *Information Processing Letters* 115 (2015), 45-51.

2 B. Bhattacharya, A. Mukhopadhyay, and G. Narasimhan. Optimal algorithms for two-guard walkability of simple polygons. *Proc. Workshop on Algorithms and Data Structures, WADS 2001*. Springer Lecture Notes in Computer Science 2125, 438–449.

3 B. Chazelle, H. Edelsbrunner, M. Grigni, L.J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica* 12 (1994), 54–68.

4 L.J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Computer and System Sciences* 39 (1989), 126–152.

5 C. Icking and R. Klein. The two guards problem. *Int'l J. Computational Geometry & Applications* 2 (1992), 126–152.

6 M. Steinkogler. Maximal walks in polygons. Master Thesis, Institute for Theoretical Computer Science, University of Technology, Graz, Austria, 2017.

7 L.H. Tseng, P. Heffernan, and D.T. Lee. Two-guard walkability of simple polygons. *Int'l J. Computational Geometry & Applications* 8 (1998), 85–116.

Rectilinear Link Diameter and Radius in a Rectilinear Polygonal Domain* †

Man-Kwun Chiu[1,2], Elena Khramtcova[3], Aleksandar Markovic[4], Yoshio Okamoto[5,6], Aurélien Ooms[3], André van Renssen[1,2], and Marcel Roeloffzen[1,2]

1 **National Institute of Informatics, Tokyo, Japan**
{chiumk,andre,marcel}@nii.ac.jp
2 **JST, ERATO, Kawarabayashi Large Graph Project**
3 **Université libre de Bruxelles (ULB), Brussels, Belgium**
elena.khramtsova@gmail.com, aureooms@ulb.ac.be
4 **TU Eindhoven, Eindhoven, the Netherlands**
a.markovic@tue.nl
5 **University of Electro-Communications, Tokyo, Japan**
okamotoy@uec.ac.jp
6 **RIKEN Center for Advanced Intelligence Project, Tokyo, Japan**

Abstract

We study the computation of the diameter and radius under the rectilinear link distance within a rectilinear polygonal domain of n vertices and h holes. We introduce a *graph of oriented distances* to encode the distance between pairs of points of the domain. This helps us transform the problem so that we can search through the candidates more efficiently. Our algorithm computes both the diameter and the radius in $O(\min(n^\omega, n^2 + nh \log h + \chi^2))$ time, where $\omega < 2.373$ denotes the matrix multiplication exponent and $\chi \in \Omega(n) \cap O(n^2)$ is the number of edges of the graph of oriented distances. We also provide a faster algorithm for computing the diameter that runs in $O(n^2 \log n)$ time.

1 Introduction

Diameters and radii are popular characteristics of metric spaces. For a compact set S with a metric $d\colon S \times S \to \mathbb{R}^+$, its diameter is defined as $\mathrm{diam}(S) := \max_{p\in S} \max_{q\in S} d(p,q)$, and its radius is defined as $\mathrm{rad}(S) := \min_{p\in S} \max_{q\in S} d(p,q)$. The points that realize these distances are called the *diametral pair* and *center*, respectively. All of these terms are the natural extensions of the same concepts in a disk and give some interesting properties of the environment, such as the worst-case response time or ideal location of a serving facility.

Much research has been devoted towards finding efficient algorithms to compute the diameter and radius for various types of sets and metrics. In computational geometry, one of the most well-studied and natural metric spaces is a polygon in the plane. This paper focuses on the computation of the diameter and the radius of a rectilinear polygon, possibly with holes (i.e., a *rectilinear polygonal domain*) under the *rectilinear link distance*. Intuitively,

* MC, AvR and MR were supported by JST ERATO Grant Number JPMJER1201, Japan. EK was partially supported by the SNF Early Postdoc Mobility grant P2TIP2-168563, Switzerland, and F.R.S.-FNRS, Belgium. AM was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003. YO was partially supported by JSPS KAKENHI Grant Number 15K00009 and JST CREST Grant Number JPMJCR1402, and Kayamori Foundation of Informational Science Advancement. AO was supported by the Fund for Research Training in Industry and Agriculture (FRIA).

† Omitted proofs and further details can be found in the full version [4]

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

this metric measures the minimum number of links (segments) required in any rectilinear path connecting two points in the domain, where rectilinear indicates that we are restricted to horizontal and vertical segments only.

Many problems that are easy under the L_1 or Euclidean metric turn out to be more challenging under the link distance. For example, computing the shortest path between two points in a polygonal domain can be done in $O(n \log n)$ time for both Euclidean [8] and L_1 metrics [10, 11]. However, even approximating the same within a factor of $(2 - \epsilon)$ under the link distance is 3-SUM hard [12], and thus it is unlikely that a significantly subquadratic-time algorithm is possible.

Computing the diameter and radius is no exception: when considering simple polygons (i.e., polygons without holes) of n vertices, the diameter and center can be found in linear time for both Euclidean [1, 7] and L_1 metrics [2]. However, the best known algorithms for the link distance run in $O(n \log n)$ time [5, 16]. Lowering the running times or proving the impossibility of this is a longstanding open problem in the field. The only partial answer to this question was given by Nilsson and Schuierer [14, 15]; they showed that the diameter and center can be found in linear time when we are only allowed to use rectilinear paths.

If we consider polygons with holes, the difference becomes even bigger: no algorithm for computing the diameter and radius under the link distance is known, not even one that runs in exponential time. In comparison, polynomial-time algorithms are known both for diameter and radius under L_1 and Euclidean metrics.

1.1 Results

We introduce the *graph of oriented distances*, a graph that implicitly encodes the distance between regions of the domain. In Section 3 we use this graph to transform the problem: rather than searching pairwise distances in a list of potential candidates for diameter or center, we transform the problem into a rectangle intersection problem. Intuitively speaking, we cover the domain with several rectangles, and we find two pairs of rectangles that pairwise intersect (and satisfy other properties). In particular, once we have found the diametral pair, the four rectangles that satisfy the property can be used as a *witness*.

This transformation leads to an algorithm for computing both the rectilinear link diameter and radius of a rectilinear polygonal domain with n vertices and h holes. The algorithm is described in Section 4 and runs in $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication exponent [9]. Alternatively, we can also bound the running time in terms of the number χ of edges of the graph of oriented distances (χ will range from $\Omega(n)$ to $O(n^2)$ depending on P). With this parameter the running time becomes $O(n^2 + nh \log h + \chi^2)$. In Section 5 we use a different approach to obtain an $O(n^2 \log n)$ time algorithm to compute the diameter. All of the algorithms presented in the paper can be modified to return not only diameter or radius, but also the points that realize it (i.e., diametral pair and center).

1.2 Preliminaries

A *rectilinear simple polygon* (also called an orthogonal polygon) is a simple polygon that has horizontal and vertical edges only. A *rectilinear polygonal domain* P with h pairwise disjoint holes and n vertices is a connected and compact subset of $\mathbb{R}^2$ with h pairwise disjoint holes, in which the boundary of each hole is a simple closed rectilinear curve.

A *rectilinear path* π from $p \in P$ to $q \in P$ is a path from p to q that consists of vertical and horizontal segments, each contained in P, and such that along π each vertical segment is followed by a horizontal one and vice versa. Recall that P is a closed set, so π can traverse

the boundary of P (along the outer face and any of the h obstacles). We define the *link length* of such a path to be the number of segments composing it. The *rectilinear link distance* between points $p, q \in P$ is defined as the minimum link length of a rectilinear path from p to q, and denoted by $\ell_P(p, q)$. It is well known that in rectilinear polygonal domains there always exists a rectilinear polygonal path between any two points $p, q \in P$, and thus the distance is well defined. Once the distance is defined, the definitions of *rectilinear link diameter* $\mathrm{diam}(P)$ and *rectilinear link radius* $\mathrm{rad}(P)$ directly follow. For simplicity in the description, we assume that a pair of vertices do not share the same x- or y-coordinate unless they are connected by an edge.

2 Graph of Oriented Distances

For any domain P, we virtually shoot a ray left and right from any horizontal segment of the domain until it hits another segment of P, partitioning it into rectangles. We call this partition the *horizontal decomposition*, $\mathcal{H}(P)$. Similarly, if we shoot rays up and down from vertical segments, we get the *vertical decomposition*, $\mathcal{V}(P)$. Observe that both decompositions have linear size and can be computed in $O(n \log n)$ time with a plane sweep.

Given two rectangles $i, j \in \mathcal{H}(P) \cup \mathcal{V}(P)$, we use $i \sqcap j$ to denote the boolean operation which returns *true* if and only if (1) the rectangles i and j properly intersect (i.e. their intersection has non-zero area), and (2) one of i, j belongs to $\mathcal{H}(P)$, and the other to $\mathcal{V}(P)$.

▶ **Definition 2.1** (Graph of Oriented Distances)**.** Given a rectilinear polygonal domain P we define the undirected graph $\mathcal{G}(P) = (\mathcal{H}(P) \cup \mathcal{V}(P), \{\, (h, v) \in \mathcal{H}(P) \times \mathcal{V}(P) \colon\ h \sqcap v \,\})$.

In other words, vertices of $\mathcal{G}(P)$ correspond to rectangles of the horizontal and the vertical decompositions of P. We add an edge between two vertices if and only if the corresponding rectangles properly intersect. Note that this graph is bipartite, and has $O(n)$ vertices. From now on, we make a slight abuse of notation and identify a rectangle with its corresponding vertex (thus, we talk about the neighbors of a rectangle $i \in \mathcal{H}(P)$ in $\mathcal{G}(P)$, for example).

The name *Graph of Oriented Distances* is easily explained: consider a rectilinear path π between two points in P. Each horizontal edge of π is contained in a rectangle of $\mathcal{H}(P)$ and each vertical edge is contained in a rectangle of $\mathcal{V}(P)$. A bend in the path takes place in the intersection of the rectangles containing the two adjacent edges and corresponds to an edge of $\mathcal{G}(P)$. So every rectilinear path π has a corresponding path π' in $\mathcal{G}(P)$ and *vice versa*. Moreover, each bend of π is associated with an edge of π'.

▶ **Definition 2.2** (Oriented distance)**.** Given a rectilinear polygonal domain P, let i and j be two vertices of $\mathcal{G}(P)$, let $\Delta(i, j)$ to be the length of the shortest path between i and j in graph $\mathcal{G}(P)$ plus one. We also define $\Delta(i, i) = 1$.

We first list some useful properties of the oriented distance and then show the relationship between the oriented distance $\Delta(\cdot, \cdot)$ in $\mathcal{G}(P)$ and the link distance $\ell_P(\cdot, \cdot)$ in P.

▶ **Lemma 2.3.** *Let i, j, i', j' be any (not necessarily distinct) rectangles in $\mathcal{H}(P) \cup \mathcal{V}(P)$ such that $i \sqcap i'$, and $j \sqcap j'$. Then, the following hold: (a) $\Delta(i, j) = \Delta(j, i)$, (b) $\Delta(i', j) \in \{\, \Delta(i, j) - 1, \Delta(i, j) + 1 \,\}$, and (c) $\Delta(i', j') \in \{\, \Delta(i, j) - 2, \Delta(i, j), \Delta(i, j) + 2 \,\}$.*

▶ **Lemma 2.4.** *Let p and q be two points of the rectilinear polygonal domain P. If p and q lie in the same vertical or horizontal rectangle of $\mathcal{V}(P)$ or $\mathcal{H}(P)$ then $\ell_P(p, q) = 1$ (if p and q share a coordinate) or $\ell_P(p, q) = 2$ (if both x- and y-coordinates of p and q are distinct). Otherwise, let $i \in \mathcal{H}(P)$, $i' \in \mathcal{V}(P)$, $j \in \mathcal{H}(P)$ and $j' \in \mathcal{V}(P)$ be vertices of the graph of oriented distances such that $p \in i \cap i'$ and $q \in j \cap j'$. Then $\ell_P(p, q) = \min\{\, \Delta(i, j), \Delta(i, j'), \Delta(i', j), \Delta(i', j') \,\}$.*

Intuitively speaking, if we are given two disjoint rectangles $i, j \in \mathcal{H}(P)$, then $\Delta(i, j)$ denotes the minimum number of links needed to connect any two points $p \in i$ and $q \in j$ under the constraint that the first and the last segments of the path are horizontal. It follows that the link distance is the minimum of the four possible options. These $O(n^2)$ distances can be precomputed using algorithms by Mitchell *et al.* [13] in $O(n^2 + nh \log h)$ time or by Chan and Skrepetos [3] in $O(n^2 \log \log n)$.

3 Characterization via Boolean Formulas

Let $\hat{d} = \max_{i,j \in \mathcal{H}(P) \cup \mathcal{V}(P)} \Delta(i, j)$ be the largest distance between vertices of $\mathcal{G}(P)$. Similarly, we define $\hat{r} = \min_{i \in \mathcal{H}(P) \cup \mathcal{V}(P)} \max_{j \in \mathcal{H}(P) \cup \mathcal{V}(P)} \Delta(i, j)$. Note that these two values are the diameter and the radius of $\mathcal{G}(P)$ plus one. We use $\hat{d}$ and $\hat{r}$ to approximate the diameter $\mathrm{diam}(P)$ and radius $\mathrm{rad}(P)$ of a domain P under the rectilinear link distance. First, we relate the distance between two points $p, q \in P$ to the oriented distances between the rectangles that contain p and q. Specifically, from Lemma 2.4, we know that $\ell_P(p, q) = \min\{ \Delta(i, j), \Delta(i, j'), \Delta(i', j), \Delta(i', j') \}$, where $i, j \in \mathcal{H}(P)$ are the horizontal rectangles containing p and q, respectively, and $i', j' \in \mathcal{V}(P)$ are the vertical rectangles containing p and q. Similarly, we define $\hat{\ell}(p, q) = \max\{ \Delta(i, j), \Delta(i, j'), \Delta(i', j), \Delta(i', j') \}$.

▶ **Lemma 3.1.** *For any two points $p, q \in P$, let $i, j \in \mathcal{H}(P)$ and $i', j' \in \mathcal{V}(P)$ be the rectangles containing p and q, i.e., $p \in i \cap i'$ and $q \in j \cap j'$. Then, it holds that $\hat{\ell}(p, q) - 2 \leq \ell_P(p, q) \leq \hat{\ell}(p, q) - 1$.*

This relation allows us to express the rectilinear link diameter of a domain in terms of $\hat{d}$ and the radius in terms of $\hat{r}$.

▶ **Theorem 3.2.** *The rectilinear link diameter $\mathrm{diam}(P)$ of a rectilinear polygonal domain P satisfies $\mathrm{diam}(P) = \hat{d} - 1$ if and only if there exist $i, i', j, j' \in \mathcal{H}(P) \cup \mathcal{V}(P)$ with $i \sqcap i'$ and $j \sqcap j'$, such that $\Delta(i, j) = \hat{d}$ and $\Delta(i', j') = \hat{d}$. Otherwise, $\mathrm{diam}(P) = \hat{d} - 2$.*

Proof. First observe that for any pair of points $p, q \in P$ we have $\ell_P(p, q) \leq \hat{\ell}(p, q) - 1 \leq \hat{d} - 1$ by Lemma 3.1. Hence, the diameter of P is at most $\hat{d} - 1$. Similarly, by the definitions of $\hat{d}$ and $\hat{\ell}(\cdot, \cdot)$, there must be a pair of points $p, q \in P$ so that $\hat{\ell}(p, q) = \hat{d}$. Again by Lemma 3.1 it follows that $\mathrm{diam}(P) \geq \ell_P(p, q) \geq \hat{\ell}(p, q) - 2 = \hat{d} - 2$.

Next we show that the diameter is $\hat{d} - 1$ if and only if the above condition holds. If $\Delta(i, j) = \hat{d}$ and $\Delta(i', j') = \hat{d}$, then by Lemma 2.3 and the fact that neither $\Delta(i, j')$ nor $\Delta(i', j)$ can be larger than $\hat{d}$, we know that $\Delta(i, j') = \Delta(i', j) = \hat{d} - 1$. This implies that a pair of points $p \in i \cap i'$ and $q \in j \cap j'$ have $\ell_P(p, q) = \hat{d} - 1$. Thus, the diameter is $\hat{d} - 1$.

Now consider any pair p, q and the set of rectangles $i, j \in \mathcal{H}(P)$ and $i', j' \in \mathcal{V}(P)$ with $p \in i \cap i'$ and $q \in j \cap j'$. Recall that $\ell_P(p, q) = \min\{\Delta(i, j), \Delta(i, j'), \Delta(j', i), \Delta(i', j')\}$. By Lemma 2.3, $\Delta(i, j)$ and $\Delta(i', j')$ must differ by exactly one from $\Delta(i', j)$ and $\Delta(i, j')$. That implies that two distances may be $\hat{d} - 1$, but if the condition in the lemma is not satisfied, at most one can be $\hat{d}$ and the fourth must be $\hat{d} - 2$ or less. Therefore, if the condition is not satisfied for i, i', j, j', then the diameter is indeed $\hat{d} - 2$. ◀

▶ **Theorem 3.3.** *The rectilinear link radius $\mathrm{rad}(P)$ of a rectilinear polygonal domain P satisfies $\mathrm{rad}(P) = \hat{r} - 1$ if and only if for all $i, i' \in \mathcal{H}(P) \cup \mathcal{V}(P)$ with $i \sqcap i'$ there exist $j, j' \in \mathcal{H}(P) \cup \mathcal{V}(P)$ with $j \sqcap j'$ such that $\Delta(i, j) \geq \hat{r}$ and $\Delta(i', j') \geq \hat{r}$. Otherwise, $\mathrm{rad}(P) = \hat{r} - 2$.*

With the above characterization, we can naively compute the diameter and the radius by checking all $O(n^4)$ quadruples $(i, i', j, j') \in \mathcal{H}(P) \times \mathcal{V}(P) \times \mathcal{H}(P) \times \mathcal{V}(P)$. However, the approach can be improved by using $\mathcal{G}(P)$.

▶ **Theorem 3.4.** *The rectilinear link diameter* $\mathrm{diam}(P)$ *and radius* $\mathrm{rad}(P)$ *of a rectilinear polygonal domain* P *consisting of* n *vertices and* h *holes can be computed in* $O(n^2 + nh\log h + \chi^2)$ *time, where* χ *is the number of edges of* $\mathcal{G}(P)$ *(i.e., the number of pairs of intersecting rectangles of* $\mathcal{H}(P)$ *and* $\mathcal{V}(P)$*).*

4 Computation via Matrix Multiplication

In this section we provide an alternative method to compute the diameter. Although not described here, a similar approach can also be used to compute the radius. This method also uses the condition in Theorem 3.2, but instead exploits the behavior of matrix multiplication on (0,1)-matrices. Recall that, given two (0,1)-matrices A and B, their product is $(AB)_{i,j} = \sum_k (A_{i,k} \cdot B_{k,j}) = |\{\, k\colon\ A_{i,k} = 1 \wedge B_{k,j} = 1 \,\}|$.

We define the (0,1)-matrices I, D and M. Note that we slightly abuse our notation and use i, j to indicate both matrix indices and their corresponding rectangles.

$$I_{i,j} = \begin{cases} 1 & \text{if } i \sqcap j, \\ 0 & \text{otherwise.} \end{cases} \qquad D_{i,j} = \begin{cases} 1 & \text{if } \Delta(i,j) = \hat{d}, \\ 0 & \text{otherwise.} \end{cases} \qquad M_{i,j} = \begin{cases} 1 & \text{if } (ID)_{i,j} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, matrix I indicates for each pair of rectangles if they properly intersect and have different orientations, whereas D indicates which rectangles are at oriented distance $\hat{d}$ from each other. For any entry in their product we then have

$$(ID)_{j,i'} = |\{\, j'\colon\ (j \sqcap j') \wedge (\Delta(i', j') = \hat{d}) \,\}|.$$

The matrix M then records which entries in (ID) are non-zero and we get

$$(DM)_{i,i'} = |\{\, j\colon\ D_{i,j} = 1 \wedge M_{j,i'} = 1 \,\}| = |\{\, j\colon\ \Delta(i,j) = \hat{d} \wedge (\exists_{j'}\colon\ j \sqcap j' \wedge \Delta(i', j') = \hat{d})\}|.$$

So $(DM)_{i,i'} > 0$ if and only if there exists a pair j, j' for which $\Delta(i,j) = \hat{d}$, $j \sqcap j'$ and $\Delta(i', j') = \hat{d}$. By Theorem 3.2 if $(DM)_{i,i'} > 0$ and $I_{i,i'} = 1$ for any pair i, i', then the diameter is $\hat{d} - 1$ and otherwise it is $\hat{d} - 2$.

▶ **Theorem 4.1.** *The rectilinear link diameter* $\mathrm{diam}(P)$ *and radius* $\mathrm{rad}(P)$ *of a rectilinear polygonal domain* P *consisting of* n *vertices can be computed in* $O(n^\omega)$ *time.*

5 Computing the Diameter Faster

To test the condition of Theorem 3.2 we could simply iterate over each pair of rectangles i, j such that $\Delta(i,j) = \hat{d}$. For each such pair we could compute all pairs (i', j') such that $i \sqcap i'$ and $j \sqcap j'$ and test if $\Delta(i', j') = \hat{d}$. However, doing this naively may take $\Theta(n^4)$ time. Note however there are only $O(n^2)$ unique pairs (i', j') to test and regardless of which pair (i, j) was used to generate it, the diameter of P is $\hat{d} - 1$ if and only if at least one pair (i', j') has $\Delta(i', j') = \hat{d}$. We show how to more efficiently generate these pairs for the diameter. Unfortunately for the radius we must remember which pair (i, j) generates each pair (i', j') so this optimization doesn't work for the radius.

▶ **Theorem 5.1.** *The rectilinear link diameter* $\mathrm{diam}(P)$ *of a rectilinear polygonal domain* P *of* n *vertices can be computed in* $O(n^2 \log n)$ *time.*

Proof. *Sketch.* First, for each rectangle i, we find in $O(n \log n)$ time the set Q_i of rectangles at distance $\hat{d}$ from i. Then, using a ray-shooting data-structure by Giyora and Kaplan [6],

we compute the set R_i which contains all rectangles j' that are orthogonal and intersect a rectangle $j \in Q_i$. We then store in a list for each rectangle in $j' \in Q_i$ the segment i. After doing this for each rectangle again iterate over all rectangles and use j' to denote the current rectangle. For each j', let $L_{j'}$ denote the set of rectangles i with $j' \in R_i$, which we stored in a list. Then using the same ray-shooting data structure we can compute in $O(n \log n)$ time the set $M_{j'}$ of all rectangles that are orthogonal to and intersect a rectangle $i \in L_{j'}$. Then we simply check every pair (i', j') with $i' \in L_{j'}$ and if any such pair is at distance $\hat{d}$ we report that the diameter is $\hat{d} - 1$. Since we iterate over all $O(n)$ rectangles twice and spend $O(n \log n)$ time on each of them the total running time is $O(n^2 \log n)$. ◀

References

1 Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete Comput. Geom.*, 56(4):836–859, 2016.

2 Sang Won Bae, Matias Korman, Yoshio Okamoto, and Haitao Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. *Comp. Geom.: Theor. Appl.*, 48(6):495–505, 2015.

3 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. In *WADS*, pages 253–264, 2017.

4 Man-Kwun Chiu, Elena Khramtcova, Matias Korman, Aleksandar Markovic, Yoshio Okamoto, Aurélien Ooms, André van Renssen, and Marcel Roeloffzen. Rectilinear link diameter and radius in a rectilinear polygonal domain. *CoRR*, abs/1712.05538, 2017. URL: `https://arxiv.org/abs/1712.05538`, `arXiv:1712.05538`.

5 Hristo Djidjev, Andrzej Lingas, and Jörg-Rüdiger Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete Comput. Geom.*, 8:131–152, 1992.

6 Yoav Giyora and Haim Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Trans. Algorithm.*, 5(3):28:1–28:51, 2009.

7 John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM J. Comput.*, 26(6):1612–1634, 1997.

8 John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.

9 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, pages 296–303, 2014.

10 Joseph S. B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. In *CCCG*, 1989.

11 Joseph S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1):55–88, 1992.

12 Joseph S. B. Mitchell, Valentin Polishchuk, and Mikko Sysikaski. Minimum-link paths revisited. *Comp. Geom.: Theor. Appl.*, 47(6):651–667, 2014.

13 Joseph S. B. Mitchell, Valentin Polishchuk, Mikko Sysikaski, and Haitao Wang. An optimal algorithm for minimum-link rectilinear paths in triangulated rectilinear domains. In *ICALP*, pages 947–959, 2015.

14 Bengt J. Nilsson and Sven Schuierer. Computing the rectilinear link diameter of a polygon. In *CG*, pages 203–215, 1991.

15 Bengt J. Nilsson and Sven Schuierer. An optimal algorithm for the rectilinear link center of a rectilinear polygon. *Comp. Geom.: Theor. Appl.*, 6:169–194, 1996.

16 Subhash Suri. On some link distance problems in a simple polygon. *IEEE T. Robotic. Autom.*, 6(1):108–113, 1990.

Geometric clustering in normed planes *

Pedro Martín[1] and Diego Yáñez[2]

1 **Departamento de Matemáticas, Universidad de Extremadura**
`pjimenez@unex.es`
2 **Departamento de Matemáticas, Universidad de Extremadura**
`dyanez@unex.es`

Abstract

Given two sets of points A and B in a normed plane, we prove that there are two linearly separable sets A' and B' such that $\operatorname{diam}(A') \leq \operatorname{diam}(A)$, $\operatorname{diam}(B') \leq \operatorname{diam}(B)$, and $A' \cup B' = A \cup B$. As a consequence, some Euclidean clustering algorithms are adapted to normed planes.

1 Introduction and notation

We denote by $\mathbb{E}^2$ the Euclidean plane, and by $\mathbb{M}^2$ a *normed plane*, namely, $\mathbb{R}^2$ endowed with a norm $\|\cdot\|$. We call $B(x,r)$ the *ball with center* $x \in \mathbb{M}^2$ *and radius* $r > 0$, and $S(x,r)$ the *sphere* of $B(x,r)$. We use the usual abbreviations $\operatorname{diam}(A)$ and $\operatorname{conv}(A)$ for the *diameter* and the *convex hull* of a set A, $\overline{ab}$ for the *line segment* connecting two points $a, b \in \mathbb{M}^2$, and $\langle a, b\rangle$ for its affine hull.

We say that two sets of points in $\mathbb{M}^2$ are *linearly separable* (for short, *separable*) if there exists a line L such that each set is situated in a different closed half-plane defined by L. In Section 2, our Theorem 2.3 extends the following result ([4]) to any normed plane.

▶ **Theorem 1.1.** *Let A and B be two finite sets in $\mathbb{E}^2$. Then, there are two separable sets A' and B' such that $\operatorname{diam}(A') \leq \operatorname{diam}(A)$, $\operatorname{diam}(B') \leq \operatorname{diam}(B)$, and $A' \cup B' = A \cup B$.*

Given a set S of n points in the plane, a *cluster* is any non-empty subset of S, and a *k-clustering* is a set of k clusters such that each point of S belongs to some cluster. In Section 3, we apply Theorem 2.3 in order to solve some k-clustering problems in any normed plane.

2 Linear separability of clusters

In the rest of this section we work in $\mathbb{M}^2$ and our objective is to prove the statement of Theorem 1.1 in this context. Without loss of generality, we assume that $\operatorname{diam}(A) \geq \operatorname{diam}(B)$. Let us denote $\{u_1, u_2, \ldots, u_{2t}\}$ the clockwise sequence of points where the boundaries of $\operatorname{conv}(A)$ and $\operatorname{conv}(B)$ cross (Figure 1). $\operatorname{conv}(A) \setminus \operatorname{conv}(B)$ and $\operatorname{conv}(B) \setminus \operatorname{conv}(A)$ consist of two interlacing sequences of polygons $\{A_1, A_2, \ldots, A_t\}$ and $\{B_1, B_2, \ldots, B_t\}$ such that (for convenience, $u_{2t+1} := u_1$ and $A_{t+1} := A_1$): A_i touches B_i at u_{2i}; B_i touches A_{i+1} at u_{2i+1}; the vertices of any A_i belong either to $A \setminus B$ or to $\operatorname{conv}(A) \cap \operatorname{conv}(B)$; the vertices of any B_j belong either to $B \setminus A$ or to $\operatorname{conv}(A) \cap \operatorname{conv}(B)$. We say that (A_i, B_j) is a *bad pair* if $\operatorname{diam}(A_i \cup B_j) > \operatorname{diam}(A)$. In such a case, A_i is a *bad set* and B_j is its *bad partner*, and vice versa. If $\|a_i - b_j\| > \operatorname{diam}(A)$ for some $a_i \in A_i$ and $b_j \in B_j$, then both a_i and b_j are *bad points*, a_i is a *bad partner* of b_j (and vice versa), and the segment $\overline{a_i b_j}$ is a *bad segment*.

▶ **Lemma 2.1.** *Let (A_i, B_j) and $(A_{i'}, B_{j'})$ be two bad pairs such that $A_i \neq A_{i'}$ and $B_j \neq B_{j'}$. Let us choose $a_i \in A_i, b_j \in B_j, a_{i'} \in A_{i'}, b_{j'} \in B_{j'}$ such that $\overline{a_i b_j}$ and $\overline{a_{i'} b_{j'}}$ are bad segments.*

* Partially supported by Junta de Extremadura grant GR15055 (partially financed with FEDER)

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

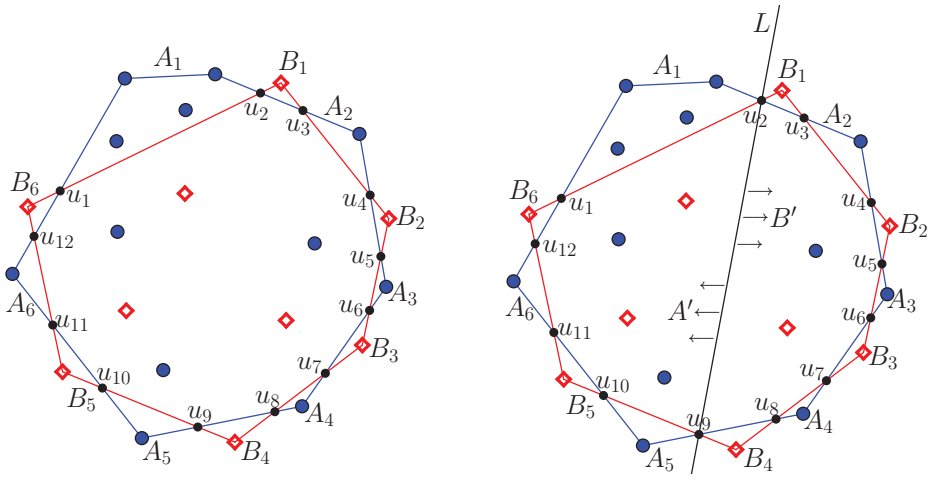


Figure 1 A (blue points) and B (red points) are not separable (left). $A \cup B$ can be split by L into new subsets A' and B' without increase of the Euclidean diameters (right).

If these bad segments do not cross, then $A_i, A_{i'}, B_{j'}, B_j$ (disregarding symmetric variations) is the sequence clockwise of these polygons and there is not any bad set from A between $B_{j'}$ and B_j.

Proof. Let us assume that $A_i, A_{i'}, B_j, B_{j'}, a_i, a_{i'}, b_j$, and $b_{j'}$ satisfy the conditions of the Lemma. All of them must be situated around $\text{conv}(A \cap B)$. If $\overline{a_i b_j} \cap \overline{a_{i'} b_{j'}} = \emptyset$, there are two cases (disregarding symmetric variations) for the relative positions of the polygons (and points):

Case 1: *$A_i, B_{j'}, A_{i'}, B_j$ is the clockwise sequence of the polygons.* Since the sum of the diagonals of the quadrangle $a_i, b_{j'}, a_{i'}, b_j$ is larger than the sum of two opposite sides, we get a contradiction:

$$\text{diam}(A) + \text{diam}(B) \geq \|a_i - a_{i'}\| + \|b_j - b_{j'}\| \geq \|a_i - b_j\| + \|a_{i'} - b_{j'}\| > 2\,\text{diam}(A).$$

Case 2: *$A_i, A_{i'}, B_{j'}, B_j$ is the clockwise sequence of the polygons.* Let us assume that there exists a bad point $a_m \in A_m$ for some m, such that $B_{j'}, A_m, B_j$ is the clockwise sequence. Let b_k be a bad partner of a_m for some k. The half-lines starting in a_m and connecting a_m with a_i and with $a_{i'}$, and the lines $\langle a_m, b_j \rangle$ and $\langle a_m, b_{j'} \rangle$, divide the plane into six zones (see Figure 2). If b_k is situated in the shaded zone in Figure 2, then $\|a_m - b_k\| \leq \text{diam}(B)$ and $\overline{a_m b_k}$ is not a bad segment. If b_k belongs to any other zone, it is possible to consider a quadrangle whose vertices are situated in clockwise order like in Case 1, and we get a contradiction. ◀

▶ Remark 1. In the Euclidean subcase, every two bad segments from disjoint bad pairs (A_i, B_j) and $(A_{i'}, B_{j'})$ cross ([4]). The property that the longest side of every obtuse triangle is opposite to the obtuse angle is used in the proof. Nevertheless, this property is not true for any normed plane, and there exist bad segments that do not cross.

Before splitting the sets A and B, we group all the adjacent bad subsets A_i from the cluster A. Thus, we define a group of bad subsets from A to be a maximal cyclic subsequence of bad subsets A_i. (Intervening subsets B_j of the other cluster must not be bad). The same

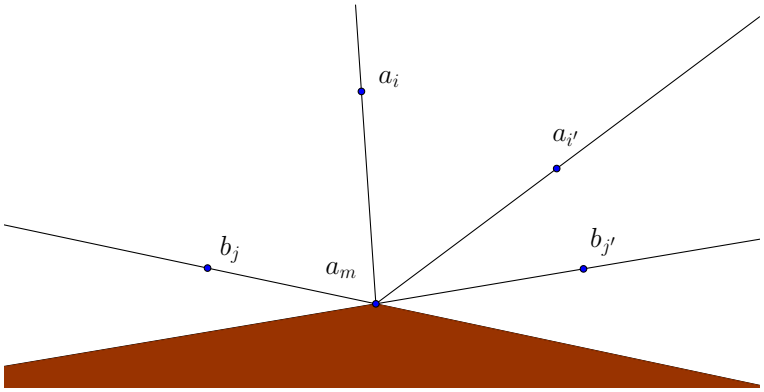


Figure 2 If (a_i, b_j) and $(a_{i'}, b_{j'})$ are bad partners, then the shaded zone cannot contain a bad partner of $a_m \in A_m$

is made with cluster B. These maximal cyclic groups are noted by $\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_p$ and $\bar{B}_1, \bar{B}_2, \ldots, \bar{B}_q$.

We say that $(\bar{A}_i, \bar{B}_j)$ is a *bad pair of groups* if there exists a bad segment from $\bar{A}_i$ to $\bar{B}_j$. Two pairs of sets (A_i, B_j) and $(A_{i'}, B_{j'})$ cross if there exist two (one from every pair) bad segments that cross. Similarly, $(\bar{A}_i, \bar{B}_j)$ and $(\bar{A}_{i'}, \bar{B}_{j'})$ cross if there exist two (one from every pair) bad segments that cross.

The structure of the rest of the section is similar to that presented by [4], but the proofs are different due to Remark 1. Some of the these proofs are omitted in this extended abstract.

▶ **Lemma 2.2.** *The following holds:*

1. *Let (A_i, B_j) and $(A_{i'}, B_{j'})$ be two bad pairs such that $A_i \neq A_{i'}$ and $B_j \neq B_{j'}$. If A_i and $A_{i'}$ belong to a group $\bar{A}_n$ for some n, then B_j and $B_{j'}$ belong to a group $\bar{B}_t$ for some t.*
2. *The number of maximal cyclic groups for A and for B is the same.*
3. *Let $(\bar{A}_i, \bar{B}_j)$ and $(\bar{A}_{i'}, \bar{B}_{j'})$ be two bad pairs of groups such that $\bar{A}_i \neq \bar{A}_{i'}$ and $\bar{B}_j \neq \bar{B}_{j'}$. Then $(\bar{A}_i, \bar{B}_j)$ and $(\bar{A}_{i'}, \bar{B}_{j'})$ cross.*

Proof. Statement 2 is a consequence of 1. In order to prove 1, let us assume that (A_i, B_j) and $(A_{i'}, B_{j'})$ are two disjoint bad pairs such that $A_i, A_{i'} \in \bar{A}_n$ for some n. If B_j and $B_{j'}$ belong to different groups, there is a bad pair (A_m, B_k) for some m and k such that A_m separates B_j from $B_{j'}$. (A_i, B_j) and $(A_{i'}, B_{j'})$ must cross (Lemma 2.1), and since A_i and $A_{i'}$ belong to the same group, only one of the pairs (not both) and (A_m, B_k) cross. For simplicity, let us assume that (A_i, B_j) and (A_m, B_k) cross. There exist $a_m \in A_m, b_k \in B_k, a_{i'} \in A_{i'}, b_{j'} \in B_{j'}$ that would be situated in an impossible clockwise sequence $a_m, b_k, a_{i'}, b_{j'}$ (similar to Case 1 in Lemma 2.1), and we get a contradiction.

Let us see Statement 3. Let $(\bar{A}_i, \bar{B}_j)$ and $(\bar{A}_{i'}, \bar{B}_{j'})$ be two bad pairs of groups such that $\bar{A}_i \neq \bar{A}_{i'}$ and $\bar{B}_j \neq \bar{B}_{j'}$. The clockwise order cannot be $\bar{A}_i, \bar{B}_{j'}, \bar{A}_{i'}, \bar{B}_j$ (due to the arguments used in Case 1 of Lemma 2.1); and neither $\bar{A}_i, \bar{A}_{i'}, \bar{B}_{j'}, \bar{B}_j$, because then $\bar{B}_{j'}$ and $\bar{B}_j$ cannot be separated by a bad polygon A_m (Lemma 2.1, Case 2). Therefore, the clockwise order must be $\bar{A}_i, \bar{A}_{i'}, \bar{B}_j, \bar{B}_{j'}$, and 3 holds.

◀

The groups from A and B are interlacing, and Statement 3 of Lemma 2.2 implies that there exist a complete matching among the groups, and the number of groups from each cluster has to be odd.

Let A_i be the last bad set of a group (in clockwise order), and let $B_{j'}$ be the last bad partner of A_i. Let B_j be the first bad set after A_i, and let $A_{i'}$ be the first bad partner of B_j. We choose the separating line L to go through the point u_{2j} before B_j and the point $u_{2j'+1}$ after $B_{j'}$ (see Figure 1). We define B' to be the points in $A \cup B$ lying on the same side of L as B_j and $B_{j'}$, and A' as the remaining points.

▶ **Theorem 2.3.** *Let A and B be two finite sets in $\mathbb{M}^2$. Then, there are two linearly separable sets A' and B' such that* $\operatorname{diam}(A') \leq \operatorname{diam}(A)$, $\operatorname{diam}(B') \leq \operatorname{diam}(B)$, $A' \cup B' = A \cup B$, *and*

$$\operatorname{perimeter}(\operatorname{conv}(A)) + \operatorname{perimeter}(\operatorname{conv}(B)) \geq \operatorname{perimeter}(\operatorname{conv}(A')) + \operatorname{perimeter}(\operatorname{conv}(B')).$$

Proof. We consider $A_i, B_j, A_{i'}, B_{j'}, L, A'$ and B' defined above. Since L cuts all bad pairs, $\operatorname{diam}(A') \leq \operatorname{diam}(A)$. In order to prove $\operatorname{diam}(B') \leq \operatorname{diam}(B)$, let us consider $a, b \in B'$. If $a, b \in B$ there is nothing to prove. In any other case, let us assume that $\|a - b\| > \operatorname{diam}(B)$. Let us choose $a_i \in A_i$, $b_j \in B_j$, $a_{i'} \in A_{i'}$, $b_{j'} \in B_{j'}$ such that $(a_i, b_{j'})$ and $(a_{i'}, b_j)$ are bad pairs. There are three possible cases.

Case 1: $a \in \operatorname{conv}(A) \setminus \operatorname{conv}(B)$ and $b \in \operatorname{conv}(B) \setminus \operatorname{conv}(A)$. The points $\{b_j, a, b, b_{j'}, a_{i'}, a_i\}$ are situated around $\operatorname{conv}(A) \cap \operatorname{conv}(B)$ and it is possible to consider a clockwise order. If $\{a, b\}$ is the clockwise order of these two points, we observe the quadrangle with vertices (clockwise) $\{b_j, a, b, a_{i'}\}$ and the following contradiction holds:

$$\operatorname{diam}(A) + \operatorname{diam}(B) \geq \|a - a_{i'}\| + \|b - b_j\| \geq \|b_j - a_{i'}\| + \|a - b\| > \operatorname{diam}(A) + \operatorname{diam}(B).$$

If the clockwise order is $\{b, a\}$, we obtain a similar contradiction on the quadrangle with vertices (clockwise order) $\{a_i, b, a, b_{j'}\}$.

Case 2: $a, b \in \operatorname{conv}(A) \setminus \operatorname{conv}(B)$. Case 1 implies that $\|b - b'\| \leq \operatorname{diam}(B)$ for every $b' \in (\operatorname{conv}(B) \setminus \operatorname{conv}(A)) \cap B'$. If $\{a, b\}$ is the clockwise order of these two vertices, applying the above arguments to the quadrangle $\{b_j, a, b, a_{i'}\}$:

$$\operatorname{diam}(A) + \operatorname{diam}(B) \geq \|a - a_{i'}\| + \|b - b_j\| \geq \|b_j - a_{i'}\| + \|a - b\| > \operatorname{diam}(A) + \operatorname{diam}(B),$$

which is again a contradiction. If the order is $\{b, a\}$, we use the quadrangle $\{b_j, b, a, a_{i'}\}$.

Case 3: $a \in \operatorname{conv}(A) \setminus \operatorname{conv}(B)$ and $b \in \operatorname{conv}(A) \cap \operatorname{conv}(B)$. Since the distance from a is maximized at some vertex of $\operatorname{conv}(A) \cap \operatorname{conv}(B) \cap \operatorname{conv}(B')$, we may assume that b is one of these vertices and apply an analysis similar to Case 1 or to Case 2.

The proof in [4] for the perimeter inequality is valid for $\mathbb{M}^2$. ◀

3 Some applications to clustering problems

From now on, S is a set of n points in $\mathbb{M}^2$. We assume that in our computation model an oracle answers the required questions about the unit ball of $\mathbb{M}^2$ (see Section 3.3 of [6]).

3.1 2-clustering problem: minimize the maximum diameter.

Given a metric, the 2-*clustering problem of minimizing the maximum diameter* asks about how to split S into two sets minimizing the maximum diameter. Avis [1] solves the problem in $\mathbb{E}^2$ looking for two separable sets with the following algorithm ($O(n^2 \log^2 n)$ time): sort the distances d_i between the points of S into increasing order ($O(n^2 \log n)$ time); locate the minimum d_i that admits a *stabbing line*[1] (using [5] for the stabbing line) by a binary search. We obtain the following as a consequence of Theorem 2.3.

[1] A *stabbing line* for a set of segments is a line that intersects every segment of the set.

▶ **Corollary 3.1.** *Given a set of n points in $\mathbb{M}^2$, the 2-clustering problem of minimizing the maximum diameter can be solved in $O(n^2 \log^2 n)$ time using the algorithm presented by Avis.*

The similar approach of Asano et al. ([2], that reduces the cost of Avis' approach to $O(n \log n)$ time using a maximum spanning tree) could be used as well, but as far as we know, an efficient method to build a maximum spanning tree for any normed plane is not known.

3.2 2-clustering problem: constraints over the diameters

Given two fixed numbers $d_1 \geq d_2 > 0$, Hershberger and Suri ([8]) solve in $O(n \log n)$ time the problem of dividing S into two sets S_1 and S_2, such that $\text{diam}(S_1) \leq d_1$ and $\text{diam}(S_2) \leq d_2$ in $\mathbb{E}^2$. They use the fact that if $\|a - b\| \geq d_1$, then $B(a, d_2) \cap B(b, d_1)$ can always be split into two subsets whose diameters are at most d_1 and d_2, respectively. Nevertheless, the following example shows that this cannot be extended to $\mathbb{M}^2$. Let us consider $a = (0, 0)$, $b = (-9.81, 6.24)$, and the strictly convex norm whose unit sphere is bounded by the two arcs of circles with center at $(0, 10)$ and in $(0, -10)$, respectively, and radius $5\sqrt{13}$ (see Figure 3). Let $\{r = (-9.39, r_2), \quad s = (-8.24, s_2)\} \in S(a, 1)$ and $\{p, q\} = S(a, 1) \cap S(b, 1.1)$, such that $r_2 > 0$, $s_2 > 0$, and $\{p, r, s, q\}$ is the clockwise order on $S(a, 1)$. Then, $\|a - b\| \geq 1.1$, $\min\{\|s - p\|, \|r - q\|, \|p - q\|\} > 1.1$ and $\min\{\|r - p\|, \|s - q\|\} > 1$, and $S = \{p, q, r, s\} \in B(a, 1) \cap B(b, 1.1)$ cannot be divided into two subsets whose diameters are at most 1.1 and 1, respectively.

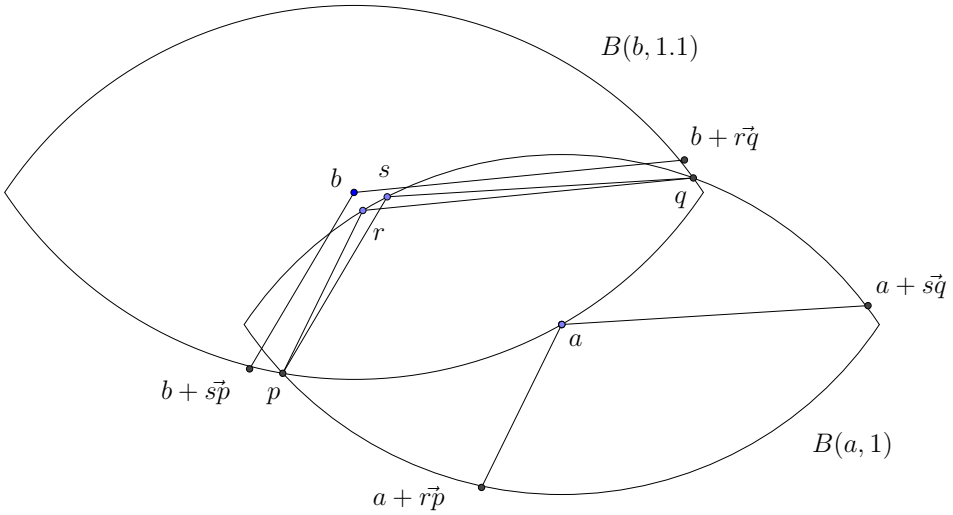


Figure 3 $S = \{p, q, r, s\}$ cannot be divided into two subsets with diameters less than or equal to 1.1 and 1, respectively.

Theorem 2.3 can help to solve this problem in any normed plane as follows. Build the graph (S, E_{d_1}) with the points of S and the set of edges E_{d_1} connecting two points of S at distance more than d_1 (in $O(n^2 \log n)$ time). Check if E_{d_1} has a stabbing line (in $O(n \log n)$ time with the algorithm presented in [5]). If the stabbing line does not exist, there is no solution (Theorem 2.3). If some stabbing lines exist, check if one of them split S into two subsets with the required diameters.

3.3 k-clustering problems

Let us consider the *k-clustering problem of minimizing $\mathcal{F}$ to the diameters* (equivalently, *to the radii*), where $\mathcal{F}$ is a monotone increasing function $\mathcal{F} : \mathbb{R}^k \to \mathbb{R}$ that is applied to the diameters (equivalently, to the radii) of the clusters. For instance, $\mathcal{F}$ can be the *maximum*, the *sum*, or the *sum of squares* of the diameters (or the radii). Capoyleas, Rote and Woeginger (see Lemma 8 and Theorem 9 in [4] for details) design an algorithm that solves these geometric k-clustering problems in polynomial time. Using Theorem 2.3 and a result of Banasiak [3] describing precisely the intersection of two balls, we prove the following statements. The proofs are omitted in this extended abstract.

▶ **Theorem 3.2.** *Let S be a set of n points in $\mathbb{M}^2$. Consider the k-clustering problem of minimizing a monotone increasing function $\mathcal{F} : \mathbb{R}^k \to \mathbb{R}$ that is applied to the diameters or to the radii of k subsets of S. Then there is an optimal k-clustering such that each pair of clusters is linearly separable. A solution can be obtained by the algorithm presented by Capoyleas-Rote-Woeginger, and it takes polynomial time for the case of the diameters.*

3.4 3-clustering problems: minimize the maximum diameter

▶ **Theorem 3.3.** *Given a set of n points in $\mathbb{M}^2$ and $d > 0$, we can determine in $O(n^3 \log^2 n)$ time whether there is a partition of S into sets A, B, C with diameters at most d, and construct in $O(n^3 \log^3 n)$ time a 3-partition of S such that the largest of the three diameters is as small as possible.*

Proof. (Scheme) A specific approach by Hagauer and Rote proves this result in $\mathbb{E}^2$. For the first statement, the authors use some lemmas (from Lemma 3 to Lemma 6 in [7]) and Theorem 1.1. Theorem 2.3 extends Theorem 1.1, and we prove results similar to the rest of lemmas in [7] for any normed plane using the notion of Birkhoff orthogonality instead of the Euclidean one. Regarding the complexity of the algorithm, we can justify that the data structure introduced by Hershberger and Suri ([8]) is usable in the same way that in $\mathbb{E}^2$. Finally, a binary search on the $\binom{n}{2}$ distances occurring in S solves the optimization problem. ◀

References

1 D. Avis. Diameter partitioning. *Discrete Comput. Geom.*, 1:265-276, 1986.
2 T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. *Proc. 4th ACM Symposium on Computational Geometry*, 252-257, 1988.
3 J. Banasiak. Some contributions to the geometry of normed linear spaces. *Math. Nachr.*, 139:175-184, 1988.
4 V. Capoyleas, G. Rote, and G. Woeginger. Geometric clustering. *J. Algorithms*, 12:341-356, 1991.
5 H. Edelsbrunner, H.A. Mauer, F.P. Preparata, A.L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, 22:274-281, 1982.
6 P. Gritzmann, and V.L. Klee. On the complexity of some basic problems in computational convexity: I. Containment problems. *Discrete Math.*, 136:129-174, 1994.
7 J. Hagauer, and G. Rote. Three-clustering of points in the plane. *Comput. Geom.*, 8:87-95, 1997.
8 J. Hershberger, and S. Suri. Finding tailored partitions. *J. Algorithms*, 12:431–463, 1991.

Coxeter triangulations have good quality

Aruni Choudhary[1], Siargey Kachanovich[2], and Mathijs Wintraecken[2]

1 **Max Planck Institute for Informatics, Saarland Informatics Campus Saarbrücken, Germany**
2 **Université Côte d'Azur, Inria, France**

Abstract

Coxeter triangulations are triangulations of Euclidean space based on a single simplex. By this we mean that given an individual simplex we can recover the entire triangulation of Euclidean space by inductively reflecting in the faces of the simplex. In this paper we establish that the quality of the simplices in all Coxeter triangulations is $O(1/\sqrt{d})$ of the quality of regular simplex. We further investigate the Delaunay property (and an extension thereof) for these triangulations. In particular, one family of Coxeter triangulations achieves the protection $O(1/d^2)$. We conjecture that both bounds are optimal for triangulations in Euclidean space.

1 Introduction

1.1 Motivation and related work

Well shaped simplices are of importance for various fields of application such as finite element methods and manifold meshing. Poorly-shaped simplices may induce various problems in finite element method, such as large discretization errors or ill-conditioned stiffness matrices. A simplex is well shaped if its quality is good, which can be expressed in terms of various *quality measures*. Some examples of quality measures are: the ratio between minimal height and maximal edge length ratio called *thickness*, the ratio between volume and a power of the maximal edge length called *fatness*, and the *inradius-circumradius ratio*. Bounds on *dihedral angles* can also be included in the list of quality measures. We stress that there are many other quality measures in use and authors often find useful to introduce measures that are specific to whatever problem they study. Finding triangulations, even in Euclidean space, of which all simplices have good quality is a non-trivial exercise in arbitrary dimension.

In this paper we shall discuss Coxeter triangulations.

▶ **Definition 1.1.** A monohedral[1] triangulation is called *Coxeter triangulation* if all its d-simplices can be obtained by consecutive reflections through facets of the d-simplices in the triangulation.

There are four families of Coxeter triangulations and five exceptional ones. All three two-dimensional Coxeter triangulations and the simplices of the three three-dimensional Coxeter triangulations are illustrated in Figure 1. For an extended introduction we refer to the pioneering paper on reflection groups by Coxeter [10] and the classical book on Lie groups and algebras by Bourbaki [5]. Another classical reference book is "Sphere packings, Lattices, and Groups" by Conway and Sloane [9].

To our knowledge, these are the triangulations with the best quality in arbitrary dimension. In particular, all dihedral angles of simplices in Coxeter triangulations are 45°, 60° or 90°, with the exception of the so-called $\tilde{G}_2$ triangulation of the plane where we also can find an

[1] A triangulation of $\mathbb{R}^d$ is called *monohedral* if all its d-simplices are congruent.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

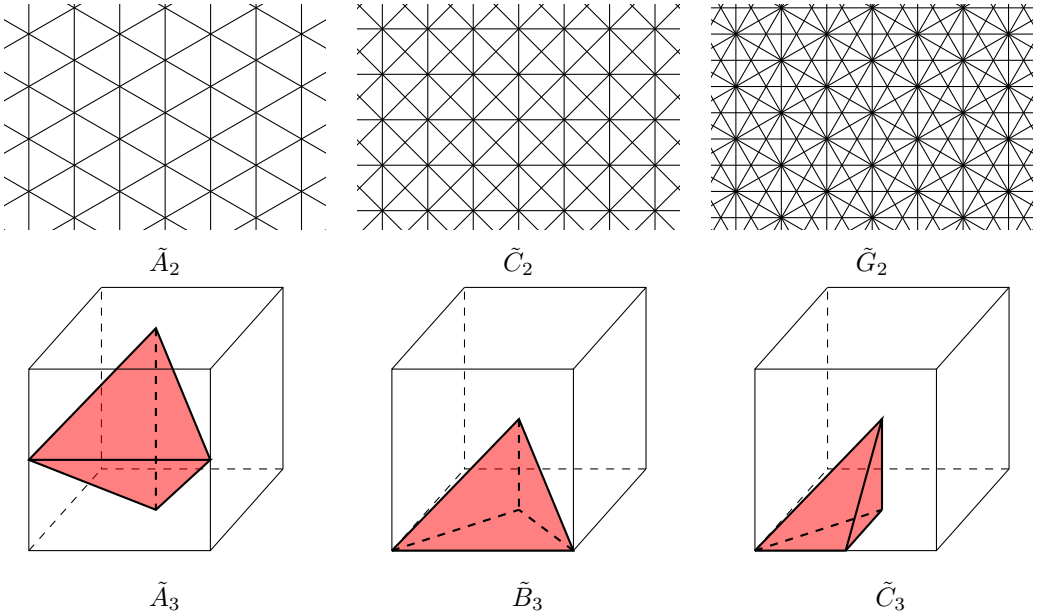


Figure 1 Above: Coxeter triangulations in $\mathbb{R}^2$. Below: simplices of Coxeter triangulations in $\mathbb{R}^3$ represented as a portion of a cube.

angle of 30°. This is a clear sign of the exceptional quality of the simplices involved. Our goal is to exhibit the extraordinary properties of Coxeter triangulations and promote their use in the Computational Geometry community.

The notion of Coxeter triangulations was introduced to the computational geometry community by Dobkin, Wilks, Levy and Thurston in [11], where they tackled the problem of contour-tracing in $\mathbb{R}^d$. The choice of Coxeter triangulations was motivated by the following requirements:

- It should be easy to find the simplex that shares a facet with a given simplex.
- It should be possible to label the vertices of all the simplices at the same time with indices $0, \ldots, d$, in such a way that each of the $d+1$ vertices of a d-simplex has a different label.
- The triangulations should be monohedral, meaning that all simplices are congruent.
- All simplices should be isotropic, meaning that they should be roughly the same in all directions.

Coxeter triangulations exactly fit these requirements. After comparing the inradius-circumradius ratio of the simplices in the triangulations Dobkin et al. chose the $\tilde{A}_d$ Coxeter triangulation as the underlying triangulation for their contour-tracing algorithm.

The same $\tilde{A}_d$ Coxeter triangulation appeared in the works by Adams, Baek and Davis [1] and Choudhary, Kerber and Raghvendra [7], among others.

The three-dimensional Coxeter triangulation $\tilde{A}_3$ has attracted attention in the 3D mesh generation community for the high-quality of its simplices. The vertex set of this triangulation is also known as the *body-centred cubic lattice*, or bcc lattice, and its tetrahedron is sometimes referred as *Sommerville's type II tetrahedron* or simply bcc-tetrahedron. This tetrahedron has been shown to be the best-conditioned space-filling tetrahedron out of all space-filling tetrahedra used in the 3D mesh community by a number of conditioning measures.

Apart from quality, we are also interested in the stronger requirement of *protection* [2], which is specific to Delaunay triangulations. It has been proven that protection guarantees

good quality [2]. Some algorithms were introduced for the construction of a protected set, such as the perturbation-based algorithms in [3] and [4]. Both of these algorithms take a general ε-net in $\mathbb{R}^d$ as input and output a δ-protected net with δ of the order just $\Omega(2^{-d^2}\varepsilon)$. The d-dimensional Coxeter triangulation $\tilde{A}_d$ provides us another extremity. As we will see in the following, this highly-structured triangulation is Delaunay with protection $O(\frac{1}{d^2}\varepsilon)$. This protection value is the greatest in a general d-dimensional Delaunay triangulation we know.

The Coxeter triangulations we study are intricately linked with root systems and root lattices. Delaunay triangulations of the root lattices have been studied by Conway and Sloane [9] and Moody and Patera [12]. These triangulations are different from the ones we study: the vertex sets we use are not necessarily lattices (see Theorem 4.1).

1.2 Contribution

In this paper we give explicit expressions of a number of quality measures of Coxeter triangulations for all dimensions, presented in Section 4. This is an extension of the work by Dobkin et al. [11] who presented a table of the values of the inradius-circumradius ratio for the Coxeter triangulations up to dimension 8. We also provide explicit measures of the corresponding simplices in the full version of the current paper [8, Appendix B], allowing the reader to compute quality measures other than the ones listed.

In Section 2, we state the theorem of optimality of the regular d-simplex for each of the chosen quality measures. This theorem justifies the definition of the normalized versions of these quality measures.

In Section 3, we established a criterion to identify if any given monohedral triangulation is Delaunay.

The proofs of the statements can be found in the full version [8], as well as extra introductory material.

1.3 Future work

The simplex qualities, defined in Definition 2.1, of the four families of Coxeter triangulations behave as $O(1/\sqrt{d})$ in terms of dimension. We conjecture that this quality is optimal for a general space-filling triangulation in $\mathbb{R}^d$. In addition, the d-dimensional Coxeter triangulation $\tilde{A}_d$ has the relative Delaunay protection $O(1/d^2)$. We further conjecture that it is optimal for a general space-filling triangulation in $\mathbb{R}^d$. These conjectures are motivated by the extraordinary lower and upper bounds on the dihedral angles of simplices in Coxeter triangulations; they are precisely 45°, 60° or 90° for the four families. Moreover the circumcentres of the simplices of $\tilde{A}_d$ lie very far inside the simplices.

2 Quality definitions

The quality measures we are interested in, we call *aspect ratio*, *fatness*, *thickness* and *radius ratio*. Their formal definitions are as follows:

▶ **Definition 2.1.** Let $h(\sigma)$ denote the minimal height, $r(\sigma)$ the inradius, $R(\sigma)$ the circumradius, $vol(\sigma)$ the volume and $L(\sigma)$ the maximal edge length of a given d-simplex σ.

- The *aspect ratio* of σ is the ratio of its minimal height to the diameter of its circumscribed ball: $\alpha(\sigma) = \frac{h(\sigma)}{2R(\sigma)}$.
- The *fatness* of σ is the ratio of its volume to its maximal edge length taken to the power d: $\Theta(\sigma) = \frac{vol(\sigma)}{L(\sigma)^d}$.

- The *thickness* of σ is the ratio of its minimal height to its maximal edge length: $\theta(\sigma) = \frac{h(\sigma)}{L(\sigma)}$.
- The *radius ratio* of σ is the ratio of its inradius to its circumradius: $\rho(\sigma) = \frac{r(\sigma)}{R(\sigma)}$.

To be able to compare the presented quality measures between themselves, we will normalize them by their respective maximum value. As we show, all of these quality measures are maximized by regular simplices.

▶ **Theorem 2.2.** *Out of all d-dimensional simplices, the regular d-simplex has the highest aspect ratio, fatness, thickness and radius ratio.*

For a quality measure κ we will define the normalized quality measure $\hat{\kappa}$, such that for each d-simplex σ, $\hat{\kappa}(\sigma) = \frac{\kappa(\sigma)}{\kappa(\Delta)}$, where Δ is the regular d-simplex. Theorem 2.2 ensure that the quality measures $\hat{\rho}$, $\hat{\alpha}$, $\hat{\theta}$ and $\hat{\Theta}$ take their values in $[0, 1]$ surjectively.

3 Delaunay criterion for Coxeter triangulations

Many of the provably good mesh generation algorithms are based on Delaunay triangulations [6]. This motivated us to investigate if Coxeter triangulations have the Delaunay property. We established the following criterion, extending the work by Rajan [13] on triangulations consisting of self-centered simplices.

▶ **Definition 3.1.** A simplex is called *self-centred* if it contains its circumcentre inside or on the boundary.

▶ **Theorem 3.2.** *A Coxeter triangulation is Delaunay if and only if its simplices are self-centred.*

Because some of the triangulations that interest us here are Delaunay, we will also look at their *protection* value.

▶ **Definition 3.3.** The *protection* of a d-simplex σ in a Delaunay triangulation on a point set P is the minimal distance of points in $P \setminus \sigma$ to the circumscribed ball of σ:

$$\delta(\sigma) = \inf_{p \in P \setminus \sigma} d(p, B(\sigma)), \text{ where } B(\sigma) \text{ is the circumscribed ball of } \sigma.$$

The *protection* δ of a Delaunay triangulation $\mathcal{T}$ is the infimum over the d-simplices of the triangulation: $\delta = \inf_{\sigma \in \mathcal{T}} \delta(\sigma)$. A triangulation with a positive protection is called *protected*.

We define the *relative protection* $\hat{\delta}(\sigma)$ of a given d-simplex σ to be the ratio of the protection to its circumscribed radius: $\hat{\delta}(\sigma) = \frac{\delta(\sigma)}{R(\sigma)}$.

The *relative protection* $\hat{\delta}$ of a Delaunay triangulation $\mathcal{T}$ is the infimum over the d-simplices of the triangulation: $\hat{\delta} = \inf_{\sigma \in \mathcal{T}} \hat{\delta}(\sigma)$. We can determine if a Coxeter triangulation is not protected with the help of the following theorem.

▶ **Theorem 3.4.** *1. A Delaunay triangulation of $\mathbb{R}^d$ where a simplex with maximal circumradius contains the circumcentre on its boundary is not protected.*

2. If a simplex of a Coxeter triangulation contains the circumcentre on its boundary, then the triangulation is non-protected Delaunay.

3. If a simplex of a Coxeter triangulation contains the circumcentre strictly inside, then the triangulation is Delaunay with a non-zero protection.

4 Main result

In this section we present a table with explicit expressions of quality measures of Coxeter triangulations. In addition to that we also identify which Coxeter triangulations are Delaunay and give their protection values. Finally, we identify which Coxeter triangulations have vertex sets with lattice structure.

▶ **Theorem 4.1.** *The normalized fatness, aspect ratio, thickness and radius ratio of simplices in Coxeter triangulations, as well as Delaunay property are:*

	Fatness $\hat{\Theta}^{1/d}$	*Aspect Ratio* $\hat{\alpha}$	*Thickness* $\hat{\theta}$	*Radius Ratio* $\hat{\rho}$	*Delaunay?*
$\tilde{A}_d$, d *odd*	$\frac{2^{3/2}}{\left(\sqrt{d+1}\right)^{1+2/d}}$	$\sqrt{\frac{6d}{(d+1)(d+2)}}$	$\frac{2\sqrt{d}}{d+1}$	$\sqrt{\frac{6d}{(d+1)(d+2)}}$	✓
$\tilde{A}_d$, d *even*	$\frac{2^{3/2}(\sqrt{d+1})^{1-2/d}}{\sqrt{d(d+2)}}$		$\frac{2}{\sqrt{d+2}}$		
$\tilde{B}_d$	$\frac{2^{1/2+1/d}}{\sqrt{d}\left(\sqrt{d+1}\right)^{1/d}}$	$\frac{d\sqrt{2}}{(d+1)\sqrt{d+2}}$	$\frac{1}{\sqrt{d+1}}$	$\frac{2d}{\sqrt{d+2}(1+(d-1)\sqrt{2})}$	✗
$\tilde{C}_d$	$\frac{\sqrt{2}}{\sqrt{d}\left(\sqrt{d+1}\right)^{1/d}}$	$\frac{\sqrt{2d}}{d+1}$	$\frac{1}{\sqrt{d+1}}$	$\frac{2\sqrt{d}}{2+(d-1)\sqrt{2}}$	✓
$\tilde{D}_d$	$\frac{2^{1/2+2/d}}{\sqrt{d}\left(\sqrt{d+1}\right)^{1/d}}$	$\frac{d\sqrt{2}}{(d+1)\sqrt{d+4}}$	$\frac{1}{\sqrt{d+1}}$	$\frac{d\sqrt{2}}{(d-1)\sqrt{d+4}}$	✗
$\tilde{E}_6$	$\sqrt[12]{\frac{64}{137781}}$	$\frac{2}{7}$	$\frac{1}{\sqrt{14}}$	$\frac{1}{2}$	✗
$\tilde{E}_7$	$\sqrt[14]{\frac{1}{177147}}$	$\frac{7\sqrt{13}}{104}$	$\frac{\sqrt{21}}{24}$	$\frac{14\sqrt{13}}{117}$	✗
$\tilde{E}_8$	$\sqrt[8]{\frac{1}{3240}}$	$\frac{8\sqrt{19}}{171}$	$\frac{2\sqrt{19}}{57}$	$\frac{8\sqrt{19}}{95}$	✗
$\tilde{F}_4$	$\sqrt[8]{\frac{1}{405}}$	$\frac{4\sqrt{2}}{15}$	$\frac{2\sqrt{5}}{15}$	$\frac{4\sqrt{2}}{3(2+\sqrt{2})}$	✓
$\tilde{G}_2$	$\frac{\sqrt{2}}{2}$	$\frac{1}{\sqrt{3}}$	$\frac{1}{2}$	$\frac{2}{1+\sqrt{3}}$	✓

Out of them, only $\tilde{A}$ family triangulations have a non-zero relative protection value equal to:

$$\hat{\delta} = \frac{\sqrt{d^2+2d+24} - \sqrt{d^2+2d}}{\sqrt{d^2+2d}} \sim \frac{12}{d^2}$$

Only $\tilde{A}$ family, $\tilde{C}$ family and $\tilde{D}_4$ triangulations have vertex sets with lattice structure.

For the proof, refer to the full version [8]. The corresponding quality measures for the regular d-simplex Δ (which does not correspond to a triangulation in general) are:

	Fatness Θ	Aspect Ratio α	Thickness θ	Radius Ratio ρ
Δ	$\frac{1}{d!}\sqrt{\frac{d+1}{2^d}}$	$\frac{d+1}{2d}$	$\sqrt{\frac{d+1}{2d}}$	$\frac{1}{d}$

All simplex quality measures in the table above are normalized with respect to the regular simplex. Note that the fatness values in the table are given with power $1/d$. It is due to the fact that fatness is a volume-based simplex quality, and taking the $(1/d)$-th power allows a better comparison. Also note that all normalized simplex qualities for the families $\tilde{A}_d$, $\tilde{B}_d$, $\tilde{C}_d$ and $\tilde{D}_d$ behave as $O(1/\sqrt{d})$. As illustrated for fatness and radius ratio in Figure 2, $\tilde{A}_d$ achieves the greatest simplex quality among the four families of Coxeter triangulations in each dimension d.

Acknowledgements

The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement No. 339025 GUDHI (Algorithmic Foundations of Geometry Understanding in Higher Dimensions). We also thank Arijit Ghosh, Ramsay Dyer and Jean-Daniel Boissonnat for discussion and suggestions.

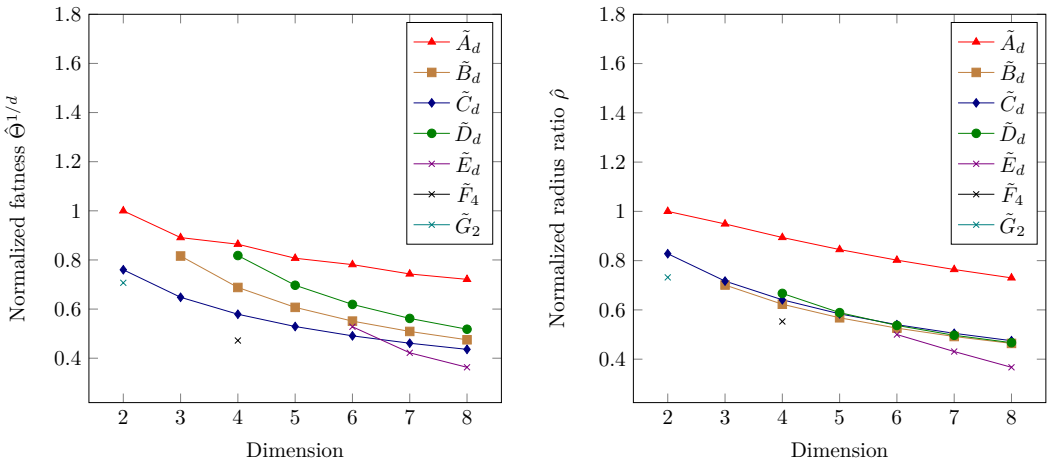


Figure 2 The visual representation of the normalized fatness and the radius ratio for simplices of $\tilde{A}_d$, $\tilde{B}_d$, $\tilde{C}_d$ and $\tilde{D}_d$ triangulations.

References

1 Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast High-Dimensional Filtering Using the Permutohedral Lattice. In *Computer Graphics Forum*, volume 29, pages 753–762. Wiley Online Library, 2010.

2 Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. The stability of Delaunay Triangulations. *Int. J. Comput. Geometry Appl.*, 23(4-5):303–334, 2013.

3 Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. Delaunay stability via perturbations. *Int. J. Comput. Geometry Appl.*, 24(02):125–152, 2014.

4 Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. A Probabilistic Approach to Reducing Algebraic Complexity of Delaunay Triangulations. In *Proceedings 17th ESA*, pages 595–606, 2015.

5 Nicolas Bourbaki. Lie groups and Lie algebras. Chapters 4–6. Translated from the 1968 French original by Andrew Pressley. Elements of Mathematics, 2002.

6 Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012.

7 A. Choudhary, M. Kerber, and S. Raghvendra. Polynomial-sized Topological Approximations using the Permutahedron. *Discrete & Computational Geometry*, Nov 2017.

8 Aruni Choudhary, Siargey Kachanovich, and Mathijs Wintraecken. Coxeter triangulations have good quality. Preprint, December 2017. URL: `https://hal.inria.fr/hal-01667404`.

9 J. H. Conway and N. J. A. Sloane. *Sphere-packings, Lattices, and Groups*. Springer-Verlag New York, Inc., 1987.

10 Harold SM Coxeter. Discrete groups generated by reflections. *Annals of Mathematics*, pages 588–621, 1934.

11 David P Dobkin, Allan R Wilks, Silvio VF Levy, and William P Thurston. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics (TOG)*, 9(4):389–423, 1990.

12 Robert V Moody and Jiří Patera. Voronoi and Delaunay cells of root lattices: classification of their faces and facets by Coxeter-Dynkin diagrams. *Journal of Physics A: Mathematical and General*, 25(19):5089, 1992.

13 VT Rajan. Optimality of the Delaunay triangulation in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 12(2):189–202, 1994.

A combinatorial measure of closeness in point sets

Alexander Pilz*[1] and Patrick Schnider[1]

1 **Department of Computer Science, ETH Zürich. Zürich, Switzerland.**
{alexander.pilz,patrick.schnider}@inf.ethz.ch

Abstract

We introduce *stripe closeness* and *stripe remoteness*, two combinatorial measures that capture how close together or far apart a set of query points lies within another set of points. The idea behind these concepts is that we look at all possible projections of the point set to a line and count the number of points that lie between the query points. For two points in a point set, the notion of stripe closeness can be seen as a combinatorial distance measure. We give bounds on the stripe closeness of two closest points. Further, we analyze stripe remoteness for triples in point sets and show that there are always three points that have high stripe remoteness.

1 Introduction

Let P be a set of points in general position (i.e., no three points on a line) and let a and b two points of P. How far is a from b? The common answer to this question is of course the Euclidean distance of a and b. However, this distance depends on the embedding of P and is not invariant under affine transformations. In many settings where point sets occur, we are not interested in actual coordinates of the points, but only their combinatorial structure (e.g., their allowable sequence or their order type). In these settings it seems natural to define a distance measure that only depends on the combinatorial structure of the point set.

For points on a line, there is a natural combinatorial distance measure: For any two points a and b in P we define the distance of a and b as the number of points of P that lie between a and b. Alternatively, we can also count the points that are not between a and b and define the distance between the total number of points and the number of these points. We can extend this idea for more query points. We define the *remoteness* of a one-dimensional point set Q with respect to a one-dimensional point set P as follows: Let a and b be the two extreme points of Q. Then the remoteness of Q with respect to P is the number of points of P that are between a and b. In particular, if Q consists of two points, then a small remoteness can be considered to have a small combinatorial distance. We generalize this concept to $\mathbb{R}^2$ by considering all projections of the two-dimensional point set to a line.

A *stripe* $s = (\ell_1, \ell_2)$ is a pair of two parallel lines ℓ_1, ℓ_2 in the plane. We say that the area of the plane that lies between the two lines is *inside* the stripe and denote it by $\mathsf{int}(s)$, while the rest of the plane is *outside* of the stripe, denoted by $\mathsf{out}(s)$. We consider ℓ_1 and ℓ_2 to be both inside and outside of s, that is, $\mathsf{int}(s) \cap \mathsf{out}(s) = \ell_1 \cup \ell_2$. Let P be a planar point set in general position. For any stripe s we define $i_P(s) := |\{p \in P \mid p \in \mathsf{int}(s)\}|$ and $o_P(s) := |\{p \in P \mid p \in \mathsf{out}(s)\}|$ as the number of points of P inside and outside of s, respectively. Let Q be another set of points in general position. We define the *stripe remoteness* of Q with respect to P as follows: consider all the stripes for which all of Q lies inside. Among those, pick one that has the smallest number of points of P inside. The stripe remoteness of Q with respect to P, denoted by $\mathsf{instripe}_P(Q)$, is the number of points of P

* Supported by a Schrödinger fellowship of the Austrian Science Fund (FWF): J-3847-N35.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

inside this stripe. Using the notation above, we can write this as

$$\mathsf{instripe}_P(Q) := \min_{s:Q\subset\mathsf{int}(s)} i_P(s) \ .$$

Note that if $|Q| = 2$ and $Q \subset P$, then $\mathsf{instripe}(Q) = 2$ as we can always choose both ℓ_1 and ℓ_2 to be the line through the two points. Hence, stripe remoteness is not a good candidate for a combinatorial distance measure. But for $|Q| > 2$ the situation is non-trivial. A point set Q having high stripe remoteness in P can be interpreted in the following way: In every projection to a line, there are two points of Q that have high distance w.r.t. P (i.e., the number of projected points of P between them is large). We show that for every point set P in general position, there are 3 points in P whose stripe remoteness is in $\Omega(|P|)$.

Note that in the one-dimensional case we might as well count the number of points that are not between a and b. The fewer such points there are, the further a is from b. We can also extend this idea to more points and two dimensions: We say that a stripe s is *between* Q if all points of Q are in $\mathsf{out}(s)$ and each connected component of $\mathsf{out}(s)$ contains at least one point of Q; this is denoted by $Q \prec s$. We define the *stripe closeness* of Q with respect to P as the minimal number of points of P outside a stripe that is between Q, i.e.,

$$\mathsf{outstripe}_P(Q) := \min_{s:Q\prec s} o_P(s) \ .$$

This measure is non-trivial already for $|Q| = 2$. We may define

$$d_P(a,b) := |P| - \mathsf{outstripe}_P(\{a,b\}) + 1 \ .$$

This corresponds to the maximum number of points strictly inside a stripe s.t. one of a and b is on each of the two lines defining the stripe. It is not too hard to check that d_P is a metric when defining $d_P(a,a) := 0$; note that (somewhat counterintuitive) the additive 1 is needed for the triangle inequality. Also note that any point set P contains two points a and b such that $d_P(a,b) = |P| - 1$. A point set Q having high stripe closeness can be interpreted in the following way: In every projection to a line, there are two points of Q with no other point of Q and only few points of P between them. We show that for every point set P in general position, we can find a subset of 2 points that have linear stripe remoteness. For the combinatorial distance measure $d_P(a,b)$, this implies that there are always two points of distance at most $(1-c)|P|$, for some constant c. We will see that this is asymptotically tight.

Projections of multivariate data to one and two dimensions is common in data analysis (see, e.g., [5]). However, distances usually do not only depend on the combinatorial properties of the point set. Combinatorial distance measures for two points a and b in a finite set P may be defined via the size of the intersection $P \cap R$ of P and a region R that contains a and b. More specifically, so-called *region-counting* distance functions have been used [3,6]: we are given two points p and q as well as a region R; translate, rotate, and scale both the points and R such that p and q coincide with a and b, and measure the distance as the size of the intersection with P and the transform of R. These measures have been used for point searching and nearest-neighbor problems [3,4,6]. However, they are not invariant under affine transformations. This is a property fulfilled by our approach; the distances are equivalent for all point sets with the same allowable sequence (i.e., there is a bijection between the point sets such that the order of the slopes defined by all point pairs is preserved). Our approach of taking the minimum or maximum is inspired by combinatorial properties of single points w.r.t. the point set, e.g., the *Tukey depth* [7] of a point p (which is the minimal number of points contained in a half-plane that also contains p). See [2] for a survey on depth measures.

2 Stripe Remoteness

In this section we prove the following result:

▶ **Theorem 2.1.** *Let α be the unique zero of $f(x) = 2x^3 - 3x^2 - 6x + 1$ in $[0, 1]$ ($\alpha \approx 0.155792$). Then for every $\epsilon > 0$ there is an $n_0 \in \mathbb{N}$ such that for every $n > n_0$ any set P of n points in general position contains three points p_1, p_2, p_3 for which $\mathsf{instripe}_P(\{p_1, p_2, p_3\}) \geq (\alpha - \epsilon)n$.*

This result can also be phrased in a slightly less technical manner:

▶ **Corollary 2.2.** *Let P be a set of n points in general position. Then P contains three points p_1, p_2, p_3 such that any stripe with p_1, p_2, p_3 inside has $\Omega(n)$ points of P inside.*

Proof of Theorem 2.1. Let P be a point set of n points in general position. We want to show that there are 3 points such that every stripe with them inside has at least c points inside, where $c = (\alpha - \epsilon)n$ for any $\epsilon > 0$. Let A be the dual line arrangement of P under the point-line-duality, in which we map a point $p = (x_p, y_p)$ to a line $p^* : y = x_p x + y_p$ (we may assume w.l.o.g. that no two points have the same x-coordinate). In the dual setting, a stripe translates into a vertical line segment, and the points that lie inside the stripe correspond to the lines intersected by the segment. Hence, in the dual setting, we want to find three lines such that any vertical line segment intersecting these three lines intersects at least c lines. We will show that there are three such lines in the following way: for every triple T of lines, we look at the shortest vertical line segment $s_T(-\infty)$ that intersects the three lines and lies on an (arbitrary) vertical line to the left of the leftmost crossing of the arrangement. We list all the lines crossed by $s_T(-\infty)$ in a list $L_T(-\infty)$. We then sweep the arrangement from left to right, always looking at the shortest vertical line segment $s_T(x)$ intersecting the triple and update the list $L_T(x)$ of lines crossed by it, whenever necessary. Clearly, an update is only necessary after the sweeping line passes over a crossing, so we only need to consider one x-coordinate between any two consecutive crossings c_i and c_{i+1}, which we denote by x_i. Additionally, let x_0 be an x-coordinate to the left of the first crossing c_1. We say that the triple T of lines is *valid after crossing* c_i if during the whole movement from left infinity to x_i (or, equivalently, shortly after the ith crossing), the list of crossed lines L_T contains at least c lines, that is, $|L_T(x_j)| \geq c$ for all $j \in \{0, \dots, i\}$. In particular, any triple that is valid after the last crossing satisfies the desired properties.

The triples T that are valid before the first crossing are the ones for which $s_T(x_0)$ intersects at least k lines, for $k \geq c$. As $s_T(x_0)$ is the shortest line segment intersecting T, the topmost and bottommost intersected lines have to be lines of T, the third line of T being one of the remaining $k - 2$. There are $n + 1 - k$ pairs of lines with exactly $k - 2$ lines between them, thus the total number of triples that are valid before the first crossing is

$$\sum_{k=c}^{n} (k-2)(n+1-k) = \frac{1}{6}(c-n-1)(2c^2 - cn - 10c - n^2 + 4n + 12) \ .$$

Moving over a crossing c_i, a triple T becomes invalid only if $s_T(x_{i-1})$ intersects exactly c lines, the topmost or the bottommost line is one of the lines of crossing i and the second line in the crossing is not in T. More precisely, let c_i be the crossing of two lines a and b where a is above b before the crossing. Let T be a triple that is valid after crossing c_{i-1} and that contains a where a is the topmost line intersected by $s_T(x_{i-1})$. If b is also in T, then $s_T(x_i)$ intersects the same lines as $s_T(x_{i-1})$, only that a and b have switched places, so T is still valid after crossing c_i. If $s_T(x_{i-1})$ intersects more than c lines, T is still valid after the crossing c_i. As the bottommost intersected line has to be in T, and a triple with b does not

become invalid, at most $(c-3)$ triples can become invalid at crossing c_i, as right before it there are $c-2$ lines between a and the bottommost line, one of them being b. The same arguments hold if b is the bottommost line intersected by $s_T(x_{i-1})$, so the number of triples that become invalid at crossing c_i is at most $2(c-3)$. Thus the total number of triples that are still valid after sweeping over all $\binom{n}{2}$ crossings of A is at least

$$\begin{aligned}\beta(n) &:= \sum_{k=c}^{n}(k-2)(n+1-k) - 2(c-3)\binom{n}{2} \\ &= \frac{c^3}{3} - \frac{c^2 n}{2} - 2c^2 - cn^2 + \frac{7cn}{2} + \frac{11c}{3} + \frac{n^3}{6} + \frac{5n^2}{2} - \frac{17n}{3} - 2 \ .\end{aligned}$$

In particular, if c is linear in n, that is, $c = \gamma n$, this is a polynomial of degree 3 with leading coefficient $\frac{\gamma^3}{3} - \frac{\gamma^2}{2} - \gamma + \frac{1}{6}$. This leading coefficient is larger than 0 in the interval $(0,1)$ if and only if $\gamma < \alpha$. Hence, for $\gamma = \alpha - \epsilon$, we have that $\lim_{n\to\infty}(\beta(n)) = \infty$, so for n large enough, there is at least one triple that is still valid after the last crossing. ◀

On the other hand, there are point sets where $\mathsf{instripe}(\{p_1,p_2,p_3\}) \leq (1-\epsilon)n$ for any three points p_1, p_2, p_3: Let P be a set of points in convex position and let $Q = \{p_1,p_2,p_3\}$ be any three points of P. Let h_1, h_2 and h_3 be the number of points along the boundary of the convex hull between p_1 and p_2, p_2 and p_3 and p_3 and p_1, respectively. By the pigeonhole principle, one of these number, without loss of generality h_1, is at least $\frac{n-1}{3}$. Consider the stripe defined by the line through p_1 and p_2 and the parallel line through p_3. This stripe has Q inside, but all the points between p_1 and p_2, of which there are at least $\frac{n-1}{3}$ many outside.

3 Stripe Closeness

We proceed with showing our bounds on the stripe closeness of two points.

▶ **Theorem 3.1.** *Let $\alpha = \sqrt{5} - 2 \approx 0.23607$. Then for every $\epsilon > 0$ there is an $n_0 \in \mathbb{N}$ such that for every $n > n_0$ any set P of n points in general position contains two points p_1, p_2 for which $\mathsf{outstripe}_P(\{p_1,p_2\}) \geq (\alpha-\epsilon)n + 2$.*

Again, the result can be phrased in a less technical manner:

▶ **Corollary 3.2.** *Let P be a set of n points in general position. Then P contains two points p_1, p_2 such that any stripe with p_1, p_2 outside has $\Omega(n)$ points of P outside.*

Proof of Theorem 3.1. Let P be a set of n points in general position. As in the proof of Theorem 2.1, we will work in the dual setting, only that now we want to find a pair of lines ℓ_1 and ℓ_2 such that every shortest vertical line segment intersecting the pair only intersects few lines, namely at most $c := (1-\alpha+\epsilon)n$ many. As the points outside of a stripe correspond to the lines not intersected by the dual vertical line segment, the existence of such a pair of lines shows the claimed result, as for a shortest vertical line segment intersecting ℓ_1 and ℓ_2, these two lines must be the topmost and bottommost intersection, and we can thus shorten every segment slightly, such that at least $n - c + 2 = n - (1-\alpha+\epsilon)n + 2 = (\alpha-\epsilon)n + 2$ many lines are not intersected.

We will again show the existence of such a pair of lines using a sweeping argument. For any pair R of lines, let $s_R(x)$ be the shortest vertical line segment intersecting R at that x-coordinate and let $L_R(x)$ be the respective list of intersected lines. Again, let x_i be an x-coordinate after the crossing c_i and before the crossing c_{i+1}. Analogously to triples in

the previous section, we call a pair of lines R *valid after crossing* c_i if during the whole movement from left infinity to x_i, the list of crossed lines L_T contains at most c lines, that is, $|L_T(x_k)| \leq c$ for all $k \in \{0, \dots, i\}$.

The number of pairs that are valid before the first crossing can be computed as

$$\sum_{k=2}^{c}(n+1-k) = \frac{1}{2}(c-1)(2n-c) \ .$$

Let now c_i be the crossing of two lines a and b where a is above b before the crossing. Let ℓ_1, ℓ_2 be a pair of lines and let ℓ_1 be above ℓ_2 at the x-coordinate of the crossing c_i. Assume that the pair ℓ_1, ℓ_2 becomes invalid at the crossing c_i. Then $s_{\{\ell_1,\ell_2\}}(x_{i-1})$ crosses exactly c lines and either $\ell_1 = b$ or $\ell_2 = a$, as in all other cases the segment $s_{\{\ell_1,\ell_2\}}(x_i)$ crosses c or more lines. In particular, at each crossing at most two pairs become invalid. However, the number of initially valid pairs is strictly smaller than two times the number of crossings, and we therefore need to be more thorough. Note that if $\ell_1 = b$, the pair can only become invalid if there are at least $c-1$ lines under the crossing, that is, if the crossing is above the c-level of the arrangement. Similarly, if $\ell_2 = a$, the pair only becomes invalid if the crossing is below the $(n-c-1)$-level. Alon and Györi [1] have shown that the number of crossings below the $(n-c-1)$-level is at most $(n-c-1)n$, if $(n-c-1) < \frac{n}{2}$. Indeed $n-c-1 = n-(1-\alpha+\epsilon)n-1 = \alpha n - \epsilon n - 1 < \frac{n}{2}$ as $\alpha < \frac{1}{2}$, thus by symmetry we get that in total at most $2(n-c-1)n$ pairs become invalid. So the number of pairs that are still valid after the last crossing is at least

$$\beta(n) := \sum_{k=2}^{c}(n+1-k) - 2(n-c-1)n = -\frac{c^2}{2} + 3cn + \frac{c}{2} - 2n^2 + n \ .$$

If c is linear in n, that is, $c = \gamma n$, this is a quadratic polynomial with leading term $-\frac{\gamma^2}{2}+3\gamma-2$. This leading term is larger than 0 in the interval $(0,1)$ if and only if $\gamma > 3-\sqrt{5} = 1-\alpha$. Hence for $\gamma = 1-\alpha+\epsilon$, we have that $\lim_{n\to\infty}(\beta(n)) = \infty$, so for n large enough, there is at least one pair that is still valid after the last crossing. ◀

On the other hand, we may have $\mathsf{outstripe}(\{p_1, p_2\}) \leq (1-\epsilon)n$ for any two points p_1, p_2.

▶ **Theorem 3.3.** *For any $n \geq 12$ there exist point sets P of n points in general position such that for every pair of points p_1, p_2 in P there is a stripe with p_1, p_2 outside but at least $\lfloor \frac{n}{3} \rfloor$ points of P inside.*

Proof Sketch. See Figure 1 for an accompanying illustration. Let m be the smallest integer that is divisible by 3 such that $m \geq n \geq 12$. Let a, b and c be three points that span an equilateral triangle with side length $\frac{1}{18}m^2$. Let a_1 and a_2 be the two points on the line through a and b that have distance $\frac{1}{6}m$ from a, with a_1 being closer to b. Similarly, let b_1 and b_2 be the two points on the line through b and c that have distance $\frac{1}{6}m$ from b, with b_1 being closer to c and let c_1 and c_2 be the two points on the line through c and a that have distance $\frac{1}{6}m$ from c, with c_1 being closer to a. Place n points as follows: Place $\frac{m}{3}$ points on the line segment between a_1 and a_2 such that the first point has distance $\frac{1}{2}$ to a_1 and any two consecutive points have distance 1 and call this point set A. Do the same for the line segments between b_1 and b_2 and between c_1 and c_2 to get point sets B and C, respectively. If necessary, take away 1 point from B and possibly another one from C. Finally, wiggle the point set slightly so that it is in general position.

It can be shown that if we project the segment c_1c_2 onto the line through a and b such that the image lies entirely in the segment a_1a_2, then the length of this image is smaller

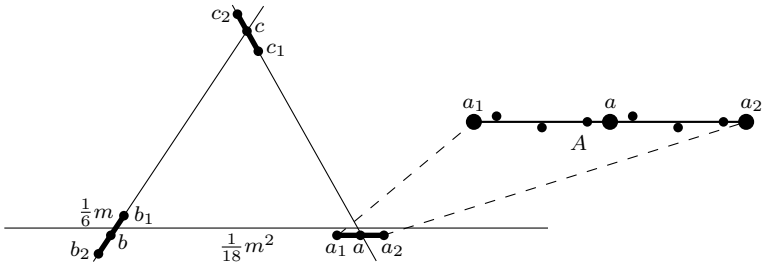


Figure 1 Construction for Theorem 3.3.

than 1. By symmetry, the lengths of the projections of a_1a_2 onto b_1b_2 and of b_1b_2 onto c_1c_2 are also smaller than 1. If p_1 and p_2 are in the same set, w.l.o.g. A, it thus follows that there is a stripe with p_1, p_2 outside and C inside. On the other hand, if p_1 and p_2 are in different sets, w.l.o.g. A and B, it can be easily argued that there is also such a stripe. ◀

4 Conclusion

We defined a combinatorial distance measure on point sets and showed that there are always points which are sufficiently close. The approach gives rise to several open problems.

The distances are equivalent for all point sets with the same allowable sequence. However, there can be two point sets with the same order type (i.e., there is a bijection between them such that the corresponding triples are oriented in the same way) for which the distance is different for two corresponding pairs. Reasonable generalizations of stripes for this setting could be double wedges. Can we get analogous bounds there?

Our result gives an upper bound on the distance between closest points when the stripe is required to be orthogonal to the line defined by the points. (This corresponds to a region-counting distance function with a stripe orthogonal and between its two reference points.) Is there a linear lower bound in that setting?

While our bounds are asymptotically tight, the gap between the constant factors are large. A natural open problem is to close these gaps.

References

1 Noga Alon and E Györi. The number of small semispaces of a finite set of points in the plane. *Journal of Combinatorial Theory, Series A*, 41(1):154 – 157, 1986.

2 Greg Aloupis. Geometric measures of data depth. In *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, volume 72 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 147–158, 2003.

3 Erik D. Demaine, John Iacono, and Stefan Langerman. Proximate point searching. *Comput. Geom.*, 28(1):29–40, 2004.

4 Jonathan Derryberry, Don Sheehy, Maverick Woo, and Danny Dominic Sleator. Achieving spatial adaptivity while finding approximate nearest neighbors. In *Proc. 20th Canadian Conference on Computational Geometry (CCCG 2008)*, 2008.

5 Jerome H. Friedman and John W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. Computers*, 23(9):881–890, 1974.

6 John Iacono and Stefan Langerman. Proximate planar point location. In *Proc. 19th Symposium on Computational Geometry (SoCG 2003)*, pages 220–226. ACM, 2003.

7 J. W. Tukey. Mathematics and the picturing of data. In *Proc. International Congress of Mathematicians*, pages 523–531, 1975.

An FPTAS for an Elastic Shape Matching Problem with Cyclic Neighborhoods

Christian Knauer, Luise Sommer, and Fabian Stehn

Institut für Informatik, Universität Bayreuth, Bayreuth, Germany

Abstract

The elastic geometric shape matching (EGSM) problem class is a generalisation of the well-known geometric shape matching problem class: Given two geometric shapes, the 'pattern' and the 'model', find a *single* transformation from a given transformation class that, if applied to the pattern, minimizes the distance between the transformed pattern and the model with respect to a suitable distance measure.

In EGSM, the pattern is divided into subshapes that are transformed by a 'transformation ensemble', i.e., a set of transformations. The goal is to minimize the distance between the union of the transformed subpatterns and the model in object space as well as the distance between specific transformations of the ensemble. The 'neighborhood graph' encodes which translations should be similar.

We present an FPTAS for EGSM instances for point sequences under translations with fixed correspondence where the neighborhood graph is a simple cycle.

1 Introduction

In classical geometric shape matching (GSM) problems, one is given a pattern P and a model Q, both from a class $\mathcal{S}$ of geometric shapes, along with a distance measure $d : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}_0^+$. The task is to find a single transformation t from a given transformation class $\mathcal{T}$ acting on $\mathcal{S}$, so that $d(t(P), Q)$ is minimized.

Matching geometric shapes is a problem that occurs in many applications such as character recognition, logo detection, human-computer-interaction, etc., and in a variety of different scientific fields, e.g., robotics, computer aided medicine, drug design, etc., and thus has already received a considerable amount of attention. We refer to the survey papers by Alt et al. [1] and Veltkamp et al. [4] for an extensive overview.

Many *geometric registration problems* (where the task is to align two shapes in different coordinate systems), e.g., between the coordinate system of an operation theatre and the coordinate system of a $3D$-model of a patient acquired during a pre-operative MRI scan, can be modelled as a GSM instance by appropriately choosing $\mathcal{S}$ and d. There, the transformation that minimizes the distance between both geometric shapes is then used as the mapping from the pattern space into the model space.

In many applications, where local distortions and complex deformations may occur, such as soft tissue registrations, GSM problems are too restrictive because a single transformation is chosen to match the entire pattern to the model. This issue is addressed by the *elastic geometric shape matching* (EGSM) framework, a generalisation of GSM. Here, the pattern is partitioned into subshapes and instead of one single transformation, a so-called transformation ensemble is computed. Each subshape of the pattern is transformed by an individual transformation of the ensemble in order to minimize the distance between the transformed pattern and the model. Also, the 'consistency' of the ensemble is guaranteed by forcing the transformations acting on some neighboring subshapes of the pattern to be similar with respect to a suitable similarity measure for the class of transformations at hand. The dependencies between the transformations of an ensemble are encoded in a so-called neighborhood graph.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

In [3], the authors considered several variants of this problem for different distance measures and graph families, including an algorithm that solves a variant of the problem for trees where only translations in a fixed direction are allowed in $O(n^2 \log n)$ time. In this paper, we focus on EGSM for point sequences under translations with fixed correspondence where the neighborhood graph is a simple cycle.

Problem Statement. In the following, everything is stated in $\mathbb{R}^2$, $\|\cdot\|$ denotes the Euclidean norm and translations are represented by translation vectors. All index arithmetic is modulo n.

▶ **Problem 1.1.** *Given two sequences of points $P = (p_0, \ldots, p_{n-1})$ (the pattern) and $Q = (q_0, \ldots, q_{n-1})$ (the model), find a sequence of translations $T = (t_0, \ldots, t_{n-1})$, so that the function $\gamma(T, P, Q) \coloneqq \max\{\max_{0 \le i < n} \|q_i - (p_i + t_i)\|, \max_{0 \le i < n} \|t_i - t_{i+1}\|\}$ is minimized.*

Measuring the distance of the points $(t_i + p_i)$ and q_i in model space is the same as measuring the distance of the points t_i and $q_i - p_i$ in translation space. This is why Problem 1.1 can be studied in translation space entirely: Let $c_i \coloneqq q_i - p_i$ for $0 \le i < n$ and $C \coloneqq (c_0, \ldots, c_{n-1})$. The function $\gamma(T, P, Q)$ can be rewritten as $\gamma(T, C) \coloneqq \max\{\max_{0 \le i < n} \|c_i - t_i\|, \max_{0 \le i < n} \|t_i - t_{i+1}\|\}$. We refer to points in translation space (i.e. translations) simply as points.

Basic Definitions. Let $c, u, v \in \mathbb{R}^2$ and $r > 0$. $D_r(c)$ denotes the disk with radius r centered in c and $\partial D_r(c)$ denotes its boundary. We define $I_r(c, u, v) \coloneqq D_r(c) \cap D_r(u) \cap D_r(v)$.
For a given sequence $C = (c_0, \ldots, c_{n-1})$, we define $\delta^* \coloneqq \min_T \gamma(T, C)$. We call a sequence of points $T = (t_0, \ldots, t_{n-1})$ *δ-admissible (for C)*, iff $\gamma(T, C) \le \delta$. A sequence, that is δ^*-admissible is called an *optimal* sequence. We will use the symbol T^* to denote an optimal sequence. A point t is called *(δ, i)-admissible (for C)*, iff there is a δ-admissible sequence T so that $T = (t_0, \ldots, t_i = t, \ldots, t_{n-1})$. Strictly speaking, δ^* and the concepts of admissibility depend on C, but since C is part of the input, we refrain from including C in the notation.

Previous Work and our Contribution. In [3] and in [2], the authors discussed several variants of EGSM problems. However, there are no results regarding problem instances, where the neighborhood graph is a simple cycle. In particular, there is no literature, that deals with efficient exact or approximation algorithms for Problem 1.1 and we do not know, if the problem is NP-hard.
In this paper, we provide an FPTAS for Problem 1.1 and prove that it computes a $(1+\epsilon)$-approximation to δ^* in $O\left(\epsilon^{-1/2} \left(\log \epsilon^{-1}\right)^2 n^3 \log n\right)$ time or in $O\left(\left(\log \epsilon^{-1}\right) \epsilon^{-2} n^2 \log n\right)$ time for some $\epsilon > 0$.

2 Our Results

Due to space limitations, the proofs of all lemmata and all figures have been omitted.
We define $\delta^{(3)} \coloneqq \gamma(C, C)$.

▶ **Lemma 2.1.** *C gives a 3-approximation to δ^*, i.e., $\delta^{(3)} \le 3\delta^*$.*

Lemma 2.1 is the basis for the construction of our FPTAS, since it implies, that every $(\delta^*, 0)$-admissible point lies within the disk $D_{\delta^{(3)}}(c_0)$. A simple way to get a $(1+\epsilon)$-approximation to δ^* for some $\epsilon > 0$ is to sample t_0 from a dense enough ϵ_{grid}-grid (a grid where the distance between samples is at most ϵ_{grid}) that covers $D_{\delta^{(3)}}(c_0)$. We call the points of this grid *translation-samples*. Here, $\epsilon_{\text{grid}} = \Theta\left(\epsilon \delta^{(3)}\right)$. We also sample the value δ of the objective

function on the interval $[\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$ with sample-distance $\epsilon_{\text{obj}} = \Theta\left(\epsilon\delta^{(3)}\right)$. We call the samples on $[\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$ *radius-samples*. We then test for every radius-sample δ, whether there exists a solution T' so that $\gamma(T', C) \leq \delta$ and a translation-sample is the ith component of T'. This test is a variant of a problem that has already been studied in [3], where the authors give an algorithm that solves this problem for paths in $O(n^2 \log n)$. Consequently, this simple FPTAS runs in $O\left(\epsilon^{-3} n^2 \log n\right)$ time.

This result can be improved in several ways. The first obvious improvement is to perform a binary search on $[\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$, which improves the run-time to $O\left(\left(\log \epsilon^{-1}\right) \epsilon^{-2} n^2 \log n\right)$.

The second idea is based on Lemma 3.1 below, which says, that for every $\delta \geq \delta^*$, there is a δ-admissible sequence T containing a point t_i that lies on $\partial D_\delta(c_i)$ for some i. Consequently, we do not have to sample the whole disk $D_{\delta^{(3)}}(c_i)$ for the current radius-sample δ, but to only sample $\partial D_\delta(c_i)$. Unfortunately, there is no way to identify the disks (the c_i) with this property, hence it is no longer possible to pick an arbitrary disk and sample it, but we have to sample the boundary of all disks. This changes the run-time to $O\left(\left(\log \epsilon^{-1}\right) \epsilon^{-1} n^3 \log n\right)$. Of course, this is only an improvement if $\epsilon^{-1} \gg n$. On the other hand, this strategy enables us to apply another modification: We can approximate each $\partial D_\delta(c_i)$ by a regular polygon with $O\left(\epsilon^{-1/2}\right)$ vertices. Due to the convexity of the problem, we can then perform a binary search on the edges of this polygon and get an FPTAS that runs in $O\left(\epsilon^{-1/2} \left(\log \epsilon^{-1}\right)^2 n^3 \log n\right)$ time. This gives us the following tradeoff between precision and input size:

▶ **Theorem 2.2.** *We can compute a $(1+\epsilon)$-approximation to δ^* in $O\left(\left(\log \epsilon^{-1}\right) \epsilon^{-2} n^2 \log n\right)$ time or in $O\left(\epsilon^{-1/2} \left(\log \epsilon^{-1}\right)^2 n^3 \log n\right)$ time.*

Since it is very clear how to implement the approximation when sampling the interior of $D_{\delta^{(3)}}(c_0)$, we elaborate on the improvements of the second strategy.

3 A Detailed Description

On (δ, i)-Admissible Points. The reason why it suffices to sample the boundaries of all disks rather than sampling the interior of one disk with a grid is, that any optimal solution T^* contains a key-point: A (δ^*, i)-admissible point t_i^* of $T^* = (t_0^*, \ldots, t_{n-1}^*)$ is called a *key-point*, iff $I_{\delta^*}(c_i, t_{i-1}^*, t_{i+1}^*) = \{t_i^*\}$ and $t_i^* \in \partial D_{\delta^*}(c_i)$.

▶ **Lemma 3.1.** *For every optimal sequence $T^* = (t_0^*, \ldots, t_{n-1}^*)$, there is an index $0 \leq i < n$ so that t_i^* is a key-point.*

On Computing δ^*. There is at least one index $0 \leq i < n$ for every $T^* = (t_0^*, \ldots, t_{n-1}^*)$, so that t_i^* is a key-point, which implies, that $t_i^* \in \partial D_{\delta^*}(c_i)$. Since we have no way of determining the index i, so that t_i^* is a key-point, the boundaries of all disks have to be sampled in order to find a suitable approximation to t_i^*. Since we do not know the optimal radius δ^* either, we have to sample the boundary of all disks for dense enough radius-samples in $[\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$. For every index i, let δ_i^* be the smallest (not necessarily a sample-radius) value, so that there is a (δ_i^*, i)-admissible point $t_i \in \partial D_{\delta_i^*}(c_i)$. In order to compute δ^* from the values $\delta_0^*, \ldots, \delta_{n-1}^*$, we need the following observation:

▶ **Lemma 3.2.** $\delta^* = \min_{0 \leq i < n} \delta_i^*$.

Consequently, in order to find δ^*, it suffices to compute δ_i^* for all $0 \leq i < n$.

Let $T^{(\epsilon)}$ be a solution so that $\delta^{(\epsilon)} := \gamma(T^{(\epsilon)}, C) \leq (1+\epsilon)\delta^*$. Let $t_i^{(\epsilon)}$ denote a $(1+\epsilon)$-approximation to a (δ_i^*, i)-admissible point t_i with $t_i \in \partial D_{\delta_i^*}(c_i)$. Let $\delta_i^{(\epsilon)}$ be the radius-sample of $t_i^{(\epsilon)}$. In order to prove that a binary search for δ_i^* on $[\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$ works for every index i, we need one more characteristic of (δ, i)-admissible points.

▶ **Lemma 3.3.** *Let $\bar{\delta} \geq 0$ be so that there is a $(\bar{\delta}, i)$-admissible point $t_i \in \partial D_{\bar{\delta}}(c_i)$. Then there is at least one (δ, i)-admissible point on $\partial D_\delta(c_i)$ for all $\delta \geq \bar{\delta}$.*

Consequently, a binary search for δ_i^* on $[\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$ can be carried out for every index $0 \leq i < n$ and it remains to determine a suitable sample-distance ϵ_{obj}: Since we aim for a $(1+\epsilon)$-approximation, we have to guarantee, that $\delta^{(\epsilon)} \leq (1+\epsilon)\delta^*$. Hence, it suffices to find some $\delta_i^{(\epsilon)} \in [\delta_i^*, \delta_i^* + \epsilon\delta_i^*]$ for each $0 \leq i < n$ and we have to choose ϵ_{obj} (the density of the radius-samples) subject to ϵ and $\delta^{(3)}$. The analysis on how ϵ_{obj} has to be chosen exactly will be carried out in Lemma 3.6, since it also depends on our final improvement, in particular on the polygons that will be used to approximate the boundaries of all disks.
In order to describe the final improvement in more detail, we need to briefly explain the 'propagation along the path' decision algorithm $A1$ of [3], that, given a radius-sample δ, a translation-sample t_i and an index i, decides, whether t_i is (δ, i)-admissible.

An Algorithm for Paths. A point $t_i \in D_\delta(c_i)$ is (δ, i)-admissible iff there are points $t_{i+1}, \ldots, t_{n-1}, t_0, \ldots, t_{i-1}$ so that all constraints encoded in $\gamma(T, C)$ are met. This chain of constraints can be interpreted as a path that starts and ends at t_i. In [3], the authors introduced an algorithm, that solves this problem for the case that only translations in a fixed direction are allowed. This algorithm can also be applied in our setting and then runs in $O(n^2 \log n)$ time and space. Starting at one end of the path, the basic idea is to propagate 'admissible regions', i.e., sets of translations that satisfy the current prefix of constraints, along the path. This is done by inflating them (i.e., computing the Minkowski sum of the region at hand and D_δ) and intersecting the result with the admissible region encoded in the subsequent node of the path. This strategy can be applied iteratively until either t_i is met again (in which case the algorithm returns YES) or the intersection of two regions is empty at some point. In this case the algorithm returns NO along with the tuple $(k(t_i), \mu(t_i))$, where $k(t_i)$ is the index of the first node that was not reached, and $\mu(t_i)$ is the Euclidean distance between the inflated version of the last non-empty admissible region and its succeeding admissible region.

Approximating the Boundary of a Disk with a Polygon. The simplest approach that tests, if there is a $(1+\epsilon)$-approximation to a key-point on $\partial D_\delta(c_i)$ is to pick $k = \Theta\left(\epsilon^{-1}\delta^{(3)}\right)$ suitably distributed translation-samples on $\partial D_\delta(c_i)$ and propagate all of them according to algorithm $A1$. In that way, $O(k)$ propagations (i.e., calls to algorithm $A1$) have to be carried out. This number can be reduced to $O(k^{1/2} \log k)$ by exploiting the convex structure of the admissible regions that occur during the propagation process: The main idea is to approximate $\partial D_\delta(c_i)$ by a regular polygon with $O(k^{1/2})$ vertices and to perform a binary search on each of its edges with a sample-distance that depends on ϵ and $\delta^{(3)}$.
Let $P_{\delta,p}(c_i)$ (or $P_\delta(c_i)$ in short) denote the inscribing regular polygon of $\partial D_\delta(c_i)$ with p vertices. By a slight abuse of notation, we identify $P_\delta(c_i)$ with its boundary, since we solely operate on the boundary of the polygons at hand. Also, let all such polygons be concentric.

▶ **Lemma 3.4.** *Let $p := \lceil 3^{1/4}\pi\epsilon^{-1/2} \rceil$ and let the edges of $P_\delta(c_i)$ be sampled with sample-distance ϵ_{edge} so that $\epsilon_{edge} \leq \frac{1}{3}\delta^{(3)}\epsilon$. Then, there is a translation-sample $t \in P_\delta(c_i)$ for every point $u \in \partial D_\delta(c_i)$ so that $\|t - u\| \leq \frac{1}{3}\delta^{(3)}\epsilon$.*

Strictly speaking, p depends on ϵ, but we refrain from including ϵ in the notation.
In the remainder, we will show, that the binary search among the samples on one edge of $P_\delta(c_i)$ can be carried out in $O\left(\log \epsilon^{-1} n^2 \log n\right)$ time. Here $O\left(n^2 \log n\right)$ is the time that is needed to carry out the propagation process for a single translation-sample t by algorithm A1. Since this approach builds on several properties of the tuple $(k(t), \mu(t))$ returned by algorithm A1, we have to introduce some of them first: The following lemma describes the dependency of the tuple on the translation-samples of one edge of $P_\delta(c_i)$.

▶ **Lemma 3.5.** *Let s and s' be two* NO*-instances of algorithm A1 for a given radius-sample δ, i.e., $A1(s, \delta, i) = (\text{NO}, (k(s), \mu(s)))$ and $A1(s', \delta, i) = (\text{NO}, (k(s'), \mu(s')))$, and let $t \in \overline{ss'}$. Then, either $A1(t, \delta, i) = \text{YES}$, or $A1(t, \delta, i) = (\text{NO}, (k(t), \mu(t)))$. In the latter case the tuple $(k(t), \mu(t))$ has the following properties:*

1. *$k(t) \geq \min\{k(s), k(s')\}$,*
2. *if $k(t) = k(s) = k(s')$, then $\mu(t) \leq \max\{\mu(s), \mu(s')\}$.*

Moreover, if $k(t) = k(s) = k(s')$ for all points $t \in \overline{ss'}$, the function $f \to [0, 1]$, $x \mapsto \mu((1-x)s + xs')$ is strictly convex.

As a consequence of the convexity of function f, in order to test if there is a (δ, i)-admissible point on the line segment $\overline{ss'}$ (which means, that there is a point on $t \in \overline{ss'}$ so that the propagation of t with radius-sample δ is successful) a binary search can be carried out among the samples along the line segment $\overline{ss'}$.
The runtime depends on the number of translation-samples that have to be evaluated, which is $O\left(\log {\epsilon_{\text{edge}}}^{-1}\right)$ for sample-distance ϵ_{edge}. Since every propagation takes $O(n^2 \log n)$ time, the procedure runs in $O\left(\log {\epsilon_{\text{edge}}}^{-1} n^2 \log n\right)$ time.
Since all edges of $P_\delta(c_i)$ have to be considered, evaluating the edges of one polygon takes $O\left(p \log {\epsilon_{\text{edge}}}^{-1} n^2 \log n\right)$ time.
We already know from Lemma 3.4, that the length of an edge of $P_\delta(c_i)$ is at most $2\delta\pi p^{-1}$. With $\epsilon_{\text{edge}} \leq \frac{1}{3}\delta^{(3)}\epsilon$ and $p = \left\lceil 3^{1/4}\pi\epsilon^{-1/2}\right\rceil$, the number of translation-samples, that have to be propagated, is

$$\log\left(\frac{2\delta\pi}{\epsilon_{\text{edge}} p}\right) \leq \log\left(\frac{6\delta\pi\sqrt{\epsilon}}{\epsilon\delta^{(3)}\pi\sqrt[4]{3}}\right) \leq \log\left(\frac{1}{\sqrt{\epsilon}}\left(\frac{6}{\sqrt[4]{3}}\right)\right) \in O\left(\log\frac{1}{\sqrt{\epsilon}}\right), \tag{1}$$

which leads to a runtime of $O\left(\epsilon^{-1/2} \log \epsilon^{-1} n^2 \log n\right)$ in total for the evaluation of one polygon and a fixed radius-sample.
For technical reasons, we also need to state the following insight:

▶ **Lemma 3.6.** *Let $\bar{\delta} \coloneqq \delta^* + \epsilon_{obj}$ and let s and s' be the endpoints of the circular arc of $(\bar{\delta}, i)$-admissible points on $\partial D_{\bar{\delta}}(c_i)$, then $\|s - s'\| \geq \epsilon_{obj}$.*
Let the sample-distance of the points on the edges of $P_{\bar{\delta}}(c_i)$ be $\epsilon_{edge} \coloneqq \frac{\sqrt{3}}{12}\epsilon\delta^{(3)}$ and let $\epsilon_{obj} \coloneqq \frac{\sqrt{3}}{12}\epsilon\delta^{(3)}$. Then there is a translation-sample on $P_{\bar{\delta}}(c_i)$ that is a $(1+\epsilon)$-approximation to t_i^*.

Description and Analysis of the Algorithm. We first describe the algorithm: At the start, the value of a 3-approximation to δ^* is computed in $O(n)$ time. Except for basic arithmetic operations, the algorithm consists of four nested loops: The first loop iterates over all of the n input points of the sequence C. For every such point a binary search for $\delta \in [\frac{1}{3}\delta^{(3)}, \delta^{(3)}]$ up to accuracy $\epsilon_{\text{obj}} = \frac{\sqrt{3}}{12}\delta^{(3)}\epsilon$ is carried out; this takes $O(\log \epsilon^{-1})$ steps. In each step of this binary search all $p = \left\lceil 3^{1/4}\pi\epsilon^{-1/2}\right\rceil \in O\left(\epsilon^{-1/2}\right)$ edges of $P_\delta(c_i)$ are inspected, and on

each of them a binary search among $\frac{2\delta\pi}{\epsilon_{\text{edge}}} \in O\left(\frac{1}{\epsilon}\right)$ translation-samples is performed. Each translation-sample is propagated with algorithm A1 for paths, which takes $O(n^2 \log n)$ time per call. This gives a total runtime of $O\left(\epsilon^{-1/2}\left(\log \epsilon^{-1}\right)^2 n^3 \log n\right)$.

If $\delta^{(\epsilon)} < \delta^{(3)}$ is returned, it is valid since there was a translation-sample that has been propagated successfully and therefore is part of a $\delta^{(\epsilon)}$-admissible sequence T. This also means that the very translation-sample that establishes $\delta^{(\epsilon)}$ is propagated and together with the intermediate steps of the propagation gives a T, which then serves as a witness. If there was no successful propagation, $\delta^{(\epsilon)} = \delta^{(3)}$ is returned and we know from Lemma 2.1 that there is always a $\delta^{(3)}$-admissible sequence.

Now we analyse the precision of the algorithm: The precision of the binary search on δ is $\epsilon_{\text{obj}} < \frac{1}{3}\delta^{(3)}\epsilon$; also, all polygons are concentric by construction. If $P_\delta(c_i)$ and $P_{\delta+\epsilon_{\text{obj}}}(c_i)$ are two polygons with circumradii that differ by ϵ_{obj}, the distance between any point on $P_\delta(c_i)$ and the polygon $P_{\delta+\epsilon_{\text{obj}}}(c_i)$ is at most ϵ_{obj} and vice versa. Every edge of these two polygons is sampled with points of distance ϵ_{edge}, and with Thales' theorem it follows that for every translation-sample on $P_\delta(c_i)$ there is a translation-sample on $P_{\delta+\epsilon_{\text{obj}}}(c_i)$ with distance $\frac{1}{3}\delta^{(3)}\epsilon$ or less and vice versa. Combined with Lemma 3.4, we have that for every δ there is a translation-sample in $D_\epsilon(t_i)$ for every (δ, i)-admissible point $t_i \in \partial D_\delta(c_i)$. According to Lemma 3.6, one of the following two cases holds: Either there is at least one (δ, i)-admissible translation-sample on $P_\delta(c_i)$ for every $\delta \geq \delta^* + \frac{1}{3}\delta^{(3)}\epsilon$ so that the line segment of all (δ, i)-admissible points on one of the edges of this polygon is at least $\frac{1}{3}\delta^{(3)}\epsilon$ long, or one vertex of the polygon is a (δ, i)-admissible point and since all polygons are concentric, this vertex is (δ, i)-admissible for every $\partial D_\delta(c_i)$ with $\delta \geq \delta^*$. We consider the radius-samples $\bar{\delta}$, $\bar{\delta}+\epsilon_{\text{obj}}$ and $\bar{\delta}+2\epsilon_{\text{obj}}$, where $\bar{\delta} \coloneqq \delta^*+\zeta-\epsilon_{\text{obj}}$ for some $0 < \zeta < \epsilon_{\text{obj}}$. Since $\bar{\delta} < \delta^*$, none of the propagations for this δ are successful. A short analysis leads to $\bar{\delta}+\epsilon_{\text{obj}} < \delta^*+\epsilon_{\text{obj}} < \bar{\delta}+2\epsilon_{\text{obj}} < \delta^*+\frac{1}{3}\delta^{(3)}\epsilon$. Due to Lemma 3.6, this means, that for radius-sample $\bar{\delta}+2\epsilon_{\text{obj}}$ the two endpoints of the circular arc of $(\bar{\delta}+2\epsilon_{\text{obj}}, i)$-admissible points in $D_{\bar{\delta}+2\epsilon_{\text{obj}}}(c_i)$ have a distance of at least ϵ_{obj}, which is why there is at least one translation-sample on the inscribing polygon of this disk, that is propagated successfully and the algorithm returns $\delta^{(\epsilon)} = \bar{\delta}+2\epsilon_{\text{obj}} < \delta^*+\frac{1}{3}\delta^{(3)}\epsilon < (1+\epsilon)\delta^*$ as the approximation to δ^*. Hence the algorithm computes a $(1+\epsilon)$-approximation to δ^* for Problem 1.1.

It also returns a $(\delta^{(\epsilon)}, i)$-admissible point $t^{(\epsilon)}$ from which an $\delta^{(\epsilon)}$-admissible sequence $T^{(\epsilon)}$ can be computed in $O(n^2 \log n)$ time.

References

1 Helmut Alt and Leonidas Guibas. Discrete Geometric Shapes: Matching, Interpolation, and Approximation. In *Handbook of Computational Geometry*, pages 121–153. Elsevier B.V., 2000.

2 Christian Knauer, Luise Sommer, and Fabian Stehn. Elastic Geometric Shape Matching for Translations under the Manhattan Norm. In *Computational Geometry: Theory and Applications*, 2018.

3 Christian Knauer and Fabian Stehn. Elastic Geometric Shape Matching for Point Sets under Translations. In *Proceedings of Algorithms and Data Structures - 14th International Symposium (WADS)*, pages 578–592, 2015.

4 Remco C. Veltkamp and Michiel Hagedoorn. State of the art in shape matching. In *Principles of Visual Information Retrieval. Advances in Pattern Recognition*, pages 87–119. Springer, London, 2001.

Group Diagrams for Representing Trajectories

Maike Buchin[1] and Bernhard Kilgus[2]

1 **Faculty of Computer Science, TU Dortmund,** `maike.buchin@tu-dortmund.de`
2 **Department of Mathematics, Ruhr University Bochum,** `bernhard.kilgus@rub.de`

Abstract

We propose the group diagram as a representation for multiple trajectories representing one or several moving groups. Given a distance threshold, a similarity measure and a minimality criterion a minimal group diagram is a minimal representation of the groups maintaining the spatio-temporal structure of the groups' movement. We give hardness results and approximation algorithms for computing several variants of the group diagram.

1 Introduction

A moving object, called *entity*, is described by its location at n time stamps and a linear interpolation inbetween each two consecutive time stamps. The corresponding trajectory therefore is a polygonal line. Given k trajectories, each of complexity n forming one or several (overlapping, i.e., splitting and merging) groups we introduce the *group diagram* as a means of compactly representing these groups. We propose the following general definition:

▶ **Definition 1.1.** A group diagram (GD) is a geometric graph with vertices augmented by a temporal component, that represents all input trajectories $\mathcal{T}$. We say the graph represents a trajectory $T \in \mathcal{T}$ if there is a similar path P in the graph, that is T and (the geometric representation of) P are similar under a given similarity measure. We say a group diagram is minimal if it is minimal in size, either with respect to its number of edges or the total length of edges.

We consider GD which are built from the input trajectories, i.e., edges of the GD are represented by subtrajectories of the input and two edges share a vertex if the endpoints of the corresponding subtrajectories are within distance d from each other. Endpoints of edges with no d-distance neighbor have degree one. Vertices in the graph are hence embedded as the set of end points of incident edges. We will use such graphs in the following. Note that we could transform these into planar embedded graphs, for instance by choosing and connecting to the midpoint of the point set of a vertex.

As similarity measure we consider three popular measures on trajectories: the Fréchet distance, equal-, and similar-time distance. Figure 1 illustrates several trajectories. The subtrajectories forming a minimal GD for the given trajectories are highlighted in red.

Minimizing the number of edges or their total length seems intuitively reasonable. However both can lead to strange effects illustrated in Figure 2 which shows two simple examples (one or two trajectories of complexity 1). In the left figure the input consists of a single trajectory of length $6d$ and the GD with minimal length consists only of the red points, which is a bad representation of the movement. In the right picture, the GD minimizing the number of edges consists of the two input trajectories, although we would like the common movement between A and B to be represented by only one representative. To prevent these effects we make the following further requirements.

- When minimizing number of edges, we require that as much as possible is jointly represented. Given a subtrajectory τ of an edge of a GD $\mathcal{G}$, let $c(\tau)$ denote all subtrajectories

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

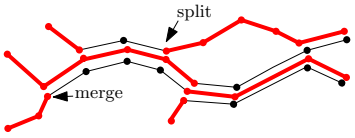


Figure 1 Illustration of GD.

Figure 2 unintuitive GD with minimal edge length (left) and minimal number of edges (right).

of the input trajectories within distance at most d to τ and let $G^*_\tau := c(c(\tau))$ denote the union of all subtrajectories within distance at most d to a curve in $c(\tau)$. Furthermore, we define $A_\tau := \mathcal{G} \cap G^*_\tau$. We now demand that for each subtrajectory τ the resulting set A_τ is a minimal representation for $c(A_\tau)$.

- When minimizing the total length, we require that no clusters are artificially split up to reduce the length. Formally, we require that no subgraph of the GD can be contracted, i.e., substituted by a subgraph of smaller size (but possibly larger length).

Related Work Two related notions to the GD are the grouping structure and flow diagrams. The grouping structure is the unique graph representing all density-connected groups traveling at equal-time [6]. A flow diagram is a minimal (in the number of vertices) diagram representing segmentations of all input trajectories. In a flow diagram nodes represent criteria and edges transitions between criteria [3]. The grouping structure is a specialization of the GD, which uses the equal-time distance, and density connectedness as inner group distance. The flow diagram can be seen as generalization of the GD (after switching between vertices and edges) where criteria are more general than small distance of the trajectories. Computing a GD with Fréchet distance as distance measure is also highly related to map construction algorithms, where the goal is to determine the underlying network of a set of trajectories [1]. Similar modeling choices (edges, similarity measure, and minimality condition) occur in the problem of finding a representative (median, middle, ...) trajectory of a set of similar trajectories.

Complexity Analysis By a reduction from the known NP-complete Dominating-Set problem for a grid graph [7] we can show that the decision problem for GD is NP-complete for all variants we consider.

▶ **Theorem 1.2.** *Given an integer l, deciding whether there exists a GD of size l is NP-complete for both l denoting the edge length and l denoting the edge number, and for both Fréchet distance and equal-time distance as similarity criteria.*

In the following two sections we give approximation algorithms and their experimental evaluation on a real data set. Due to space limitations many details, in particular proofs, are omitted in this extended abstract, and will be given in the full version.

2 Approximation Algorithms

Our approach is based on a natural formulation of the problem as a Set-Cover instance, which we first build and then solve approximately. To do so, we use the following concepts. A *cluster* is a set of trajectories called *cluster curves* that are all similar (under some similarity measure) to one *representative* of the cluster. Each edge in a GD can be identified with a representative of a cluster of subtrajectories from the input. We first detect all *relevant cluster representatives* and then we select a minimal set of these clusters where the union covers the complete input. Thus our approach is to construct and solve a Set-Cover instance

with universe $\mathcal{U}$ consisting of all segments of the input trajectories. To make this approach computationally feasible we segment the trajectories such that we only need to consider subtrajectories starting and ending at vertices of the segmentation as cluster representatives.

As we construct and approximately solve a Set-Cover instance we obtain approximation algorithms in both cases, minimizing the total size of the GD and minimizing the length. When minimizing length for the Fréchet distance we additionally make a small additive error for each edge of the GD, see Lemma 2.2.

To take the two different minimality criteria into account we can formulate a weighted Set-Cover problem where we assign a weight to each representative (subset) depending on which minimality condition we choose: unit weight for number of edges and length of representative for edge length. Next we show how to implement this approach for the different distance measures.

2.1 Fréchet Distance

Recall that the Fréchet distance between two curves τ and σ is defined as the infimum over all reparameterizations α and β of $[0,1]$ of the maximum over all $t \in [0,1]$ of $\|\tau(\alpha(t)) - \sigma(\beta(t))\|$ [2]. To compute a minimal GD with Fréchet distance as similarity measure we use a sweep algorithm with two moving points a and b along every trajectory and report all *relevant* clusters represented by the subtrajectory between the current positions of a and b as described in [4].

▶ **Definition 2.1.** A cluster representative τ which represents the cluster $c(\tau)$ is *irrelevant* if it can be extended to τ' such that $c(\tau')$ contains only extended curves of $c(\tau)$ and $|c(\tau)| = |c(\tau')|$ and no other trajectory not in the cluster enters a d-tube around one of the cluster curves. If a cluster representative cannot be extended in such a way the representative and the corresponding cluster are *relevant*.

▶ **Lemma 2.2.** *When minimizing size, there always exists a minimal GD solution where edges correspond to relevant cluster representatives. When minimizing length, this solution adds at most an additive error of* $2de$*, where* d *is the distance threshold and* e *is the number of edges of an optimal solution.*

Note that this additive error is tight: Consider an input setting of trajectories of complexity one and length greater than $2d$, where always two trajectories are congruent and the pairs are within distance greater than d. Here, only the whole trajectories are relevant and therefore each representative (whole trajectory) is $2d$ longer than a minimal representative (middle part with distance d to endpoints).

When each segment has length greater than $4d$, which is the case in our experiments, we have a multiplicative error of at most 2 when using only relevant representatives.

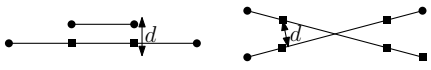


Figure 3 Inserting new vertices. The vertices from the input are shown as disks whereas the newly added ones are marked as squares.

Segmentation First we observe that using the segmentation given by the input vertices does not suffice for a minimal representation, e.g., in the case of parallel lines with different starting points. To obtain a sufficiently fine partition of the trajectories we consider two different triggers for inserting a new vertex. Firstly, for every vertex v of the input data we add a vertex to every segment which has distance to v less than or equal to d (see Figure 3*a*) at the point along the segment where the distance to v is

minimal (type 1). Secondly, we add a new vertex if the distance between two segments is less than d for the first time and if the distance exceeds d again (type 2) (see Figure 3*b*).

▶ **Lemma 2.3.** *After two steps of inserting new vertices all relevant clusters start and end at vertices.*

From Lemma 2.3 it follows that we can use the vertices of the trajectories after two steps of vertex insertion as the event points of the sweep algorithm. For each trajectory τ we move b to the right until the representative is relevant and all cluster curves of the corresponding cluster start and end at vertices. Then we report this cluster as one subset of the Set-Cover instance, set a to the position of b and proceed like this until we reach the end of τ.

▶ **Theorem 2.4.** *Let N be the complexity of a trajectory after two steps of vertex insertion. Given a GD instance using Fréchet distance we can compute in $\mathcal{O}(k^2N^3)$ time a* Set-Cover *instance of size $|\mathcal{U}| = \mathcal{O}(kN)$ and $|\mathcal{S}| = \mathcal{O}(kN)$, the solution of which solves the GD instance.*

▶ Remark. The value N is in $O(k^2C^2n)$, where C is a constant bounding the number of intersections of one segment with all segments of the input (see the full version for details). Note that this is linear in the dominating parameter n, since $k \ll n$.

2.2 Equal- and Similar-Time Distance

Next we want to compute a GD based on equal-time distance as similarity measure [5]. A path P within a group diagram is similar to an input trajectory τ if for any t in the domain of τ the Euclidean distance $dist(P(t), \tau(t))$ is at most d. The following observation follows directly from the linear interpolation between two vertices of a trajectory.

▶ Remark. Given two piecewise-linear trajectories τ_1, τ_2 with vertices at the locations corresponding to the time stamps $t_1, ..., t_m$. Then if $dist(\tau_1(t_i), \tau_2(t_i)) \leq d$ and $dist(\tau_1(t_{i+1}), \tau_2(t_{i+1})) \leq d$ we have $dist(\tau_1(t), \tau_2(t)) \leq d$ for all $t \in (t_i, t_{i+1})$.

Segmentation Using this observation we insert a sufficient number of time stamps and corresponding vertices additional to the input vertices to ensure that between consecutive time stamps the pairwise equal-time distance of the trajectories does not change with respect to threshold d. We do this by simulating equal-time distance first, i.e., inserting (by interpolation) a vertex to each trajectory for the at most kn different time stamps. Subsequently, we consider only the common time interval of all trajectories. Then we compare all segments between two consecutive time stamps in a second step.

Let $\overline{AB}$ and $\overline{CD}$ be two segments of different trajectories between two consecutive time stamps i and $i+1$. If $dist(\overline{AB}_t, \overline{CD}_t) \leq d$ holds for $t = t_i$ and $t = t_{i+1}$ the segments are at equal-time distance at most d for all $t \in (t_i, t_{i+1})$ and we do not need to insert any new vertices. If $dist(\overline{AB}_t, \overline{CD}_t) \leq d$ holds for i but not for $i+1$ the equation $dist(\overline{AB}_t, \overline{CD}_t) = d$ has exactly one solution t_s in (t_i, t_{i+1}) and we insert a new vertex to all trajectories (if possible) at the corresponding locations at t_s *(split event)*. Analogously we calculate t_s and insert new vertices if the inequality holds for $t = i+1$ but not for $t = i$ *(merge event)*. Lastly, if the inequality does not hold for $t = i$ nor for $t = i+1$ the equation $dist(\overline{AB}_t, \overline{CD}_t) = d$ has either no solution or exactly two solutions $t_{\min}$ and $t_{\max}$ in I. In the first case we can conclude that the segments do not share a part where the equal-time distance is less than or equal to d. In the latter case we obtain one merge and one split event between $t = i$ and $t = i+1$. Again, we insert vertices to every trajectory at time $t_{\min}$ and $t_{\max}$.

▶ Lemma 2.5. *The segmentation takes $\mathcal{O}(k^4 n \log n)$ time. After this process each of the k trajectories has at most k^3n vertices.*

Computing the GD For computing the GD we proceed in the following way. Between each two consecutive time stamps in V we compute one subset for each segment, which contains the indices of all other segments within equal-time distance at most d. The distance between two segments is the maximum of the Euclidean distance between the two starting points of the segments and their two ending points.

Then we solve the Set-Cover instance and report the segments which correspond to the selected subsets. When minimizing the total edge number of the GD we have to ensure that the representation does not change when not necessary in terms of minimality. Otherwise the GD consists of edges that could be concatenated. This can happen because the solution of the Set-Cover in general is not unique. To maintain one representation as long as possible we check if the representation R_{old} between the previous two time stamps still represents all segments between the current two timestamps and if the size of R_{old} equals the size of the current solution. In this case we maintain R_{old} and proceed with the next time stamp. This additional step is not necessary when minimizing the total edge length as the sum of the length of a minimal length representation between a series of consecutive time stamps within a time frame from start t_s to end time t_e is at most the minimal length of a representation looking at the whole interval $[t_s, t_e]$ at once.

▶ Lemma 2.6. *For each time stamp we can compute the k sets of the* Set-Cover *instance in $\mathcal{O}(k^2)$ time.*

▶ Theorem 2.7. *Given a GD instance using equal time distance, we can compute in $\mathcal{O}((k^5 + k^4 \log n)n)$ time $\mathcal{O}(k^3 n)$* Set-Cover *instances each of size $|\mathcal{U}| = k$ and $|\mathcal{S}| = k$ the solution of which solves the GD instance.*

Similar-time Distance Equal-time distance may be too restrictive for some applications, for example for entities which travel the exact same route, but such that each entity reaches each position with a small delay. We use the term similar-time distance when we allow a bounded time shift when comparing two positions.

3 Experiments

In order to investigate the usability of our definition of a group diagram and the described algorithms for real world data we performed experiments on data of migrating greater white-fronted geese (Anser a. albifrons) with parents and two juveniles. For each animal we had approximately 2000 positions which were collected in half-hourly bursts of 20 GPS positions in 1 Hz resolution. The distance between two entities is computed based on their positions on the earth's surface only. A group diagram shows when a subgroup (or one single entity) separates from the rest of the group (or from a subgroup) and when a subgroup joins another subgroup. Detecting and visualizing split and merge events is an interesting application of the group diagram to help answering questions like: When is the family flying close together, so that it can be represented by only one member and when do we need more representatives? For which distance does the family stay "together" the whole time or a given percentages of the whole observation period? We computed group diagrams based on bounded equal-time and α-similar-time distance as similarity measure for one family (two adults and two juveniles) for distances $d = 3, 5, 10, 20, 40, 80, 160, 320, 640, 1280$ meters and, for each distance, we set the allowed time shift to $\alpha = 0, 10$ seconds. We give a summary of the experiments here, for more details and an evaluation of the experimental computation time see the full version of the paper.

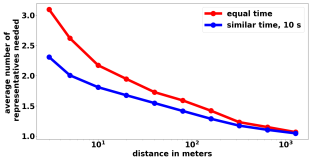


Figure 4 Representability of the family for increasing distance values.

Figure 5 Family members split when flying over a lake and merge after passing the lake.

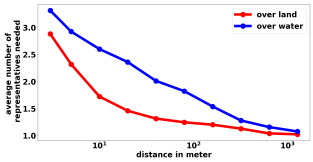


Figure 6 Difference of representability of the family while flying over land and over water.

Number of Representatives for equal- and similar-time distance In Figure 4 the average number of representatives needed to represent the whole family is plotted against the distance thresholds for bounded equal-time distance and similar-time distance as similarity criteria. For small distances (10 m and 20 m) the impact of allowing a time shift of 10 seconds is greater than the impact of doubling the distance. As distance increases it becomes the dominating parameter for the size of the group diagram. The reason for this observation most likely is the formation of the flock while flying. If the entities of the flock are flying within a V-formation or in a line two entities are represented with only one representative even if their distance is greater than the given threshold when we allow a small time shift. The impact of a time shift would be less if the birds were flying next to each other rather than behind each other like in a line or a V-formation.

Migration Over Water and Over Land During the migration one can observe that when the family is flying over surfaces of water they tend to separate more from each other than while flying over solid ground. One example of this phenomenon is shown in Figure 5 for a bounded equal-time distance of 160 meters. Figure 6 shows the difference in the number of representatives needed for flying over solid ground and flying over water. One interesting observation is that the values differ the most between 10 and 100 m.

Acknowledgments. This work was supported by the Deutsche Forschungsgemeinschaft (DFG), project BU 2419/3-1.

References

1 M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. *Map Construction Algorithms*. Springer, 2015.

2 H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5(1):75–91, 1995.

3 K. Buchin, M. Buchin, J. Gudmundsson, M. Horton, and S. Sijben. Compact flow diagrams for state sequences. In *Proc. Int. Symp. Experimental Algorithms*, pages 89–104, 2016.

4 K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *Int. J. Comput. Geometry Appl.*, 21(3):253–282, 2011.

5 K. Buchin, M. Buchin, M.J. van Kreveld, and J. Luo. Finding long and similar parts of trajectories. *Comput. Geom.*, 44(9):465–476, 2011.

6 K. Buchin, M. Buchin, M.J. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. *J. of Comput. Geometry*, 6(1):75–98, 2015.

7 B. N. Clark, C.J. Colbourn, and D.S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

Agglomerative Clustering of Growing Squares*

Thom Castermans[1], Bettina Speckmann[1], Frank Staals[2], and Kevin Verbeek[1]

1 **TU Eindhoven**
[t.h.a.castermans|b.speckmann|k.a.b.verbeek]@tue.nl
2 **Utrecht University**
f.staals@uu.nl

1 Introduction

We study an agglomerative clustering problem motivated by interactive glyphs in geo-visualization. *GlamMap* [5] is a visual analytics tool for the eHumanities which allows the user to interactively explore metadata of a book collection. Each book is depicted by a square, color-coded by publication year, and placed on a map according to the location of its publisher. Overlapping squares are recursively aggregated into a larger glyph until all glyphs are disjoint. As the user zooms out, the glyphs "grow" relative to the map to remain legible. As glyphs start to overlap, they are merged into larger glyphs to keep the map clear and uncluttered. To allow the user to filter and browse real world data sets at interactive speed we hence need an efficient agglomerative clustering algorithm for growing squares (glyphs).

Formal problem statement. Let P be a set of points in $\mathbb{R}^2$. Each point $p \in P$ has a positive weight p_w. Given a "time" parameter t, we interpret the points in P as squares. More specifically, let $\square_p(t)$ be the square centered at p with width tp_w. For ease of exposition we assume all point locations to be unique. Furthermore, we refer to P as a set of squares rather than a set of center points of squares. Observe that initially, i.e. at $t = 0$, all squares in P are disjoint. As t increases, the squares in P grow, and hence they may start to intersect. When two squares $\square_p(t)$ and $\square_q(t)$ intersect at time t, we remove both p and q and replace them by a new point $z = \kappa p + (1 - \kappa)q$, with $\kappa = p_w/(p_w + q_w)$, of weight $z_w = p_w + q_w$. Our goal is to compute the complete sequence of events where squares intersect and merge.

Results. We present a fully dynamic data structure that can maintain a set P of n disjoint growing squares. Our data structure can report the first time two squares in P intersect, and supports updates (inserting or deleting a square) in $O(\log^7 n)$ amortized time. Queries asking whether a query square $\square_q$ currently intersects a square $\square_p$ in P take $O(\log^3 n)$ time, and the space usage is $O(n(\log n \log\log n)^2)$. Using this data structure we can compute the agglomerative clustering for n squares in $O(n\alpha(n)\log^7 n)$ time. Here, α is the extremely slowly growing inverse Ackermann function. To the best of our knowledge, this is the first fully dynamic clustering algorithm which beats the straightforward $O(n^2 \log n)$ time bound. This abstract focuses on the update and query times for our data structure. Omitted proofs and detailed bounds on space usage, as well as related discussions on the relation between canonical subsets in dominance queries, can be found in the full version [4].

Related Work. Funke, Krumpe, and Storandt [6] introduced so-called "ball tournaments". Their input is a set of balls in $\mathbb{R}^d$ with an associated set of priorities. The balls grow linearly and whenever two balls touch, the lower priority ball is eliminated. The goal is to

* The Netherlands Organisation for Scientific Research (NWO) is supporting T. Castermans (project number 314.99.117), B. Speckmann (project number 639.023.208), F. Staals (project number 612.001.651), and K. Verbeek (project number 639.021.541).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

compute the elimination sequence efficiently. Funke and Storandt [7] show how to compute an elimination sequence for n balls in $O(n \log \Delta(\log n + \Delta^{d-1}))$ time in arbitrary dimensions and in $O(Cn \operatorname{polylog} n)$ time for $d = 2$, where C denotes the number of different radii and Δ the ratio of the largest to the smallest radius. Ahn et al. [2] recently developed the first sub-quadratic algorithms to compute elimination orders for ball tournaments. Their results apply to balls and boxes in two dimensions or higher. Specifically, for squares in 2D they can compute an elimination order in $O(n \log^4 n)$ time. Their results critically depend on the fact that the elimination priorities are given and that they have to handle only deletions.

Alexandron et al. [3] present a dynamic and kinetic data structure for maintaining the convex hull of points moving in $\mathbb{R}^2$. Their data structure processes (in expectation) $O(n^2 \beta_{s+2}(n) \log n)$ events in $O(\log^2 n)$ time each. Here, $\beta_s(n) = \lambda_s(n)/n$, and $\lambda_s(n)$ is the maximum length of a Davenport-Schinzel sequence on n symbols of order s. Agarwal et al. [1] present dynamic and kinetic data structures for maintaining the closest pair and all nearest neighbors. The expected number of events processed is roughly $O(n^2 \beta_{s+2}(n) \operatorname{polylog} n)$, each of which can be handled in $O(\operatorname{polylog} n)$ expected time.

2 Geometric Properties

Let ℓ_q denote the bottom left vertex of a square $\square_q$, and let r_q denote the top right vertex of $\square_q$. Furthermore, let $D(q)$ denote the subset of points of P dominating q, and let $L(q) = \{\ell_p \mid p \in D(q)\}$ denote the set of bottom left vertices of the squares of those points.

▶ **Observation 2.1.** Let $p \in D(q)$ be a point dominating q. The squares $\square_q(t)$ and $\square_p(t)$ intersect at time t if and only if $r_q(t)$ dominates $\ell_p(t)$ at time t.

Consider a line γ with slope minus one, project all points in $Z(t) = \{r_q(t)\} \cup L(q)(t)$, for some time t, onto γ, and order them from left to right. Observe that, since all points in Z move along lines with slope one, this order does not depend on the time t. Moreover, for any point p, we have $r_p(0) = \ell_p(0) = p$, so we can easily compute this order by projecting the centers of the squares onto γ and sorting them. Let $D^-(q)$ denote the (ordered) subset of points in $D(q)$ that occur before q in the order along γ, and let $D^+(q)$ denote the ordered subset of $D(q)$ that occur at or after q in the order along γ. We define $L^-(q)$ and $L^+(q)$ analogously (see Fig. 1).

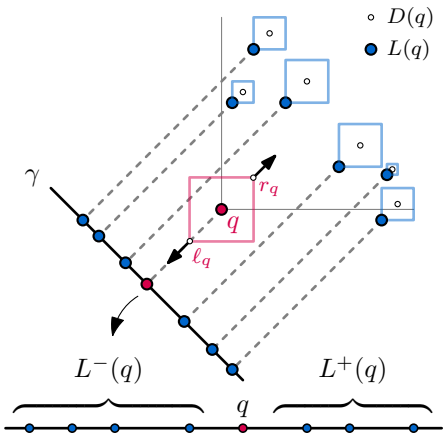


Figure 1 The projection of the square centers and relevant corners onto line γ.

▶ **Observation 2.2.** Let $p \in D(q)$ be a point dominating point q, and let t^* be the first time at which $r = r_q(t^*)$ dominates $\ell = \ell_p(t^*)$. We then have that

- $\ell_x < r_x$ and $\ell_y = r_y$ if and only if $p \in D^-(q)$;
- $\ell_x = r_x$ and $\ell_y \leq r_y$ if and only if $p \in D^+(q)$.

Observation 2.2 implies that the points p in $D^-(q)$ will start to intersect $\square_q$ at some time t^* because the bottom left vertex ℓ_p of $\square_p$ will enter $\square_q$ through the top edge, whereas the bottom left vertex of the (squares of the) points in $D^+(q)$ will enter $\square_q$ through the right edge. We thus obtain the following result.

▶ **Lemma 2.3.** *Let t^* be the time that a square $\square_p$ of a point $p \in D(q)$ touches $\square_q$. We have*

(i) $r_q(t^*)_y = \ell_p(t^*)_y$, *and $\ell_p(t^*)$ is the point with minimum y-coordinate among the points in $L^-(q)(t^*)$ at time t^** *if $p \in D^-(q)$, and*

(ii) $r_q(t^*)_x = \ell_p(t^*)_x$, *and $\ell_p(t^*)$ is the point with minimum x-coordinate among the points in $L^+(q)(t^*)$ at time t^** *if $p \in D^+(q)$.*

3 A Kinetic Data Structure for Growing Squares

We describe a data structure that can detect intersections between all pairs of squares $\square_p, \square_q$ in P such that $p \in D^+(q)$. We build an analogous data structure for $p \in D^-(q)$, and then use four copies of these data structures, one for each quadrant, to detect the first intersection among all pairs of squares.

3.1 The Data Structure

Our data structure consists of two three-layered trees T^L and T^R, and a set of certificates linking nodes from T^L and T^R. These trees essentially form two 3D range trees on the centers of the squares in P, taking the third coordinate p_γ of each point to be their rank in the order (from left to right) along the line γ. The third layer of T^L doubles as a kinetic tournament tracking the bottom left vertices of squares. Similarly, T^R tracks the top right vertices of the squares.

The Layered Trees. The tree T^L is a 3D-range tree storing the center points in P. Each layer is implemented by a BB[α] tree [8], and each node μ corresponds to a canonical subset P_μ of points stored in the leaves of the subtree rooted at μ. The points are ordered on x-coordinate first, then on y-coordinate, and finally on γ-coordinate. Let L_μ denote the set of bottom left vertices of squares corresponding to the set P_μ, for some node μ.

Consider the associated structure X_v^L of some secondary node v. We consider X_v^L as a kinetic tournament on the x-coordinates of the points L_v [1]. More specifically, every internal node $w \in X_v^L$ corresponds to a set of points P_w consecutive along the line γ. Since the γ-coordinates of a point p and its bottom left vertex ℓ_p are equal, this means w also corresponds to a set of consecutive bottom left vertices L_w. Node w stores the vertex ℓ_p in L_w with minimum x-coordinate, and will maintain certificates that guarantee this [1].

The tree T^R has the same structure as T^L: it is a three-layered range tree on the center points in P. The difference is that a ternary structure X_v^R, for some secondary node v, forms a kinetic tournament maintaining the maximum x-coordinate of the points in R_v, where R_v are the top right vertices of the squares (with center points) in P_v. Hence, every ternary node $z \in X_v^R$ stores the vertex r_q with maximum x-coordinate among R_v. Let $\mathcal{X}^L$ and $\mathcal{X}^R$ denote the set of all kinetic tournament nodes in T^L and T^R, respectively.

Linking the Trees. Next, we describe how to add *linking certificates* between the kinetic tournament nodes in the trees T^L and T^R that guarantee the squares are disjoint. More specifically, we describe the certificates, between nodes $w \in \mathcal{X}^L$ and $z \in \mathcal{X}^R$, that guarantee that the squares $\square_p$ and $\square_q$ are disjoint, for all pairs $q \in P$ and $p \in D^+(q)$.

Consider a point q. There are $O(\log^2 n)$ nodes in the secondary trees of T^L, whose canonical subsets together represent exactly $D(q)$. For each of these nodes v we can then find $O(\log n)$ nodes in X_v^L representing the points in $L^+(q)$. So, in total q is *interested in* a set $Q^L(q)$ of $O(\log^3 n)$ kinetic tournament nodes. It now follows from Lemma 2.3 that if we were to add certificates certifying that r_q is left of the point stored at the nodes in $Q^L(q)$ we can detect when $\square_q$ intersects with a square of a point in $D^+(q)$. However, as there may

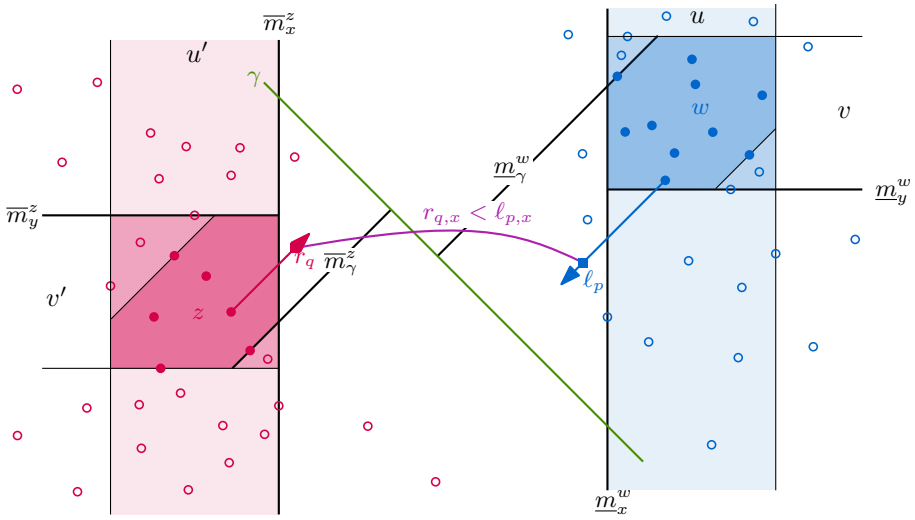


Figure 2 The points $\overline{m}^z$ and $\underline{m}^w$ are defined by a pair of nodes $z \in \mathcal{X}^R_{v'}$, with $v' \in T_{u'}$, and $w \in X^L_v$, with $v \in T_u$. If $w \in Q^L(\overline{m}^z)$ and $z \in Q(\underline{m}^w)$ then we add a linking certificate between the rightmost upper right-vertex r_q, $q \in P_z$, and the leftmost bottom left vertex ℓ_p, $p \in P_w$.

be many points q interested in a particular kinetic tournament node w, we cannot afford to maintain all of these certificates. The main idea is to represent all of these points q by a number of canonical subsets of nodes in T^R, and add certificates to only these nodes.

Consider a point p. Symmetric to the above construction, there are $O(\log^3 n)$ nodes in kinetic tournaments associated with T^R that together exactly represent the (top right corners of) the points q dominated by p and for which $p \in D^+(q)$. Let $Q^R(p)$ denote this set of kinetic tournament nodes.

Next, we extend the definitions of Q^L and Q^R to kinetic tournament nodes. To this end, we first associate each kinetic tournament node with a (query) point in $\mathbb{R}^3$. Consider a kinetic tournament node w in a tournament X^L_v, and let u be the node in the primary T^L for which $v \in T_u$. Let $\underline{m}^w = (\min_{a \in P_u} a_x, \min_{b \in P_v} b_y, \min_{c \in P_w} c_\gamma)$ be the point associated with w (note that we take the minimum over different sets P_u, P_v, and P_w for the different coordinates), and define $Q^R(w) = Q^R(\underline{m}^w)$. Symmetrically, for a node z in a tournament X^R_v, with $v \in T_u$ and $u \in T^R$, we define $\overline{m}^z = (\max_{a \in P_u} a_x, \max_{b \in P_v} b_y, \max_{c \in P_z} c_\gamma)$ and $Q^L(z) = Q^L(\overline{m}^z)$. See Fig. 2.

We now add a linking certificate between every pair of nodes $w \in \mathcal{X}^L$ and $z \in \mathcal{X}^R$ for which (i) w is a node in the canonical subset of z, that is $w \in Q^L(z)$, *and* (ii) vice versa, $z \in Q^R(w)$. Such a certificate will guarantee that the point r_q currently stored at z lies left of the point ℓ_p stored at w.

▶ **Lemma 3.1.** *Every kinetic tournament node is involved in $O(\log^3 n)$ linking certificates, and thus every point p is associated with at most $O(\log^6 n)$ certificates.*

We now argue that we can still detect the first upcoming intersection.

▶ **Lemma 3.2.** *Consider two sets of elements, say blue elements B and red elements R, stored in the leaves of two binary search trees T^B and T^R, respectively, and let $p \in B$ and $q \in R$, with $q < p$, be leaves in trees T^B and T^R. There is a pair of nodes $b \in T^B$ and $r \in T^R$, such that (i) $p \in P_b$ and $b \in C(T^B, [\max P_r, \infty))$, and (ii) $q \in P_r$ and $r \in C(T^R, (-\infty, \min P_b])$, where $C(T^S, I)$ denotes the minimal set of nodes in T^S whose canonical subsets together represent exactly the elements of $S \cap I$.*

▶ **Lemma 3.3.** *Let $\square_p$ and $\square_q$, with $p \in D^+(q)$, be the first pair of squares to intersect, at some time t^*. There is a pair of nodes w, z that have a linking certificate that fails at time t^*.*

From Lemma 3.3 it follows that we can now detect the first intersection between a pair of squares $\square_p, \square_q$, with $p \in D^+(q)$. We define an analogous data structure for when $p \in D^-(q)$. Following Lemma 2.3, the kinetic tournaments will maintain the vertices with minimum and maximum y-coordinate for this case. We then again link up the kinetic tournament nodes in the two trees appropriately.

Space Usage. Our trees T^L and T^R are range trees in $\mathbb{R}^3$, and thus use $O(n \log^2 n)$ space. However, it is easy to see that this is dominated by the space required to store the certificates. For all $O(n \log^2 n)$ kinetic tournament nodes we store at most $O(\log^3 n)$ certificates (Lemma 3.1), and thus the total space used by our data structure is $O(n \log^5 n)$. In the full version [4], we show that the number of certificates that we maintain (and thus the space used by our data structure) is actually only $O(n(\log n \log\log n)^2)$.

3.2 Inserting or Deleting a Square

At an insertion or deletion of a square $\square_p$ we proceed in three steps. (1) We update T^L and T^R, restoring range tree properties, and ensure that the ternary data structures are correct kinetic tournaments. (2) For each kinetic tournament node in $\mathcal{X}^L$ affected by the update, we query T^R to find a new set of linking certificates. We update $\mathcal{X}^R$ analogously. (3) We update the global event queue.

▶ **Lemma 3.4.** *Inserting or deleting a square in T^L takes $O(\log^3 n)$ amortized time.*

Clearly we can update T^R in $O(\log^3 n)$ amortized time as well. Next, we update the linking certificates. We say that a kinetic tournament node w in T^L is *affected by* an update if (i) the update added or removed a leaf node in the subtree rooted at w, (ii) node w was involved in a tree rotation, or (iii) w occurs in a newly built associated tree X^L_v (for some node v). Let $\mathcal{X}^L_i$ denote the set of nodes affected by update i ($\mathcal{X}^R_i$ of T^R is defined analogously). For each node $w \in \mathcal{X}^L_i$, we query T^R to find the set of $O(\log^3 n)$ nodes whose canonical subsets represent $Q^R(w)$. For each node z in this set, we test if we have to add a linking certificate between w and z. As we show next, this takes constant time for each node z, and thus $O(\sum_i |\mathcal{X}^L_i| \log^3 n)$ time in total, for all nodes w (analogously for $\mathcal{X}^R_i$).

We have to add a link between a node $z \in Q^R(w)$ and w if and only if we also have $w \in Q^L(z)$. We test this as follows. Let v be the node whose associated tree X^L_v contains w, and let u be the node in T^L whose associated tree contains v. We have that $w \in Q^L(z)$ if and only if $u \in C(T^L, [\overline{m}^z_x, \infty))$, $v \in C(T_u, [\overline{m}^z_y, \infty))$, and $w \in C(X^L_v, [\overline{m}^z_\gamma, \infty))$. We can test each of these conditions in constant time:

▶ **Observation 3.5.** Let q be a query point in $\mathbb{R}^1$, let w be a node in a binary search tree T, and let $x_p = \min P_p$ of the parent p of w in T, or $x_p = -\infty$ if no such node exists. We have that $w \in C(T, [q, \infty))$ if and only if $q \leq \min P_w$ and $q > x_p$.

Finally, we delete all certificates involving no longer existing nodes from our global event queue, and replace them by all newly created certificates. This takes $O(\log n)$ time per certificate. We charge the cost of deleting a certificate to when it gets created. Since every node w affected creates at most $O(\log^3 n)$ new certificates, all that remains is to bound the total number of affected nodes. Here we can use basically the same argument as when bounding the update time.

▶ **Lemma 3.6.** *Inserting a disjoint square into P, or deleting a square from P takes $O(\log^7 n)$ amortized time.*

3.3 Running the Simulation

All that remains is to analyze the number of events processed, and the time to do so. Since each failure of a linking certificate produces an intersection, and thus an update, the number of such events is at most the number of updates. To bound the number of events created by the tournament trees we use an argument similar to that of Agarwal et al. [1].

▶ **Theorem 3.7.** *We can maintain a set P of n disjoint growing squares in a fully dynamic data structure such that we can detect the first time that a square $\square_q$ intersects with a square $\square_p$, with $p \in D^+(q)$. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time. For a sequence of m operations, the structure processes a total of $O(m\alpha(n)\log^3 n)$ events in a total of $O(m\alpha(n)\log^7 n)$ time.*

To simulate the process of growing the squares in P, we now maintain eight copies of the data structure from Theorem 3.7: two data structures for each quadrant (one for D^+, the other for D^-). Using these data structures we obtain the following agglomerative glyph clustering solution.

▶ **Theorem 3.8.** *Given a set of n initial square glyphs P, we can compute an agglomerative clustering of the squares in P in $O(n\alpha(n)\log^7 n)$ time using $O(n(\log n \log\log n)^2)$ space.*

References

1 P. K. Agarwal, H. Kaplan, and M. Sharir. Kinetic and Dynamic Data Structures for Closest Pair and All Nearest Neighbors. *ACM Transactions on Algorithms*, 5(1):4:1–4:37, 2008.

2 H.-K. Ahn, S. W. Bae, J. Choi, M. Korman, W. Mulzer, E. Oh, J.-W. Park, A. van Renssen, and A. Vigneron. Faster Algorithms for Growing Prioritized Disks and Rectangles. In *Proceedings of the 28th International Symposium on Algorithms and Computation*, pages 3:1–3:13, 2017.

3 G. Alexandron, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for convex hulls and upper envelopes. *Computational Geometry: Theory and Applications*, 36(2):144–1158, 2007.

4 T. Castermans, B. Speckmann, F. Staals, and K. Verbeek. Agglomerative clustering of growing squares. *ArXiv e-prints*, 2017. `arXiv:1706.10195`.

5 T. Castermans, B. Speckmann, K. Verbeek, M. A. Westenberg, A. Betti, and H. van den Berg. GlamMap: Geovisualization for e-Humanities. In *Proceedings of the 1st Workshop on Visualization for the Digital Humanities*, 2016.

6 S. Funke, F. Krumpe, and S. Storandt. Crushing Disks Efficiently. In *Proceedings of the 27th International Workshop on Combinatorial Algorithms*, pages 43–54, 2016.

7 S. Funke and S. Storandt. Parametrized Runtimes for Ball Tournaments. In *Proceedings of the 33rd European Workshop on Computational Geometry*, pages 221–224, 2017.

8 J. Nievergelt and E. M. Reingold. Binary Search Trees of Bounded Balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

A New Lower Bound on the Maximum Number of Plane Graphs using Production Matrices

Clemens Huemer[1], Alexander Pilz[2], and Rodrigo I. Silveira[1]

1 **Departament de Matemàtiques, Universitat Politècnica de Catalunya, Barcelona, Spain.**
{clemens.huemer,rodrigo.silveira}@upc.edu

2 **Department of Computer Science, ETH Zürich, Switzerland.**
alexander.pilz@inf.ethz.ch

Abstract

We use the concept of production matrices to show that there exist sets of n points in the plane that admit $\Omega(41.77^n)$ crossing-free geometric graphs. This improves the previously best known bound of $\Omega(41.18^n)$ by Aichholzer et al. (2007).

1 Introduction

A *geometric graph* on a set S of n labeled points in the Euclidean plane is a graph with vertex set S in which an edge is represented by a straight line segment between the corresponding vertices. In this work, we are interested in the number of *crossing-free* geometric graphs on a set of n points, i.e., geometric graphs in which all segments are interior-disjoint. It is easy to see that, for any n points, this number is at least exponential in n. In 1982, Ajtai et al. [2] showed that the upper bound on this number is also exponential. Currently, it is known that any set of n points admits not more than $O(187.53^n)$ crossing-free graphs [13]. While it is known that the number of crossing-free graphs is minimized if the point set is in convex position [1], not much is known about sets maximizing this number. The best known example by now is the so-called *double-zig-zag chain* [1], with $\Omega(41.18^n)$ crossing-free graphs. As usual, such lower-bound constructions rely on describing a family of point sets with convenient structural properties. In this paper, we improve this bound by showing that another well-known family of point sets, a generalization of the double-zig-zag chain, admits $\Omega(41.77^n)$ crossing-free graphs. This generalization has also been used for similar bounds on triangulations [5], but the number of general crossing-free graphs on this configuration was not known. The method that allows us to analyze these point sets is the use of *production matrices*, which we consider interesting on its own.

This method works by implicitly arranging the graphs in a *generating tree*, describing a rule to produce a graph from one on fewer points. Consider a partition of the set of graphs on $i \leq n$ points into n parts according to their degree at a special *root vertex*, and represent the cardinality of each part in a vector $\vec{v}^i$. The first element of $\vec{v}^i$ is the number of graphs with the root vertex having degree 0, the second one that of graphs with root vertex with degree 1, and so on. We then devise how to generate graphs on $i + c$ points with a new root

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922.

C. H. and R. S. were also supported by projects MINECO MTM2015-63791-R and Gen. Cat. 2017SGR1336 and 2017SGR1640, respectively. R. S. was further supported by MINECO through the Ramón y Cajal program. A. P. is supported by a Schrödinger fellowship of the Austrian Science Fund (FWF): J-3847-N35.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

vertex from the graphs counted in $\vec{v}^i$, and again give the cardinalities of their parts in a vector $\vec{v}^{i+c}$ (for some small positive number c). Our point sets will allow us to devise an $n \times n$ *production matrix* A such that $\vec{v}^{i+c} = A\vec{v}^i$. We obtain the number of graphs on n vertices in $\vec{v}^n = A^j\vec{v}^{n_0}$ from the graphs on a constant number n_0 of vertices, with $j = (n - n_0)/c$. We can then use the Perron–Frobenius theorem to obtain a lower bound on the elements of A^j when j tends to infinity by approximating the largest eigenvalue of the matrix. This gives us a lower bound on the number of crossing-free graphs on such a point set.

For points in convex position, generating trees have been described for triangulations [10], spanning trees [6], and other crossing-free graphs [7]. They are the basis of the ECO method [3]. The term *production matrix* was introduced in [4], a similar concept is known as *AGT matrix* [11]. Together with Seara, the authors already addressed characteristic polynomials of production matrices for geometric graphs [8].

In the next section, we define the family of point sets used, and provide production matrices to count subgraphs in its different parts. In Section 3, we argue that bounds on the Perron roots of the matrices give us a lower bound on the number of crossing-free graphs.

2 Generalized double zig-zag chains and the new lower bound

Basically, our point sets will be described as sequences $(s_1, \ldots, s_n)$. Consider any graph G drawn on the first $i + 1$ vertices. If we replace every edge $s_j s_{i+1}$ by the edge $s_j s_i$ for all $j \leq i + 1$ (and disregard duplicates and loops), we obtain a graph G' that we call the *parent* of G. Our sets will be such that G' is crossing-free. In the other direction, we can select some edges incident to s_i in G' and replace them by edges incident to s_{i+1} in a way that G' is the parent of the new graph $\tilde{G}$, and such that $\tilde{G}$ is crossing-free. We say that G' *produces* $\tilde{G}$, and the edges incident to s_{i+1} are *inherited*. The degree of s_i in G determines how many graphs can be produced from it. For our construction, s_i is thus the root vertex, and the vector $\vec{v}^i$ contains the number of graphs with root vertex s_i of degree j, for $0 \leq j \leq n$. While this captures the basic idea of our proofs, we will actually have to use more involved constructions, in which we add a constant number of points at once and add edges, some inherited, and some not, in a well-defined, local way.

2.1 The generalized double-zig-zag chain

Let Z_k be a set of $n = 2z$ points with $z \equiv 1 \pmod{(k+1)}$ that is arranged in the following way. Consider two x-monotone circular arcs facing each other as in Fig. 1, such that each point on one arc can *see* each point on the other arc (where two points can see each other if the interior of the line segment connecting them does not intersect one of the arcs). On each arc, we place $\lceil n/(k+1) \rceil$ points. Consider the segment between two consecutive such points s and t on the lower arc. We now place a "flat" circular arc between s and t with circle center above the arc, and place k points on it; here, flat means that moving the center of the arc up (and thus the k points on it) does not change the set of crossing-free graphs drawable on Z_k. We call the group formed by s, t, and the k points in between them a *pocket*. We place k such points between each pair of consecutive points of the lower arc (obtaining the *lower chain*), and also in an analogous way on the upper arc (resulting in the *upper chain*). See Figure 1 for an example of Z_2, where each pocket consists of four points.

The points along the lower arc, including pockets, are labeled, from left to right, $p_1, \ldots, p_z$, and those on the upper arc $q_1, \ldots, q_z$. Observe that the segment between any two consecutive points $p_i p_{i+1}$ is not crossed by any other segment between two points of the set, and thus can co-exist with any other edge in a crossing-free graph. For this reason, these edges will be

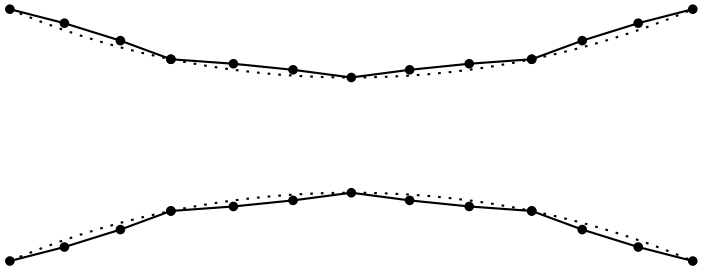


Figure 1 A generalized double-zig-zag chain Z_2. The arcs for the construction are dotted, the solid edges are not crossed by any segment between two points.

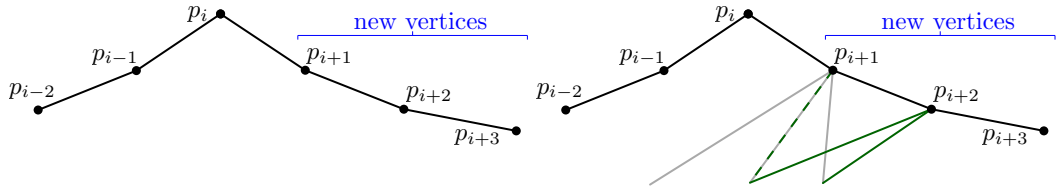


Figure 2 Part of an almost convex chain with two interior vertices (i.e., $k = 2$). Vertices p_{i-2} and p_{i+1} are leading vertices. The other vertices are regular. Since p_{i+2} is a regular vertex, any edge incident to p_{i+2} present in a plane graph can be obtained by inheriting an edge from the previous vertex p_{i+1}. The example shows p_{i+2} inheriting two edges from p_{i+2}. The last inherited edge (dashed) may also be kept at p_{i+1} without influencing the degree of p_{i+2}.

disregarded first in our counting, and will be considered in the end by multiplying by a factor of 2^n. Also note that the construction consists of two almost convex polygons [9]. Therefore we focus on counting the graphs with edges below the path $(p_1, \ldots, p_z)$ (and, symmetrically, above the path $(q_1, \ldots, q_z)$) called the *outer part*, and edges which connect vertices of the two paths, which are in the *inner part*. Our bound is obtained on Z_2.

2.2 Production matrices for the outer part

In this section we deduce matrices to count the number of plane graphs with edges below the path $(p_1, \ldots, p_z)$, as in Figure 2. Recall that a chain is composed of a series of pockets; each pocket forms a reflex chain of four vertices. The first and last vertices are convex, while the two middle ones are reflex. The first (say, with smallest index) reflex vertex is called the *leading* vertex of the chain. All other vertices we call *regular*.

We will present a matrix to count the number of plane graphs after adding one whole pocket. This matrix will be the product of three matrices, one related to each new vertex of the pocket $p_{i+1}, p_{i+2}, p_{i+3}$ (recall that p_i coincides with the last vertex of the previous pocket).

2.2.1 Matrix for regular vertices

Consider a regular vertex like p_{i+2} (refer to Figure 2). Assume that the vector $\vec{v}^{i+1}$, containing the number of plane graphs for each possible degree of p_{i+1}, is known. The plane graphs where p_{i+2} has degree 0 are equal to all the graphs counted in $\vec{v}^{i+1}$. This gives a first row of 1s in the matrix. If p_{i+2} has degree 1, it needs to inherit one edge from p_{i+1}. If the degree of p_{i+1} is 0, this is not possible, thus we get a zero in the first column of the second row. As soon as p_{i+1} has degree at least 1, p_{i+2} can inherit one edge from p_{i+1}. Moreover, there is

$$R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 1 & 1 \\ 3 & 4 & 3 & 3 & 3 & 3 \\ 1 & 4 & 5 & 4 & 4 & 4 \\ 0 & 1 & 4 & 5 & 4 & 4 \\ 0 & 0 & 1 & 4 & 5 & 4 \end{pmatrix}$$

Table 1 Matrices for computing the outer part, for $n = 6$.

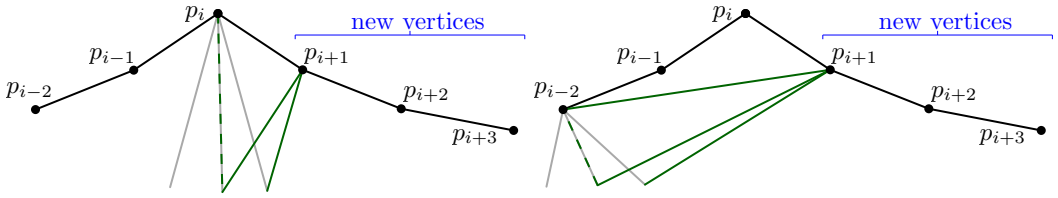


Figure 3 When edges $p_{i+1}p_{i-2}$ and $p_{i+1}p_{i-3}$ are not included, p_{i+1} can inherit edges from p_i (left). The example shows p_{i+1} inheriting two edges from p_i. The last inherited edge (dashed) may be kept without influencing the degree of p_{i+1}. The case when edges $p_{i+1}p_{i-2}$ or $p_{i+1}p_{i-3}$ are included is shown to the right. In the example $p_{i+1}p_{i-2}$ is included, and p_{i+1} inherits two edges from p_{i-2}. The dashed edge can be optionally kept.

the option of keeping (a copy of) the inherited edge incident to p_{i+1} without creating any crossing. In total, for each graph in which p_{i+1} has degree at least one, that gives two ways for making p_{i+2} have degree 1. Thus the rest of the row is made of 2s.

The following rows are analogous, shifted by one column every time: in order for p_{i+2} to have degree k, k edges need to be inherited from p_{i+1}, thus the minimum degree for p_{i+1} is k. Since we can always choose to keep the last inherited edge incident to p_{i+1}, we get 2 options every time. This results in matrix R in Table 1. Exactly the same matrix applies to p_{i+3}.

2.2.2 Matrix for leading vertices

Leading vertices like p_{i+1} in Figure 2 require a different approach, as there are edges incident to p_{i+1} that cannot be obtained by inheriting from p_i (i.e., edges $p_{i+1}p_{i-1}$, $p_{i+1}p_{i-2}$, $p_{i+1}p_{i-3}$, as p_ip_{i-1}, p_ip_{i-2}, p_ip_{i-3} are not in the outer part). To take this into account, we consider two cases, depending on whether edges $p_{i+1}p_{i-2}$ or $p_{i+1}p_{i-3}$ are included or not.

Case 1: Edges $p_{i+1}p_{i-2}$ and $p_{i+1}p_{i-3}$ are not included. When $p_{i+1}p_{i-2}$ and $p_{i+1}p_{i-3}$ are not included, p_{i+1} can inherit edges from p_i (notice that all edges from p_i cross $p_{i+1}p_{i-2}$ and $p_{i+1}p_{i-3}$). See Figure 3 (left).

The plane graphs where p_{i+1} has degree zero are, as before, all the ones counted in $\vec{v}^i$, thus this gives a first row of 1s in the matrix. If p_{i+1} has degree one, it either inherited one edge from p_i or is connected to p_{i-1}. Thus if p_i has degree zero, there is only one possibility: using edge $p_{i+1}p_{i-1}$. That gives a 1 in the first column of the second row. As soon as p_i has degree at least one, p_{i+1} can inherit one edge from p_i, with the additional option of keeping the inherited edge incident to p_i; note that in this case using also $p_{i+1}p_{i-1}$ is not considered because that would increase the degree of p_{i+1} by one. In total, for each possible degree of p_i, that gives two ways for making p_{i+1} have degree one. Thus the rest of the row is made of 2s.

The following rows are analogous. Consider the kth row ($k \geq 3$). If p_i has degree $k-2$ or less, it is impossible for p_{i+1} to obtain degree k. When p_i has degree $k-1$, there is one possibility: to inherit all edges incident to p_i and add edge $p_{i+1}p_{i-1}$. If p_i has degree at

$$Q = \begin{pmatrix} 6 & 0 & 0 & 0 & 0 & 0 \\ 10 & 6 & 0 & 0 & 0 & 0 \\ 5 & 10 & 6 & 0 & 0 & 0 \\ 1 & 5 & 10 & 6 & 0 & 0 \\ 0 & 1 & 5 & 10 & 6 & 0 \\ 0 & 0 & 1 & 5 & 10 & 6 \end{pmatrix} \quad F = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 \\ 0 & 3 & 2 & 2 & 2 & 2 \\ 0 & 0 & 3 & 2 & 2 & 2 \\ 0 & 0 & 0 & 3 & 2 & 2 \\ 0 & 0 & 0 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

Table 2 Matrices for computing the inner part, for $n = 6$.

least k, then p_{i+1} can inherit k edges from p_i, with the additional option of keeping the last inherited edge incident to p_i, giving two options for every possible degree of p_i. This leads to the matrix C in Table 1.

Case 2: At least one of $p_{i+1}p_{i-2}$ and $p_{i+1}p_{i-3}$ is included. In this case we proceed essentially by using p_{i-2} as "previous" vertex, but considering also the three special edges $\{p_{i+1}p_{i-1}, p_{i+1}p_{i-2}, p_{i+1}p_{i-3}\}$, which cannot be inherited from p_{i-2}. Refer to Figure 3 (right). We defer the details to the full version. The result for this case is matrix X, shown in Table 1.

2.3 Production matrices for the inner part

The number of graphs on the inner part can be bounded similar to [1]. However, in the full version, we show how to obtain a lower bound using production matrices, based on two additional matrices, Q and F shown in Table 2.

2.4 Putting things together

The final production matrix for the outer part is obtained by combining matrices R, C, and X. For each of the two regular vertices it is enough to multiply the previous vector by R. For the leading vertex we need to combine the two cases, thus we need to add up C and X. However, the reasoning in X uses p_{i-2} instead of the previous vertex p_i. Thus prior to multiplying by X, we need to recover the vector corresponding to p_{i-2}: for this we first multiply twice by R^{-1}. Thus the final combined matrix for the outer part is $R^2(C + X \cdot R^{-2})$. In the full version we show that a lower bound on the number of plane graphs in the inner part is given by the combined matrix $(FFR + 2R)Q$.

3 A lower bound using the eigenvalue

All our production matrices are non-negative. The zero entries are exactly those below a sub-diagonal. Thus, they are irreducible and primitive (Frobenius' test for primitivity holds, cf. [12, p. 678]). Let A be a production matrix of fixed size $m \times m$. We know therefore that

$$\lim_{n\to\infty} \left(\frac{A}{r}\right)^n = \frac{\vec{p}\vec{q}^T}{\vec{q}^T\vec{p}} > 0 \ ,$$

where $\vec{p}$ and $\vec{q}$ are the Perron vectors of A and A^T, respectively, and r is the Perron root (i.e., largest eigenvalue) of A [12, p. 674]. As these values are constant and each entry of A^n is in $\Theta(r^n)$, this provides a means of obtaining the asymptotic number of elements constructed by the production matrix: multiplying the initial degree vector with A^i gives the degree vector for $ci < m$ points. However, there is one caveat. The exponent n tends to infinity, and we thus cannot use this to argue about matrices of size n. The matrix size must be fixed. However, for obtaining lower bounds, we can take the nth power of a $(m \times m)$ production

matrix for some constant m to obtain a lower bound on the number of graphs on n vertices. In the first iteration where we add a point larger than the size of the matrix, we do not count some graphs with high degree at the last point. These are also not taken into account in the next iteration etc., where we also produce graphs of smaller degree at the last point. Still, the degree vector gives a lower bound on the number of graphs. We may thus obtain the Perron root r of a constant-size production matrix and know that the number of graphs on n vertices in that class is in $\Omega(r^n)$ for all our considered instances. For the matrix $R^2(C + X \cdot R^{-2})$, the largest eigenvalue is at least 124.22239555, when taking the constant-size production matrix large enough. For the inner part, the largest eigenvalue of the matrix $(FFR + 2R)Q$ is at least 5380.90657056 (see the full version). Accounting for the 2^n ways to add edges along the chains, we get $\Omega((\sqrt[3]{124.22239555} \cdot \sqrt[6]{5380.90657056} \cdot 2)^n) = \Omega(41.773981586^n)$ crossing-free graphs (eigenvalues computed using *Mathematica 11.2* with $m = 1024$).

4 Conclusion

We slightly improved the current lower bound on the maximum number of crossing-free geometric graphs on n points using production matrices. Applying production matrices to families of well-structured point sets appears to be an easy way of obtaining bounds for certain types of graphs (e.g., triangulations). It is also easy to mix the pocket sizes. However, our current approach results in an increasing number of cases when considering generalized double-zig-zag chains with larger pockets.

References

1 O. Aichholzer, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber. On the number of plane geometric graphs. *Graphs Combin.*, 23:67–84, 2007.

2 M. Ajtai, V. Chvátal, M.M. Newborn, and E. Szemerédi. Crossing-free subgraphs. In *Theory and Practice of Combinatorics*, pages 9–12. North-Holland, 1982.

3 E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani. ECO: a methodology for the enumeration of combinatorial objects. *J. Differ. Equations Appl.*, 5(4-5):435–490, 1999.

4 E. Deutsch, L. Ferrari, and S. Rinaldi. Production matrices. *Adv. in Appl. Math.*, 34(1):101–122, 2005.

5 A. Dumitrescu, A. Schulz, A. Sheffer, and Cs. D. Tóth. Bounds on the maximum multiplicity of some common geometric graphs. *SIAM J. Discrete Math.*, 27(2):802–826, 2013.

6 M. C. Hernando, F. Hurtado, A. Márquez, M. Mora, and M. Noy. Geometric tree graphs of points in convex position. *Discrete Appl. Math.*, 93(1):51–66, 1999.

7 C. Huemer, A. Pilz, C. Seara, and R. I. Silveira. Production matrices for geometric graphs. *Electr. Notes Discrete Math.*, 54:301–306, 2016.

8 C. Huemer, A. Pilz, C. Seara, and R. I. Silveira. Characteristic polynomials of production matrices for geometric graphs. *Electr. Notes Discrete Math.*, 61:631–637, 2017.

9 F. Hurtado and M. Noy. Counting triangulations of almost-convex polygons. *Ars Comb.*, 45:169–179, 1997.

10 F. Hurtado and M. Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Comput. Geom.*, 13(3):179–188, 1999.

11 D. Merlini and M. C. Verri. Generating trees and proper Riordan arrays. *Discrete Math.*, 218(1–3):167–183, 2000.

12 C. D. Meyer. *Matrix analysis and applied linear algebra.* SIAM, Philadelphia, 2000.

13 M. Sharir and A. Sheffer. Counting plane graphs: Cross-graph charging schemes. *Combin. Probab. Comput.*, 22(6):935–954, 2013.

Optimal Algorithms for Compact Linear Layouts

Wouter Meulemans, Willem Sonke, Bettina Speckmann, Eric Verbeek, and Kevin Verbeek

Dep. of Mathematics and Computer Science, TU Eindhoven, The Netherlands,
[w.meulemans|w.m.sonke|b.speckmann|h.m.w.verbeek|k.a.b.verbeek]@tue.nl

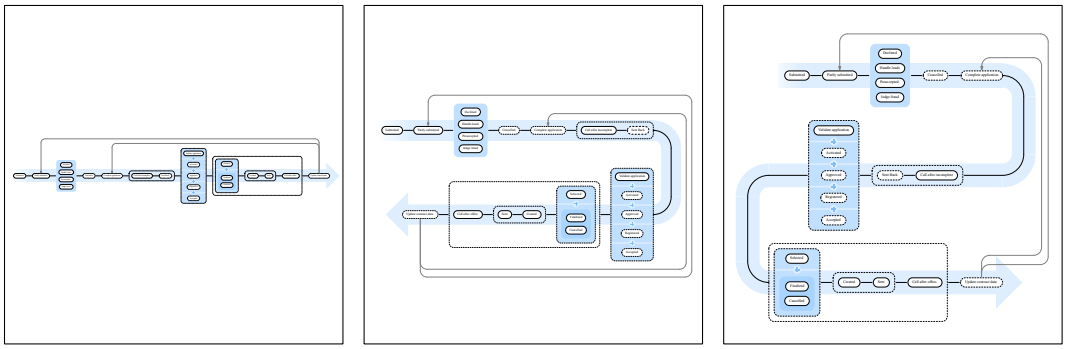


Figure 1 Optimal foldings of a linear layout with three different aspect ratios, as computed by our algorithm: process tree [8] computed from the 2012 Business Process Intelligence Challenge [1].

1 Introduction

Linear layouts are a simple and natural way to draw a graph: all vertices are placed on a single line and edges are drawn as arcs between the vertices. Despite its simplicity, a linear layout can be a very meaningful visualization if there is a particular order defined on the vertices. Common examples of such ordered—and often also directed—graphs are event sequences and processes: public transport systems tracking passenger check-in and check-out, banks checking online transactions (see Fig. 1 for an abstracted view of such a log), or hospitals recording the paths of patients through their system, to name a few. A main drawback of linear layouts are the usually (very) large aspect ratios of the resulting drawings, which prevent users from obtaining a good overview of the whole graph. In this paper we present a novel and versatile algorithm to optimally fold a linear layout of a graph such that it can be drawn effectively in a specified aspect ratio, while still clearly communicating the linearity of the layout (see Fig. 1).

Exact problem statement. We focus on the linear layout of graphs which have an order defined on their vertices. Specifically, our input consists of a graph $G = (V, E)$ with a total order on the vertices V. We are also given the desired aspect ratio ρ, or equivalently the width W_d and height H_d, of the drawing. Our goal is now to draw G as clearly as possible, in a way that communicates the total order of the vertices effectively, while minimizing the unused (empty) space in the drawing. In a classic graph drawing setting vertices are points in the plane and edges are drawn arbitrarily close to each other as (thin) lines. In any practical scenario, however, vertices carry associated data, often visualized as labels, and lines need to be spaced well for readability. We capture both constraints by associating a *block* B_i of a specified width and height with each vertex v_i. This block represents the area needed to draw the vertex v_i, which may represent the size of the corresponding label, or even a (recursive) drawing of a subgraph represented by v_i. B_i also reserves the necessary space to draw the edges surrounding v_i clearly.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Results. We describe an algorithm which optimally "folds" a given input graph (with a specified order and vertex blocks) for a desired target aspect ratio. The main ingredient of our approach is an algorithm which computes an optimal partition of the input graph and its associated blocks over the various folds, without changing the order. That is, we are solving a packing problem (packing blocks onto rows) while respecting a given order of the blocks. Our algorithm works at interactive speed for reasonably sized layouts.

Related work. A *linear layout* of a graph G is an ordering on its vertices. A linear layout can be visualized by drawing all vertices on a line, in the given order, and drawing the edges as arcs on one side of the line. A *book embedding* is a linear layout of which the edges are partitioned into a number of sets (called *pages*) of non-crossing edges. For any graph the minimum number of pages needed for a book embedding (over all possible linear layouts) is called the *book thickness*. Determining the book thickness of a graph is NP-hard, and the problem stays NP-hard even if we are given a fixed linear layout [4]. For a more complete overview of linear layouts, we refer to the survey by Dujmović and Wood [3].

Packing rectangles has been an active area of research in both algorithms and operations research. For our purposes two types of packing problems are particularly relevant: (two-dimensional) *bin packing* and *strip packing*. Bin packing is already NP-hard in one dimension (see for example [6]), which implies that both two-dimensional bin packing and strip packing are NP-hard as well [7].

Generally packing problems allow reordering of the blocks, while we have to display the blocks in order, which significantly reduces the complexity of the problem. There are some algorithms for on-line strip packing which preserve the order of the blocks. A natural approach here is *next-fit*, which greedily places as many blocks as possible onto a row, before moving to the next row. While there are no bounds on the quality of the solution obtained [2], it performs reasonably well in the average case [5]. To the best of our knowledge the exact variant which we are studying in this paper has not been treated in the literature yet.

2 Folding algorithm

Our strategy is to fold the linear layout into multiple rows, where the vertices are ordered alternatingly from left to right and from right to left. In other words, we assign the vertices to rows, such that all vertices in the same row are consecutive in the given order. We call this assignment a *folding* of G. We can distinguish between two types of edges: *spine edges* (those between two consecutive vertices in the order) and *connectors* (the other edges). Spine edges can be drawn along the folded path (*spine*) itself; connectors must be placed next to it.

We first consider the following problem: given a maximum width W, minimize the height H of the resulting drawing. For each vertex $v_i \in V$ we specify a *block* B_i, that represents the area needed to draw the vertex, including possibly its label. We specify a block B_i by its width w_i, its top-height h_i^T, and its bottom-height h_i^B (see Fig. 2). We separate top-height and bottom-height so that blocks do not need to be centered vertically on the spine. Our goal is now to compute a folding for the blocks B_i such that all blocks are disjoint and the total height is minimized. This packing problem is the core of our algorithm and is described in Section 2.1. Then, in Section 2.2 we show how to draw connectors and how to adapt the block sizes to create space for the connectors.

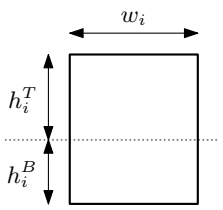


Figure 2 Width (w_i), top-height (h_i^T) and bottom-height (h_i^B) of a block, spine dotted.

2.1 Packing blocks

We first consider the problem in its full generality. That is, we can place the blocks anywhere we want along the spine, as long as the width of the drawing is at most W (see Fig. 3 (left)). In this version of the problem it can be beneficial to leave extra space between two consecutive blocks along the spine to avoid two high blocks sharing the same x-coordinate. Unfortunately we can show that minimizing the height is then NP-hard.

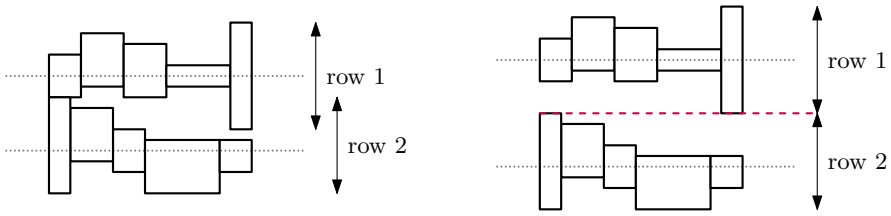


Figure 3 Packing blocks: rows can overlap vertically as long as the blocks do not overlap (left), rows cannot overlap vertically (right).

▶ **Theorem 2.1.** *If rows are allowed to overlap vertically, the problem of minimizing the drawing height is NP-hard, even if we assume that all blocks are vertically centered on the spine (their top and bottom heights are equal) and the assignment of blocks to rows is given.*

Proof sketch. By reduction from 3-Sat (see Fig. 4). We create a grid of blocks in which a column represents a variable and a pair of rows represents a clause. Between the variable columns, we put columns containing "spacer blocks" that are slightly less tall than the blocks in the variable columns. We set H and W such that spacer blocks need to be stacked on top of each other and that variable blocks on consecutive rows need to be next to each other. Necessarily, the variable blocks on even rows are on top of each other and the variable blocks on odd rows as well, forming "zigzag" configurations. A zigzag that begins on the left (on the top row of each clause) is interpreted as *true*, and one that begins on the right is interpreted as *false*. We represent each literal in a clause by a tiny block in the corresponding variable column. We place this tiny block (top row for positive and bottom row for negative literals) such that it requires additional horizontal space on a row if and only if the literal is false. Hence, to ensure that in every clause at least one literal is satisfied, we set the width W such that the rows just fit with two extra tiny blocks, but not with three. ◀

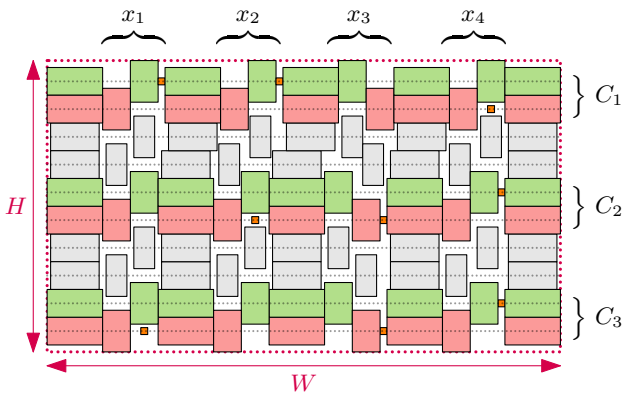


Figure 4 Instance corresponding to $\underbrace{(x_1 \vee x_2 \vee \neg x_4)}_{C_1} \wedge \underbrace{(\neg x_2 \vee \neg x_3 \vee x_4)}_{C_2} \wedge \underbrace{(\neg x_1 \vee \neg x_3 \vee x_4)}_{C_3}$.

We restrict the problem so that different rows cannot overlap vertically (see Fig. 3 (right)). In that setting there is no need to put extra space between two consecutive blocks on the same row, as the height of a row is simply determined by the maximum (top or bottom) height of the blocks in a single row. We use dynamic programming to compute the optimal folding of the blocks. To that end, we first precompute the height $H[i,j]$ $(1 \leq i \leq j \leq n)$ of a row that contains the blocks $B_i, \ldots, B_j$. Since we separate the top-height and the bottom-height of a block, we define $H[i,j] = H^T[i,j] + H^B[i,j]$, where $H^T[i,j]$ and $H^B[i,j]$ are the top-height and bottom-height of a row consisting of blocks $B_i, \ldots, B_j$, respectively. If the total width of the blocks $B_i, \ldots, B_j$ is larger than W, then we set $H^T[i,j]$ and $H^B[i,j]$ to ∞. We thus get the following for $H^T[i,j]$ (and similar for $H^B[i,j]$).

$$H^T[i,j] = \begin{cases} \max_{i \leq k \leq j} h_k^T & \text{if } \sum_{k=i}^{j} w_k \leq W; \\ \infty & \text{otherwise.} \end{cases}$$

All entries of $H[i,j]$ can be computed in $O(n^2)$ time. Next, let $T[i]$ $(0 \leq i \leq n)$ describe the minimum height of a folding involving the blocks $B_1, \ldots, B_i$. We then need to choose how many blocks we will place on the last row. This results in the following recurrence for $T[i]$.

$$T[i] = \begin{cases} 0 & \text{if } i = 0; \\ \min_{0 \leq k < i} \{T[k] + H[k+1, i]\} & \text{otherwise.} \end{cases}$$

The minimum height is then given by $T[n]$. As a result, the minimum height and the corresponding folding can be computed in $O(n^2)$ time.

2.2 Connectors

Spine edges can be drawn by adding a sufficient margin to the width of blocks and using the resulting space between blocks to draw the edges. However, connectors need to be drawn between rows, and we need to ensure that there is enough space to draw them. We can reserve this space by changing the height of the blocks in the dynamic programming formulation, to include the width of adjacent connectors.

We first assume that the connectors are properly nested. That is, if e_{ij} $(i < j)$ is a connector between B_i and B_j, and e_{kl} $(k < l)$ is another connector between B_k and B_l where $i \leq k$, then $j \leq k$ or $l \leq j$. This implies that the connectors can be drawn without crossings on one side of the spine. If the connectors are not properly nested, crossings may be needed; we discuss how to handle such crossings in Section 2.3.

We assume that all connectors are routed along the right side of the drawing. Hence on left-to-right rows, incoming and outgoing connectors go along the top of the row, while on right-to-left rows, they go along the bottom (see Fig. 5). Therefore, the height of a row can differ depending on whether it is drawn left-to-right or right-to-left. To accommodate for this we split $H[i,j]$ into two different tables: $H_{\rightarrow}[i,j]$ and $H_{\leftarrow}[i,j]$.

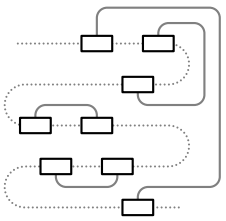


Figure 5 Connectors are routed along the right side of the drawing. (Blocks without incident connectors omitted.)

We show how to compute $H_{\rightarrow}[i,j]$ in the presence of connectors ($H_{\leftarrow}[i,j]$ can be computed similarly). We consider all connectors that start or end at a block B_k with $i \leq k \leq j$. For each such connector e_{kl} we determine the interval of blocks above which e_{kl} must be drawn. Now, for every block B_k, we add $r_k\, w_{\text{conn}}$ to h_k^T to represent the space needed by connectors above B_k, where r_k is the number of connectors that need

to be drawn above B_k and w_{conn} is the space needed per connector. We can then compute $H_{\rightarrow}[i,j]$ by taking the maximum of h_k^T over all $i \leq k \leq j$. To compute r_k efficiently for every block, note that r_k is simply the number of connector intervals that contain k. Since the intervals are nested, we can build a tree (or forest in general) on the intervals where an interval I_1 is a descendant of an interval I_2 if and only if I_1 is contained in I_2. The leaves of this tree are formed by the individual blocks. The value r_k is then simply the depth of B_k in this tree, which can easily be computed for all blocks in $O(m)$ time, where m is the number of connectors. Thus, we can compute a single entry of $H_{\rightarrow}[i,j]$ in $O(m + |j - i + 1|)$ time.

Finally, to draw the connectors that span multiple rows, we need to reserve space on the right side of the drawing. Unfortunately we cannot incorporate this into the dynamic programming algorithm. Instead we compute the nesting depth of the connectors, that is, the size of the largest set of connectors where, for every two connectors, one is always properly contained in the other. This is the largest number of connectors that we may need to draw next to each other on the right side of the drawing in the worst case. The nesting depth is independent from the folding and can hence be precomputed. We then subtract w_{conn} times the nesting depth from W before we compute the optimal folding. Note that, based on the folding, we may not need all of this additional space on the right side of the drawing. In that case we push the connectors as far to the right as possible to create some visual separation.

We note that, due to our versatile setup, we can also show additional information on connectors. In fact, we can add an additional block or even sequences of blocks on a single connector by using our algorithm recursively. We can incorporate blocks on connectors by changing the width w_{conn} of a connector. As a result, connectors can have different widths; our algorithm can easily be adapted to this scenario.

2.3 Crossing connectors

We now consider the case where the connectors are not properly nested. Here we may have connectors that cross each other, which we want to avoid as much as possible. Even if we already know the order of the vertices along the spine, minimizing the number of crossings in this situation is still known to be NP-hard (see Section 1). We therefore use a heuristic to obtain a low number of crossings: we compute a maximum set of properly-nested connectors, remove them and iterate until no connectors are left. This results in a collection of sets $E_1, \dots E_k$. We then draw each set of connectors separately as described in Section 2.2, ignoring any crossings among the different sets.

To find the largest subset of properly-nested connectors, we use the following dynamic programming formulation. We first order the connectors such that $e_{ij} < e_{kl}$ if $i < k$, or if $i = k$ and $j > l$. Let $c_1, \dots, c_m$ be the resulting ordered set of connectors, and let $f(i)$ be the index of the first connector in the order that has B_i as a starting block. Now we define $T[i,j]$ $(1 \leq i \leq m+1,\ 0 \leq j \leq n)$ as the size of the largest subset of connectors among $c_i, \dots, c_m$ that are properly-nested and all end at a block before or at B_j. Now, for every connector in order, we simply need to choose whether we want to include the connector in our set or not. We obtain the following recurrence (here we assume that $c_i = e_{kl}$).

$$T[i,j] = \begin{cases} 0 & \text{if } i = m+1; \\ T[i+1,j] & \text{if } l > j; \\ \max\{T[i+1,l] + T[f(l),j] + 1, T[i+1,j]\} & \text{otherwise.} \end{cases}$$

The size of the largest subset of properly-nested connectors is given by $T[1,n]$. It can be computed in $O(mn)$ time, where n is the number of blocks and m is the number of connectors.

2.4 Aspect ratio

So far we have presented an algorithm that, given a maximum width W, computes the minimum height $H(W)$ of a folding of the graph. Our goal is to find a folding that has a particular aspect ratio ρ: we need to find a width W such that $W/H(W) = \rho$. As $H(W)$ is non-increasing as W increases (see Fig. 6), we can use a binary search to find the width W for which $W/H(W) = \rho$. As the initial lower bound for W we take the maximum width of all blocks, because the drawing can never be narrower than that; as the upper bound we use the sum of the widths of all blocks. Since $H(W)$ is not continuous, we may not be able to obtain the exact correct aspect ratio, but the binary search will at least find the width W at which our folding algorithm jumps over the aspect ratio ρ. The resulting drawing then may have some unused height, but the drawing is as close to the correct aspect ratio as possible. More precisely, the binary search maximizes the size of the vertices (labels) in the resulting drawing. That is, if we are given a drawing area of size $W_d \times H_d$ (with aspect ratio ρ, so $W_d/H_d = \rho$), and we scale our drawing by a factor α to fit the drawing area (that is, $\alpha \cdot W \leq W_d$ and $\alpha \cdot H \leq H_d$), the binary search results in a drawing that maximizes α.

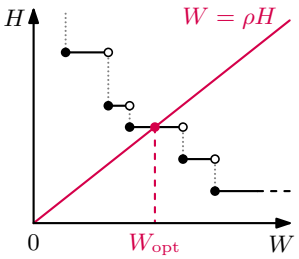


Figure 6 The height of a drawing is a descending function of its width.

Acknowledgments. The authors wish to thank Tim Ophelders for the helpful discussions. Willem Sonke, Bettina Speckmann and Kevin Verbeek are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208 (W.S. and B.S.) and no. 639.021.541 (K.V.). Wouter Meulemans is (partially) supported by the Netherlands eScience Center (NLeSC) under grant number 027.015.G02.

References

1 BPI Challenge 2012. `http://www.win.tue.nl/bpi/doku.php?id=2012:challenge`.
2 János Csirik and Gerhard J. Woeginger. On-line packing and covering problems. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms*, pages 147–177. Springer, 1998.
3 Vida Dujmović and David R. Wood. On linear layouts of graphs. *Discrete Mathematics and Theoretical Computer Science*, 6:339–358, 2004.
4 M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Matrix Analysis and Applications*, 1(2), 1980.
5 Micha Hofri. Two-dimensional packing: Expected performance of simple level algorithms. *Information and Control*, 45:1–17, 1980.
6 Bernhard Korte and Jens Vygen. *Combinatorial Optimization*, chapter Bin-Packing, pages 426–441. Springer, 2005.
7 Silvano Martello, Michele Monaci, and Daniele Vigo. An exact approach to the Strip-Packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.
8 Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2012.

A Framework for Algorithm Stability and its Application to Kinetic Euclidean MSTs *

Wouter Meulemans, Bettina Speckmann, Kevin Verbeek, and Jules Wulms

Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands
[w.meulemans | b.speckmann | k.a.b.verbeek | j.j.h.m.wulms]@tue.nl

1 Introduction

The performance of a particular algorithm is usually judged with respect to a variety of criteria, with the two most common being solution quality and running time. In the context of algorithms for time-varying data, a third important criterion is *stability*. We say that an algorithm is *stable* if small changes in the input result in small changes in the output. The stability of algorithms or methods has been well-studied in a variety of research areas, such as numerical analysis, machine learning, control systems, and topology. In contrast, the stability of combinatorial algorithms for time-varying data has received little attention in the theoretical computer science community so far. Here it is of particular interest to understand the tradeoffs between solution quality, running time, and stability. As an example, consider maintaining a minimum spanning tree of a set of moving points. If the points move, it might have to frequently change significantly. On the other hand, if we start with an MST for the input point set and then never change it combinatorially as the points move, the spanning tree we maintain is very stable – but over time it can devolve to a low quality and very long spanning tree.

Our goal, and the focus of this paper, is to understand the possible tradeoffs between solution quality and stability. This is in contrast to earlier work on stability in other research areas, such as the ones mentioned above, where stability is usually considered in isolation. Since there are currently no suitable tools available to formally analyze tradeoffs involving stability, we introduce a new analysis framework. Our framework allows for three types of stability analysis with increasing degrees of complexity: event stability, topological stability, and Lipschitz stability. We demonstrate the use of our stability framework by applying it to kinetic Euclidean minimum spanning trees. We believe that there are many interesting and relevant questions to be solved in the general area of algorithmic stability analysis and we hope that our framework is a first meaningful step towards tackling them.

Related work. Stability is a natural point of concern in more visual and applied research areas such as graph drawing, (geo-)visualization, and automated cartography. For example, in dynamic map labelling [2], the *consistent dynamic labelling* model allows a label to appear and disappear only once, making it very stable. There are very few theoretical results, with the noteworthy exception of so-called simultaneous embeddings [3] in graph drawing, which can be seen as a very restricted model of stability. However, none of these results offer any real structural insight into the tradeoff between solution quality and stability.

In computational geometry there are a few results on the tradeoff between solution quality and stability. Specifically, Durocher and Kirkpatrick [5] study the stability of centers of kinetic point sets, and define the notion of κ-stable center functions, which is closely related

* W. Meulemans and J. Wulms are (partially) supported by the Netherlands eScience Center (NLeSC) under grant number 027.015.G02. B. Speckmann and K. Verbeek are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208 and no. 639.021.541, resp.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

to our concept of Lipschitz stability. In later work [6] they consider the tradeoff between the solution quality of Euclidean 2-centers and a bound on the velocity with which they can move. De Berg *et al.* [4] show similar results in the black-box KDS model. One can argue that the KDS framework [8] already indirectly considers stability in a limited form, namely as the number of *external events*. However, the goal of a KDS is typically to reduce the running time of the algorithm, and rarely to sacrifice the running time or solution quality to reduce the number of external events.

2 Stability framework

Intuitively, we can say that an algorithm is stable if small changes in the input lead to small changes in the output. More formally, let Π be an optimization problem that, given an input instance I from a set $\mathcal{I}$, asks for a feasible solution S from a set $\mathcal{S}$ that minimizes (or maximizes) some optimization function $f\colon \mathcal{I} \times \mathcal{S} \to \mathbb{R}$. An algorithm $\mathcal{A}$ for Π can be seen as a function $\mathcal{A}\colon \mathcal{I} \to \mathcal{S}$. Similarly, the optimal solutions for Π can be described by a function $\mathrm{OPT}\colon \mathcal{I} \to \mathcal{S}$. To define the stability of an algorithm, we need to quantify changes in the input instances and in the solutions. We can do so by imposing a metric on $\mathcal{I}$ and $\mathcal{S}$. Let $d_{\mathcal{I}}\colon \mathcal{I} \times \mathcal{I} \to \mathbb{R}_{\geq 0}$ be a metric for $\mathcal{I}$ and let $d_{\mathcal{S}}\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ be a metric for $\mathcal{S}$. We can then define the *stability* of an algorithm $\mathcal{A}\colon \mathcal{I} \to \mathcal{S}$ as follows.

$$\mathrm{St}(\mathcal{A}) = \max_{I,I' \in \mathcal{I}} \frac{d_{\mathcal{S}}(\mathcal{A}(I), \mathcal{A}(I'))}{d_{\mathcal{I}}(I, I')} \tag{1}$$

This definition for stability is closely related to that of the multiplicative distortion of metric embeddings, where $\mathcal{A}$ induces a metric embedding from the metric space $(\mathcal{I}, d_{\mathcal{I}})$ into $(\mathcal{S}, d_{\mathcal{S}})$. The lower the value for $\mathrm{St}(\mathcal{A})$, the more stable we consider the algorithm $\mathcal{A}$ to be. There are many other ways to define the stability of an algorithm given the metrics, but the above definition suffices for our purpose.

For many optimization problems, the function OPT may be very unstable. This suggests an interesting tradeoff between the stability of an algorithm and the solution quality. Unfortunately, the generic formulation of stability provided above is very unwieldy. It is not always clear how to define metrics $d_{\mathcal{I}}$ and $d_{\mathcal{S}}$ such that meaningful results can be derived. Additionally, it is not obvious how to deal with optimization problems with continuous input and discrete solutions, where the algorithm is inherently discontinuous, and thus the stability is unbounded by definition. Finally, analyses of this form are often very complex, and it is not straightforward to formulate a simplified version of the problem. In our framework we hence distinguish three types of stability analysis: event stability, topological stability, and Lipschitz stability.

Event stability follows the setting of kinetic data structures (KDS). That is, the input (a set of moving objects) changes continuously as a function over time. However, contrary to typical KDSs where a constraint is imposed on the solution quality, we aim to enforce the stability of the algorithm. For event stability we simply disallow the algorithm to change the solution too rapidly. Doing so directly is problematic, but we formalize this approach using the concept of k-optimal solutions. As a result, we can obtain a tradeoff between stability and quality that can be tuned by the parameter k. Note that event stability captures only *how often* the solution changes, but not *how much* the solution changes at each event.

Topological stability takes a first step towards the generic setup described above. However, instead of measuring the amount of change in the solution using a metric, we merely require the solution to behave continuously. To do so we only need to define a topology on the solution

space $\mathcal{S}$ that captures stable behavior. Surprisingly, even though we ignore the amount of change in a single time step, this type of analysis still provides meaningful information on the tradeoff between solution quality and stability. In fact, the resulting tradeoff can be seen as a lower bound for any analysis involving metrics that follow the used topology.

Lipschitz stability finally captures the generic setup described above. As the name suggests, we require the algorithm to be Lipschitz continuous and we provide an upper bound on the Lipschitz constant, which is equivalent to $\mathrm{St}(\mathcal{A})$. We are again interested in the quality of the solutions that can be obtained with any Lipschitz stable algorithm. Given the complexity of this type of analysis, a complete tradeoff for any value of the Lipschitz constant is typically out of reach, but results for sufficiently small or large values can be of interest.

Remark. Our framework makes the assumption that an algorithm is a function $\mathcal{A}\colon \mathcal{I} \to \mathcal{S}$. However, in a kinetic setting this is not necessarily true, since the algorithm has *history*. More precisely, for some input instance I, a kinetic algorithm may produce different solutions for I based on the instances processed earlier. We generally allow this behavior, and for event stability this behavior is even crucial. However, for the sake of simplicity, we will treat an algorithm as a function. We also generally assume in our analysis that the input is time-varying, that is, the input is a function over time, or follows a trajectory through the input space $\mathcal{I}$. Again, for the sake of simplicity, this is not always directly reflected in our definitions. Beyond that, we operate in the black-box model, in the sense that the algorithm does not know anything about future instances.

In the remainder we focus on topological stability, all omitted material (the description of event and Lipschitz stability, as well as proofs) can be found in the full version [9].

3 Topological stability

Topological stability analysis is applicable to a wide variety of problems and enforces continuous changes to the solution. Even though it does not capture stability in its entirety, as changes can happen in infinitesimally short time, topological stability still illustrates clearly how solutions and their quality have to change as the input changes.

3.1 Topological stability analysis

Let Π be an optimization problem with input instances $\mathcal{I}$, solutions $\mathcal{S}$, and optimization function f. An algorithm $\mathcal{A}\colon \mathcal{I} \to \mathcal{S}$ is *topologically stable* if, for any (continuous) path $\pi\colon [0,1] \to \mathcal{I}$ in $\mathcal{I}$, $\mathcal{A}\pi$ is a (continuous) path in $\mathcal{S}$. To properly define a (continuous) path in $\mathcal{I}$ and $\mathcal{S}$ we need to specify a topology $\mathcal{T}_\mathcal{I}$ on $\mathcal{I}$ and a topology $\mathcal{T}_\mathcal{S}$ on $\mathcal{S}$. Alternatively we could specify metrics $d_\mathcal{I}$ and $d_\mathcal{S}$, but this is typically more involved. We then want to analyze the approximation ratio of any topologically stable algorithm with respect to OPT. That is, we are interested in the ratio

$$\rho_{\mathrm{TS}}(\Pi, \mathcal{T}_\mathcal{I}, \mathcal{T}_\mathcal{S}) = \inf_{\mathcal{A}} \sup_{I \in \mathcal{I}} \frac{f(I, \mathcal{A}(I))}{f(I, \mathrm{OPT}(I))} \tag{2}$$

where the infimum is taken over all topologically stable algorithms. Naturally, if OPT is already topologically stable, then this type of analysis does not provide any insight and the ratio is simply 1. However, in many cases, OPT is not topologically stable. The above analysis can also be applied if the solution space (or the input space) is discrete. In such cases, continuity can often be defined using the graph topology of so-called flip graphs, for example,

based on edge flips for triangulations or rotations in rooted binary trees. We can represent a graph as a topological space by representing vertices by points, and representing every edge of the graph by a copy of the unit interval $[0, 1]$. These intervals are glued together at the vertices. In other words, we consider the corresponding simplicial 1-complex. Although the points in the interior of the edges of this topological space do not necessarily represent proper solutions, we can still use this topological space in Equation 2 by extending f over the edges via linear interpolation. It is not hard to see that we need to consider only the vertices of the flip graph (which do represent proper solutions) to compute the topological stability ratio.

Lower bounds. When proving lower bounds on the topological stability of a problem we want to force any algorithm that continuously updates the solution to produce a particularly bad intermediate result. We first show that updating a certain configuration continuously will always result in an intermediate solution of low quality, no matter which algorithm is used. We then also provide a particular motion that forces an update to the solution in that configuration. Updating the solution at any other point during the motion should lead to an even worse result. The motion and the described configuration together allow us to prove a lower bound on the topological stability ratio. In this abstract we do not describe the motions for the lower bound proofs; the complete proofs can be found in the full version [9].

3.2 Topological stability of EMSTs

Our input consists of a set of n points where each point has a trajectory. We require that the trajectories are continuous. The goal is to maintain a combinatorial description of a short spanning tree on these points, whose length stays close to optimal. To define this properly, we need to define a topology on the input space, but for a kinetic point set with n points in d dimensions we can simply use the standard topology on $\mathbb{R}^{dn}$ as $\mathcal{T}_\mathcal{I}$. To apply topological stability analysis, we also need to specify a topology on the (discrete) solution space. As the points move, the minimum spanning tree may have to change at some point in time by removing one edge and inserting another edge. Since these two edges may be very far apart, we do not consider this operation to be stable or continuous. Instead we specify the topology of $\mathcal{S}$ using a flip graph, where the operations are either *edge slides* or *edge rotations* [1, 7]. The optimization function f, measuring the quality of the EMST, is naturally defined for the vertices of the flip graph as the length of the spanning tree, and we use linear interpolation to define f on the edges of the flip graph. For edge slides and rotations we provide upper and lower bounds on $\rho_{\mathrm{TS}}(\mathrm{EMST}, \mathcal{T}_\mathcal{I}, \mathcal{T}_\mathcal{S})$.

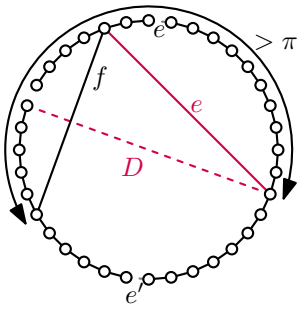
Edge slides. An edge slide is defined as the operation of moving one endpoint of an edge to one of its neighboring vertices along the edge to that neighbor. More formally, an edge (u, v) can be replaced by (u, w) if w is a neighbor of v and $w \neq u$. Since this operation is very local, we consider it to be stable. Note that after every edge slide the tree must still be connected.

▶ **Lemma 1.** *If $\mathcal{T}_\mathcal{S}$ is defined by edge slides, then $\rho_{\mathrm{TS}}(EMST, \mathcal{T}_\mathcal{I}, \mathcal{T}_\mathcal{S}) \leq \frac{3}{2}$.*

Proof. Consider a time where the EMST has to be updated by removing an edge e and inserting an edge e', where $|e| = |e'|$. Note that e and e' form a cycle C with other edges of the EMST. We now slide edge e to edge e' by sliding it along the vertices of C. Let x be the longest intermediate edge when sliding from e to e' (see Fig. 1(a)). To allow x to be as long as possible with respect to the length of the EMST and as such achieving an upper bound on ρ_{TS}, the EMST should be fully contained in C. By the triangle inequality we get that $2|x| \leq |C|$. Since the length of the EMST is $\mathrm{OPT} = |C| - |e|$, we get that $|x| \leq \mathrm{OPT}/2 + |e|/2$. Thus, the length of the intermediate tree is $|C| - 2|e| + |x| = \mathrm{OPT} - |e| + |x| \leq \frac{3}{2}\,\mathrm{OPT}$. ◀

▶ **Lemma 2.** *If $\mathcal{T}_S$ is defined by edge slides, then $\rho_{\text{TS}}(EMST, \mathcal{T}_I, \mathcal{T}_S) \geq \frac{\pi+1}{\pi}$.*

Proof. Consider a point in time where the EMST has to be updated by removing an edge e and inserting an edge e', where $|e|$ is very small. Let the remaining points be arranged in a circle with diameter D, as shown in the figure on the right. We get that $OPT < \pi D$, where OPT is the length of the EMST. Simply sliding e to e' will always grow e to be nearly the diameter of the circle at some point, as shown by the red dashed line. More precisely, e will grow to length at least $D - \varepsilon$, and we can make ε arbitrarily small by using a sufficient number of points. Alternatively, e (in the red configuration) can take a shortcut by sliding over another edge f. This is only beneficial if $|e| + |f| < D - \varepsilon$. However, if f helps e to avoid becoming a diameter of the circle, then e and f, as chords, must span an angle larger than π together. Hence, by triangle inequality, $|e| + |f| \geq D$. Thus, for any $\varepsilon > 0$, $\rho_{\text{TS}}(\text{EMST}, \mathcal{T}_I, \mathcal{T}_S) \geq \frac{OPT+D}{OPT} - \varepsilon > \frac{OPT+OPT/\pi}{OPT} - \varepsilon = \frac{\pi+1}{\pi} - \varepsilon \approx 1.318 - \varepsilon$. ◀



Edge rotations. Edge rotations are a generalization of edge slides, that allow one endpoint of an edge to move to any other vertex. These operations are clearly not as stable as edge slides, but they are still more stable than the deletion and insertion of arbitrary edges.

▶ **Lemma 3.** *If $\mathcal{T}_S$ is defined by edge rotations, then $\rho_{\text{TS}}(EMST, \mathcal{T}_I, \mathcal{T}_S) \leq \frac{4}{3}$.*

Proof. Consider a time where the EMST has to be updated by removing an edge $e = (u, v)$ and inserting an edge $e' = (u', v')$, where $|e| = |e'|$. Note that e and e' form a cycle C with other edges of the EMST. We now rotate edge e to edge e' along some of the vertices of C. Let x be the longest intermediate edge when rotating from e to e'. To allow x to be as long as possible with respect to the length of the EMST, the EMST should be fully contained in C. We argue that $|x| \leq \text{OPT}/3 + |e|$, where OPT is the length of the EMST. Removing e and e' from C splits C into two parts, where we assume that u and u' (v and v') are in the left (right) part. First assume that one of the two parts has length at most OPT /3. Then we can rotate e to (u, v'), and then to e', which implies that $|x| = |(u, v')| \leq \text{OPT}/3 + |e|$ by the triangle inequality (see Fig. 1(b)). Now assume that both parts have length at least OPT /3. Let $e_L = (a, b)$ be the edge in the left part that contains the midpoint of that part, and let $e_R = (c, d)$ be the edge in the right part that contains the midpoint of that part, where u_L and u_R are closest to e (see Fig. 1(c)). Furthermore, let Z be the length of $C \setminus \{e, e', e_L, e_R\}$.

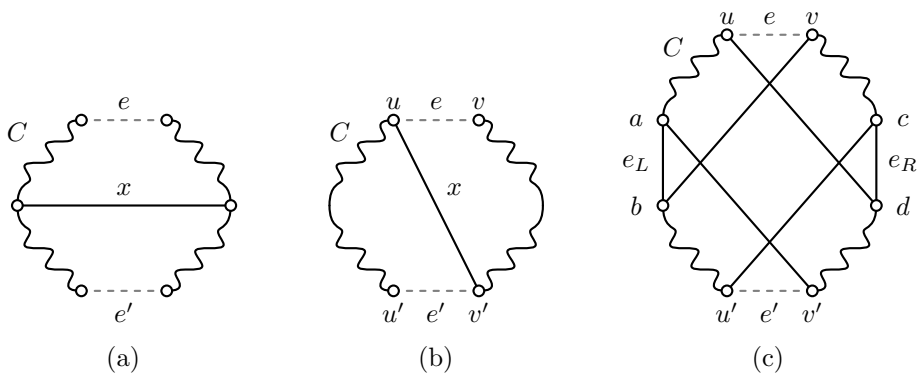
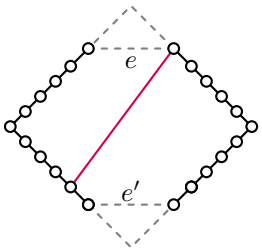


Figure 1 (a): Illustration for Lemma 1. (b) and (c): Illustrating the two cases for Lemma 3.

Now consider the potential edges (u,d), (v,b), (u',c), and (v',a). By the triangle inequality, the sum of the lengths of these edges is at most $4|e|+2|e_L|+2|e_R|+Z$. Thus, one of these potential edges has length at most $|e|+|e_L|/2+|e_R|/2+Z/4$. Without loss of generality let (u,d) be that edge (the construction is fully symmetric). We can now rotate e to (u,d), then to (u',d), and finally to e'. As each part of C has length at most $2\,\mathrm{OPT}/3$, we get that $|(u',d)| \leq \mathrm{OPT}/3+|e|$ by construction. Furthermore we have that $\mathrm{OPT} = |e|+|e_L|+|e_R|+Z$. Thus, $|(u,d)| \leq |e|+|e_L|/2+|e_R|/2+Z/4 = \mathrm{OPT}/3+2|e|/3+|e_L|/6+|e_R|/6-Z/12$. Since e needs to be removed to update the EMST, it must be the longest edge in C. Therefore $|(u,d)| \leq \mathrm{OPT}/3+|e|$, which shows that $|x| \leq \mathrm{OPT}/3+|e|$. Since the length of the intermediate tree is $\mathrm{OPT}-|e|+|x| \leq \frac{4}{3}\,\mathrm{OPT}$, we obtain that $\rho_{\mathrm{TS}}(\mathrm{EMST},\mathcal{T}_\mathcal{I},\mathcal{T}_\mathcal{S}) \leq \frac{4}{3}$. ◀

▶ **Lemma 4.** *If $\mathcal{T}_\mathcal{S}$ is defined by edge rotations, then, $\rho_{\mathrm{TS}}(EMST,\mathcal{T}_\mathcal{I},\mathcal{T}_\mathcal{S}) \geq \frac{10-2\sqrt{2}}{9-2\sqrt{2}}$.*

Proof. Consider a point in time where the EMST has to be updated by removing an edge e and inserting an edge e'. Let the remaining points be arranged in a diamond shape as shown in the figure on the right, where the side length of the diamond is 2, and $|e| = |e'| = 1$. Now we define a *top-connector* as an edge that intersects the vertical diagonal of the diamond, but is completely above the horizontal diagonal of the diamond. A *bottom-connector* is defined analogously, but must be completely below the horizontal diagonal. Finally, a *cross-connector* is an edge that hits both diagonals of the diamond. Note that a cross-connector has length at least 2, and a top- or bottom-connector has length at least $|e| = 1$. In the considered update, we start with a top-connector and end with a bottom-connector. Since we cannot rotate from a top-connector to a bottom-connector in one step, we must reach a state that either has both a top-connector and a bottom-connector, or a single cross-connector. In both options the length of the spanning tree is $10-2\sqrt{2}$, while the minimum spanning tree has length $9-2\sqrt{2}$. Thus $\rho_{\mathrm{TS}}(\mathrm{EMST},\mathcal{T}_\mathcal{I},\mathcal{T}_\mathcal{S}) \geq \frac{10-2\sqrt{2}}{9-2\sqrt{2}} \approx 1.162$. ◀



References

1 O. Aichholzer, F. Aurenhammer, and F. Hurtado. Sequences of spanning trees and a fixed tree theorem. *Comp. Geom. Theory Appl.*, 21(1-2):3–20, 2002.

2 K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comp. Geom. Theory Appl.*, 43(3):312–328, 2010.

3 P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. P. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. Mitchell. On simultaneous planar graph embeddings. *Comp. Geom. Theory Appl.*, 36(2):117–130, 2007.

4 M. de Berg, M. Roeloffzen, and B. Speckmann. Kinetic 2-centers in the black-box model. In *Proc. 29th Symp. Comp. Geom.*, pages 145–154, 2013.

5 S. Durocher and D. Kirkpatrick. The Steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. *I. J. Comp. Geom. Appl.*, 16(04):345–371, 2006.

6 S. Durocher and D. Kirkpatrick. Bounded-velocity approximation of mobile Euclidean 2-centres. *I. J. Comp. Geom. Appl.*, 18(03):161–183, 2008.

7 W. Goddard and H. C. Swart. Distances between graphs under edge operations. *Discrete Mathematics*, 161(1-3):121–132, 1996.

8 L. J. Guibas. Kinetic data structures. In D. P. Mehta and S. Sahni (eds), *Handbook of Data Structures and Applications*, pages 23.1–18. Chapman and Hall/CRC, 2004.

9 W. Meulemans, B. Speckmann, K. Verbeek, and J. Wulms. A framework for algorithm stability. *CoRR*, abs/1704.08000, 2017. URL: `http://arxiv.org/abs/1704.08000`.

Guarding Monotone Polygons with Vertex Half-Guards is NP-Hard*

Matt Gibson[1], Erik Krohn[2], and Matthew Rayford[2]

1 **University of Texas - San Antonio, gibson@cs.utsa.edu**
2 **University of Wisconsin - Oshkosh, {krohne, rayfom16}@uwosh.edu**

Abstract

We consider a variant of the art gallery problem where all guards are limited to seeing to the right inside a monotone polygon. We show that the problem is NP-hard if guards are restricted to be at the vertices of the polygon.

1 Introduction

An instance of the art gallery problem takes as input a simple polygon P. If these edges do not intersect other than at the vertices in V, then P is called a simple polygon. The edges of a simple polygon give us two disjoint regions: the interior and exterior of the polygon. For any two points $p, q \in P$, we say that p *sees* q if the line segment $\overline{pq}$ does intersect the exterior of P. The art gallery problem seeks to find a set of points $G \subseteq P$ such that every point $p \in P$ is seen by a point in G. We call this set G a guarding set. In the point guarding problem, guards can be placed anywhere in the interior of P. In the vertex guarding problem, guards are only allowed to be placed at V. The optimization problem is thus defined as finding the smallest such G.

Art gallery problems are motivated by applications such as line of-sight transmission networks in terrains, signal communications and cellular telephony systems and other telecommunication technologies as well as placement of motion detectors and security cameras.

1.1 Previous Work

The question of whether guarding simple polygons is NP-hard was independently confirmed by Aggarwal [2] and Lee and Lin [15]. They showed that the problem is NP-hard for both vertex guarding and point guarding. Along with being NP-complete, Brodén et al. [6] and Eidenbenz [8] independently proved that point guarding simple polygons is APX-hard. This means that there exists a constant $\epsilon > 0$ such that no polynomial-time algorithm can guarantee an approximation ratio of $(1+\epsilon)$ unless P=NP. Ghosh provides a $O(\log n)$-approximation for the problem of vertex guarding an n-vertex simple polygon in [10]. This result can be improved for simple polygons using randomization, giving an algorithm with expected running time $O(nOPT^2 \log_4 n)$ that produces a vertex guard cover with approximation factor $O(\log OPT)$ with high probability, where OPT is the smallest vertex guard cover for the polygon [7]. Bhattacharya et. al claim a constant factor approximation for guarding simple polygons using vertex guards in [4]. Assuming integer coordinates and a specific general position, Bonnet and Miltzow present an algorithm for finding a point guard cover with approximation factor $O(\log OPT)$ in [5]. King and Kirkpatrick provide a $O(\log \log OPT)$-approximation algorithm for the problem of guarding a simple polygon with guards on the perimeter in [12].

* Partially supported by a faculty development grant from UW-Oshkosh

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Additional Polygon Structure. Due to the inherent difficulty in fully understanding the art gallery problem for simple polygons, there has been some work done guarding polygons with some additional structure. A simple polygon P is x-monotone (or simply monotone) if any vertical line intersects the boundary of P in at most two points. Let l and r denote the leftmost and rightmost point of P respectively. Consider the "top half" of the boundary of P by walking along the boundary clockwise from l to r. We call this the *ceiling* of P. Similarly we obtain the *floor* of P by walking clockwise along the boundary from r to l. Notice that both the ceiling and the floor are x-monotone polygonal chains, that is a vertical line intersects it in at most one point. Krohn and Nilsson [14] give a polynomial-time constant factor approximation algorithm for point guarding monotone polygons. They also proved point guarding and vertex guarding a monotone polygon is NP-hard [13, 14].

α-Floodlights. Motivated by the fact that many cameras and other sensors generally are not able to sense in 360 degrees, previous works have considered the problem when guards have a fixed sensing angle α for some $0 < \alpha \leq 360$. This problem is often referred to as the *α-floodlight problem*. 180°-floodlights are sometimes referred to as *half-guards*. Some of the work on this problem has involved proving necessary and sufficient bounds on the number of α-floodlights required to guard (or illuminate) an n vertex simple polygon P, where floodlights are anchored at vertices in P and no vertex is assigned more than one floodlight, see for example [17, 9, 16]. From an approximation complexity standpoint, it is known that computing a minimum cardinality set of α-floodlights to illuminate a simple polygon P is APX-hard for both the point guard and vertex guard variants [1, 3]. Other works in this area include considering the problem where $\alpha < 180°$.

1.2 Our Contribution

In this paper, we consider guarding monotone polygons with half-guards that can see in one direction, namely to the right. Let $p.x$ denote the x-coordinate of a point p. We modify the definition of *sees* to be the following: a point p sees a point q if the line segment $\overline{pq}$ does not intersect the exterior of P and $p.x \leq q.x$. A constant factor approximation for this problem was given in [11].

Our main result is to show that vertex guarding a monotone polygon with half-guards is NP-hard. Krohn and Nilsson [14] obtained a similar NP-hardness result using full guards, but guards were required to see in all directions. The reduction could not be trivially tweaked to show the half-guard problem is NP-hard.

In Section 2, we provide a high level overview that vertex guarding a monotone polygon with half-guards is NP-hard. Section 3 provides the details of the proof.

2 NP-Hardness for Vertex Guards

The reduction is from *3SAT*. A 3SAT instance (X, C) contains a set of Boolean variables, $X = \{x_1, x_2, \ldots, x_n\}$ and a set of clauses, $C = \{c_1, c_2, \ldots, c_m\}$. Each clause contains three literals, $c_i = (x_j \vee x_k \vee x_l)$. A 3SAT instance is satisfiable if a satisfying truth assignment for X exists such that all clauses c_i are true. We show that any 3SAT instance is polynomially transformable to an instance of vertex guarding a monotone polygon using half-guards. We construct a monotone polygon P from the 3SAT instance such that P is guardable by $K = (2 + m)n + 1$ or fewer guards if and only if the 3SAT instance is satisfiable.

The high level overview of the reduction is that certain vertices represent the truth values of the variables in the 3SAT instance. All starting patterns are placed on the ceiling on the left side of the polygon, see Figure 1. We assume that all guards can see only to the

right. In these starting patterns, one must choose one of two guardset locations in order to guard distinguished vertices for that particular pattern. A *distinguished vertex* is a vertex that is seen only by a small number of specific vertices. In each variable pattern, similar to a starting pattern, certain vertices will represent a truth assignment of true and certain vertices will represent a truth assignment of false for some variable. This information is then "mirrored rightward" going from the ceiling, to the floor and then back to the ceiling such that there is a consistent choice of the x_j vertices or the $\overline{x_j}$ vertices for each variable. This differs from previous results where variable information was mirrored from the "left side" of the polygon to the "right side" of the polygon and then back to the left side. A distinguished clause vertex is placed to the right of the variable patterns such that only the vertices representing the literals in the specific clause can see the clause distinguished vertex. A high level example of the entire reduction is shown in Figure 1.

▶ **Theorem 1.** *Finding the smallest vertex guard cover for a monotone polygon using half guards is NP-hard.*

3 Hardness Details

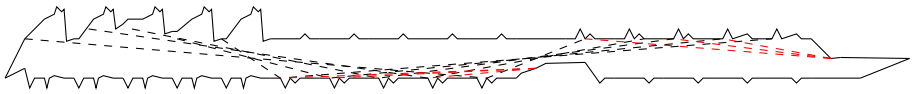


Figure 1 A high level overview of the reduction.

Starting Pattern: This pattern appears along the left side of the monotone polygon a total of n times, one corresponding to each variable, see Figure 2. In each pattern, there are 3 distinguished vertices: $\{h_4, h_{10}, h_{13}\}$. These vertices are seen by a specific subset of vertices in each starting pattern. It is important to note that no other vertex outside of this starting pattern sees these distinguished points. Let $v_l(p)$ be the set of vertices that see p. Note that all vertices in $v_l(p)$ lie to the left of p or on the vertical line that contains p.

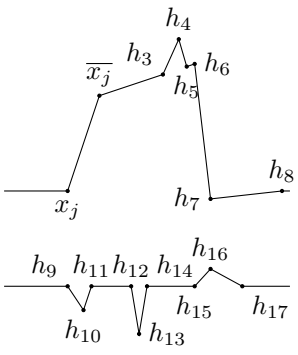


Figure 2 A starting pattern.

Let's assume we are considering the starting pattern for variable x_j. $v_l(h_{10}) = \{h_{10}, h_9, x_j\}$, $v_l(h_{13}) = \{h_{12}, h_{13}, \overline{x_j}\}, v_l(h_4) = \{h_3, h_4, h_9, h_{11}, h_{12}, h_{14}\}$. One should note that one guard does not see all of the distinguished points. Two guards are necessary and sufficient. The only possible combinations of vertex guards that see each distinguished vertex are: $\{x_j, h_{12}\}, \{\overline{x_j}, h_9\}$. If the second option is chosen, then it appears that the x_j vertex is unseen. However, the polygon is drawn in such a way such that the leftmost point in the polygon sees x_j for all j, see Figure 1.

Variable Pattern: On the floor of the polygon to the right of the n starting patterns are the first n variable patterns, one for each variable, that verify and propagate the assigned truth value of each variable. The variables are in reverse order from the initial starting pattern. The variables are ordered from $x_1, x_2, \ldots, x_n$ in the starting patterns from left to right. However, the variables are ordered from $x_n, x_{n-1}, \ldots, x_1$ in the first grouping of variable patterns from left to right. When the variables are "mirrored" rightward again to the ceiling, the ordering will again reverse. See Figure 1 for a high level overview.

A single variable pattern is shown in Figure 4. Similar to the starting pattern, there are 3 distinguished vertices located at $\{v_2, v_5, v_7\}$. The visibility for these points within this pattern are as follows: $v_l(v_2) = \{v_1, v_2, x_j, v_6, \overline{x_j}, v_8\}, v_l(v_5) = \{v_5, x_j\}, v_l(v_7) = \{v_7, \overline{x_j}\}$. It should be noted that v_2 is not seen by another vertex outside of this pattern. One guard within this pattern is necessary to guard this distinguished vertex. Along with these visibilities, v_5 is seen by the $\overline{x_j}$ vertex in the starting pattern representing x_j. v_5 does not see the x_j vertex from the starting pattern because it is angled in such a way that its line of sight is "above" the x_j vertex in the starting pattern. v_7 is seen by the x_j vertex in the starting pattern representing x_j. The reason it is not seen by $\overline{x_j}$ is because the $\overline{x_j}$ vertex in the starting pattern is being blocked by h_7 in the starting pattern. Figure 3 shows how the starting patterns are connected to variable patterns.

Variable patterns are connected to other variable patterns in a similar fashion. Consider a set of n variable patterns on the floor representing one mirroring of the variables. At the far right of these patterns is a vertex called c_2. This vertex will block our x_j and $\overline{x_j}$ vertices from seeing too far to the right. Consider a single variable x_j being mirrored from the floor to the ceiling. In Figure 5, the ceiling variable pattern is simply an inverted floor variable pattern. The v_5 vertex in the ceiling variable pattern sees the $\overline{x_j}$ vertex in the floor variable pattern and not the x_j vertex in the floor variable pattern because the angle of the polygon blocks it. v_7 in the ceiling variable pattern sees the x_j vertex in the floor variable pattern but not the $\overline{x_j}$ vertex in the floor variable pattern because it is being blocked by c_2.

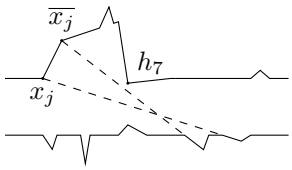


Figure 3 Starting pattern interacting with first variable gadget.

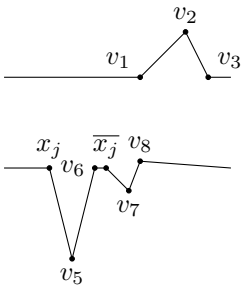


Figure 4 Variable pattern x_j.

Different variable patterns that represent different variables will not affect each other. For example, take the starting pattern for an arbitrary x_i and call the vertices that see the distinguished vertices in that starting pattern the X_i set. Now consider the variable pattern for x_i and look at the variable patterns to the left of x_i on the floor. None of X_i can see the distinguished vertices of variable patterns to the left of the variable pattern for x_i because the distinguished vertices in those variable patterns are angled too far to the "right." None of X_i can see distinguished vertices of variable patterns to the right of the variable pattern for x_i because h_7 or h_{16} is blocking them from seeing too far right, see Figure 2.

In a similar fashion, variable patterns will not affect other variable patterns when mirroring, see Figure 6. The variable pattern on the ceiling for x_i will not be seen by the previous variable pattern on the floor for x_{i+1} because the angle of the polygon in the variable pattern for x_i on the ceiling is too steep. In other words, the distinguished vertices for x_i on the ceiling will not be able to be seen from that far left. The vertices in the variable pattern on the ceiling for x_i will not be seen by the previous variable pattern on the ceiling for x_{i-1} because the c_2 vertex will block them.

We allow one guard to be placed in a single variable pattern. No single guard is able to see all of the distinguished points. Therefore, one must rely on previously placed guards to help see at least 1 of the distinguished points in the variable pattern. If we choose x_j in the starting pattern or in some previous variable pattern, we see the v_7 vertex in the subsequent variable pattern. The only guard in the variable pattern that sees v_2 and v_5 is x_j. If we choose $\overline{x_j}$ in the starting pattern or in some previous variable pattern, the distinguished points that are unseen are v_2 and v_7 in the subsequent variable patterns and the only guard in the variable pattern that sees them is $\overline{x_j}$. In this second case, x_j is seen by that previously placed guard that also sees v_5.

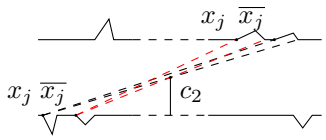


Figure 5 An example of a variable being mirrored.

Clauses: For each clause c in the boolean formula, there is a sequence of variable patterns $x_1, \ldots, x_n$ along either the ceiling or the floor of the polygon. Immediately to the right of the variable patterns exists a clause pattern. A clause pattern consists of one vertex such that the vertex is only seen by the variable patterns corresponding to the literals in the clause; see Figure 7. The distinguished vertex of the clause pattern is the c_3 vertex. This vertex is seen only by specific vertices in its respective sequence of variable patterns.

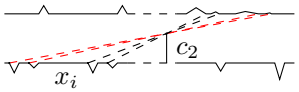


Figure 6 An example of multiple variables being mirrored.

To see how a clause is placed in the polygon, consider Figure 8 that represents the clause $x_1 \vee \overline{x_3} \vee x_5$. Initially, all x_i and $\overline{x_i}$ vertices in their respective variable patterns are blocked from seeing the c_3 clause point by their respective v_8 vertex. Consider the example clause of $x_1 \vee \overline{x_3} \vee x_5$. In the case of x_1 and x_5, their respective v_8 vertices have been lowered just enough such that the v_8 vertex is no longer blocking them from seeing c_3. However, v_8 is still blocking $\overline{x_j}$ from seeing c_3. In the case of $\overline{x_3}$, the v_8 guard is lowered enough such that $\overline{x_3}$ sees c_3. To keep x_3 from seeing c_3, we raise the v_6 vertex just enough so it blocks x_3 from c_3. It should be noted that these small tweaks do not affect the mirroring of variable truth values. None of the x_j or $\overline{x_j}$ vertices were moved. Their position with respect to the key blocker of c_2 is the same. Therefore, c_2 still blocks each respective vertex from seeing too far to the right.

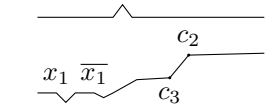


Figure 7 A clause gadget to the right of x_1.

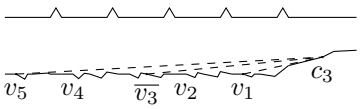


Figure 8 The clause $(v_1 \vee \overline{v_3} \vee v_5)$.

Putting it all together: We choose our truth value for each variable in the starting variable patterns. The truth values are then mirrored in turn between variable patterns on the ceiling and the floor. In the example of Figure 8 the 3SAT clause corresponds to $c = x_1 \vee \overline{x_3} \vee x_5$. Hence, a vertex guard placement that corresponds to a truth assignment that makes c_3 true, will have at least one guard on $x_1, \overline{x_3}$ or x_5 and can therefore see vertex c_3 without additional guards. We still have variables x_2 and x_4 on the polygon, however, none of them or their negations see the vertex c_3. They are simply there to transfer their truth values in case these variables are needed in later clauses.

The monotone polygon we construct consists of $17n + (9n + 3)m + 2$ vertices. Each starting variable pattern has 17 vertices, each variable pattern 9 vertices, the clause pattern has 3 vertices, plus 2 vertices for the leftmost and rightmost points of the polygon. Exactly $K = (2 + m)n + 1$ guards are required to guard the polygon. 2 guards are required to see the

distinguished points of the starting patterns ($2n$) and 1 guard is required at every variable pattern, of which there are (mn) of them. Lastly, since a starting pattern cannot begin at the leftmost point, a guard is required at the leftmost vertex of the polygon. If the 3SAT instance is satisfiable, then guards are placed at vertices in accordance to whether the variable is true or false in each of the sequences of variable patterns. Each clause vertex is seen since one of the literals in the associated clause is true and the corresponding vertex has a guard.

References

1. Ahmed Abdelkader, Ahmed Saeed, Khaled A. Harras, and Amr Mohamed. The inapproximability of illuminating polygons by α-floodlights. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*, 2015.
2. Alok Aggarwal. *The art gallery theorem: its variations, applications and algorithmic aspects.* PhD thesis, 1984. AAI8501615.
3. Jay Bagga, Laxmi Gewali, and David Glasser. The complexity of illuminating polygons by alpha-flood-lights. In *Proceedings of the 8th Canadian Conference on Computational Geometry, Carleton University, Ottawa, Canada, August 12-15, 1996*, pages 337–342, 1996.
4. Pritam Bhattacharya, Subir Kumar Ghosh, and Sudebkumar Pal. Constant approximation algorithms for guarding simple polygons using vertex guards. `https://arxiv.org/abs/1712.05492v1`.
5. Édouard Bonnet and Tillmann Miltzow. An approximation algorithm for the art gallery problem. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 20:1–20:15, 2017.
6. Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is apx-hard. In *CCCG*, pages 45–48, 2001.
7. Alon Efrat and Sariel Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.
8. Stephan Eidenbenz. Inapproximability results for guarding polygons without holes. In Kyung-Yong Chwa and Oscar H. Ibarra, editors, *ISAAC*, volume 1533 of *Lecture Notes in Computer Science*, pages 427–436. Springer, 1998.
9. Vladimir Estivill-Castro, Joseph O'Rourke, Jorge Urrutia, and Dianna Xu. Illumination of polygons with vertex lights. *Information Processing Letters*, 56(1):9 – 13, 1995.
10. Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997.
11. Matt Gibson, Erik Krohn, and Matthew Rayford. Guarding monotone polygons with half-guards. In *Proceedings of the 29th Canadian Conference on Computational Geometry, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada*, pages 168–173, 2017.
12. James King and David G. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete & Computational Geometry*, 46(2):252–269, 2011.
13. Erik Krohn and Bengt J. Nilsson. The complexity of guarding monotone polygons. In *CCCG*, pages 167–172, 2012.
14. Erik A. Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, pages 1–31, 2012.
15. D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inform. Theory*, 32(2):276–282, March 1986.
16. Bettina Speckmann and Csaba D. Tóth. Allocating vertex π-guards in simple polygons via pseudo-triangulations. *Discrete & Computational Geometry*, 33(2):345–364, 2005.
17. Csaba D. Tóth. Art gallery problem with guards whose range of vision is 180°. *Computational Geometry*, 17(3):121 – 134, 2000.

Progressive Simplification of Polygonal Curves

Kevin Buchin[1], Maximilian Konzack[2], and Wim Reddingius[3]

1 **TU Eindhoven, The Netherlands**
`k.a.buchin@tue.nl`
2 **TU Eindhoven, The Netherlands**
`m.p.konzack@tue.nl`
3 **TU Eindhoven, The Netherlands**
`wimreddingius@gmail.com`

Abstract

Simplifying polygonal curves at different levels of detail is an important problem with many applications. Existing geometric optimization algorithms are only capable of minimizing the complexity of a simplified curve for a single level of detail. We present an $O(n^3m)$-time algorithm that takes a polygonal curve of n vertices and produces a set of consistent simplifications for m scales while minimizing the cumulative simplification complexity. This algorithm is compatible with distance measures such as Hausdorff, Fréchet and area-based distances, and enables simplification for continuous scaling in $O(n^5)$ time.

1 Introduction

Given a polygonal curve as input, the curve simplification problem asks for a polygonal curve that approximates the input well using as few vertices as possible. Because of the importance of data reduction, curve simplification has a wide range of applications. One such application is cartography, where the visual representation of line features like rivers, roads, and region boundaries needs to be reduced. Most maps nowadays are interactive and incorporate zooming, which requires curve simplification that facilitates different levels of detail. A naive approach would be to simplify for each zoom level independently. This however has the drawback that the resulting simplifications are not consistent between different scales. Therefore, we require *progressive simplification*, that is, a series of simplifications for which the level of detail is progressively increased for higher zoom-levels. This is shown in Figure 1a.

Progressive simplifications are used in cartography [7]. Existing algorithms for progressive simplification (e.g. Cao et al. [2]) work by simplifying the input curve, then simplifying this simplification, and so on. Cao et al. [2] referred to progressive curve simplification as "aging". More concretely, a common approach is to iteratively discard vertices, such that we always discard the vertex whose removal introduces the smallest error (according to some criterion). For example, the algorithm by Visvalingam and Whyatt [9] always removes the vertex which together with its neighboring vertices forms a triangle with the smallest area.

Such approaches stand in stark contrast to (non-progressive) curve simplification algorithms that aim to minimize the complexity of the simplification while guaranteeing a (global) bound on the error introduced by the simplification. The most prominent algorithm with a preset error bound was proposed by Douglas and Peucker [5]. However, while heuristically aiming at a simplification with few vertices, this algorithm does not actually minimize the number of vertices. A general algorithm for the problem of minimizing the number of vertices was introduced by Imai and Iri [6]. Their approach uses *shortcut graphs*, which we describe in more detail below. An efficient algorithm to compute shortcut graphs for the Hausdorff distance was presented by Chan and Chin [3]. Inspired by the work of Visvalingam and Whyatt, Daneshpajouh et al. [4] defined an error measure for non-progressive simplification by

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

measuring the sum or the difference in area between a simplification and the input curve. In the line of these algorithms, the goal of our work is to develop algorithms that solve progressive simplification as an optimization problem.

A (vertex-restricted) *simplification* $\mathcal{S}$ of a polygonal curve $\mathcal{C}$ is an ordered subsequence of $\mathcal{C}$ (denoted by $\mathcal{S} \sqsubseteq \mathcal{C}$) that includes the first and the last point of $\mathcal{C}$. An *ε-simplification* $\mathcal{S}$ is a simplification that ensures that each edge of $\mathcal{S}$ has a distance of at most ε to its corresponding subcurve, where the distance measure can for instance be the Hausdorff or the Fréchet distance [1]. For an ordered pair of vertices (p_i, p_j) of $\mathcal{C}$ we denote the distance between the segment (p_i, p_j) and the corresponding subchain by $\varepsilon(p_i, p_j)$. We denote by $(p_i, p_j) \in \mathcal{S}$ that (p_i, p_j) is an edge of $\mathcal{S}$.

We next define the *progressive simplification problem* in the plane. Given a polygonal curve $\mathcal{C} = \langle p_1, \ldots, p_n \rangle$ in $\mathbb{R}^2$ and a sequence $\mathcal{E} = \langle \varepsilon_1, \ldots, \varepsilon_m \rangle$ with $\varepsilon_i \in \mathbb{R}_{>0}$ where $0 < \varepsilon_1 < \ldots < \varepsilon_m$, we want to compute a sequence of (vertex-restricted) simplifications $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ of $\mathcal{C}$ such that

1. $\mathcal{S}_m \sqsubseteq \mathcal{S}_{m-1} \sqsubseteq \ldots \sqsubseteq \mathcal{S}_1 \sqsubseteq \mathcal{C}$ (*monotonicity*),
2. $\mathcal{S}_k$ is an ε_k-simplification of $\mathcal{C}$,
3. $\sum_{k=1}^{m} |\mathcal{S}_k|$ is minimal.

We refer to a sequence of simplifications fulfilling the first two conditions as *progressive simplification*. A sequence fulfilling all three conditions is called a *minimal progressive simplification*, and the problem of computing such a sequence is called the *progressive simplification problem*. We present an $O(n^3 m)$-time algorithm for the progressive simplification problem in the plane.

The cornerstone of progressive simplification is that we require monotonicity. This guarantees that, when "zooming out", vertices are only removed and cannot (re)appear. As error measure, we will mostly use the Hausdorff distance. This is not essential to the core algorithm, and we will discuss how to use the Fréchet distance [1] or area-based measures [4] without affecting the worst-case running time. Furthermore, our algorithm generalizes to the *weighted* version of the problem in which $\sum_{i=1}^{m} w_i |\mathcal{S}_i|$ with positive weights w_i is minimized, and to the *continuous* version, where $\mathcal{S}_\varepsilon$ needs to be computed for all $0 \leq \varepsilon < \varepsilon_M$. As in the discrete setting, we require $\mathcal{S}_{\varepsilon'} \sqsubseteq \mathcal{S}_\varepsilon$ for $\varepsilon' > \varepsilon$; the resulting algorithm minimizes $\int_0^{\varepsilon_M} |\mathcal{S}_\varepsilon| \, d\varepsilon$ in $O(n^5)$ time. Note that ε_M is the error at which we can simplify the curve by the single line segment (p_1, p_n); thus, we have $\varepsilon_M = \varepsilon(p_1, p_n)$.

In our algorithms we will make use of the *shortcut graph* as introduced by Imai and Iri [6]. For a given curve $\mathcal{C}$, a *shortcut* (p_i, p_j) is an ordered pair $(i < j)$ of vertices. Given an error $\varepsilon > 0$, a shortcut (p_i, p_j) is *valid* if $\varepsilon(p_i, p_j) \leq \varepsilon$. The *shortcut graph* $G(\mathcal{C}, \varepsilon)$ [6] as shown in Figure 1b represents all valid shortcuts (p_i, p_j) with $1 \leq i < j \leq n$. A bottleneck in computing (progressive) simplifications is the construction and space usage of these graphs.

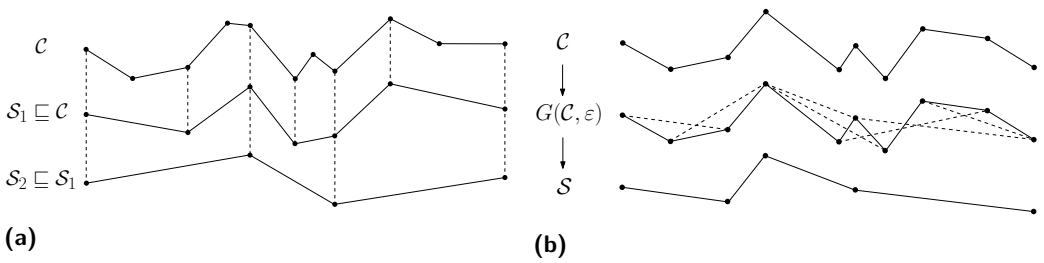


Figure 1 (a) a progressive simplification and (b) curve simplification using the shortcut graph.

2 Optimal Progressive Simplification

We show how to solve the progressive simplification problem in $O(n^3m)$ time in this section. The same running time holds for the weighted version, and based on this we show that the continuous progressive simplification problem can be solved in $O(n^5)$ time, see Section 3.

By the monotonicity property of the progressive simplification problem (see condition 1 in the definition in Section 1), we require that all vertices within a simplification $\mathcal{S}_k$ of the sequence must also occur within all subsequent simplifications $\mathcal{S}_l$ with $k < l$. Adding shortcuts to a specific simplification thus influences the structure of the other simplifications. We therefore associate a cost value $c^k_{i,j} \in \mathbb{N}$ for each shortcut (p_i, p_j) in the shortcut graph $G(\mathcal{C}, \varepsilon_k)$ that relates to the cost of including (p_i, p_j) in $\mathcal{S}_k$. We use the Hausdorff distance as an error measure to determine whether a shortcut is valid, but since the shortcut graph is flexible to use any error measures, we can employ any other distance measure for our algorithms. In particular for the Fréchet distance [1] and area-based distances [4], we can use brute-force to compute whether a shortcut is valid in $O(n)$ time, and therefore use these measures without changing the worst-case running time. We obtain a cost value $c^k_{i,j}$ for a shortcut $(p_i, p_j) \in G(\mathcal{C}, \varepsilon_k)$ by minimizing the costs of all possible shortcuts in $\langle p_i, \ldots, p_j \rangle$ at lower scales recursively. The dynamic program is defined as follows:

$$c^k_{i,j} = \begin{cases} 1 & \text{if } k = 1 \\ 1 + \min\limits_{\pi \in \prod^{k-1}_{i,j}} \sum\limits_{(p_x, p_y) \in \pi} c^{k-1}_{x,y} & \text{if } 1 < k \leq m \end{cases}$$

We use $\prod^k_{i,j}$ to denote the set of all paths in $G(\mathcal{C}, \varepsilon_k)$ from p_i to p_j.

The algorithm starts with constructing the shortcut graphs $G(\mathcal{C}, \varepsilon_1), \ldots, G(\mathcal{C}, \varepsilon_m)$. For most distance measures, the distance of shortcut (p_i, p_j) to the subcurve $\langle p_i, \ldots, p_j \rangle$ can be determined in $O(j - i)$ time. For such measures, constructing these graphs naively takes $O(n^3m)$ time. By employing the algorithm by Chan and Chin [3] we can compute it in $O(n^2m)$ time for the Hausdorff distance.

We compute all cost values from scale $k = 1$ up to m by assigning a weight $c^k_{i,j}$ to each shortcut $(p_i, p_j) \in G(\mathcal{C}, \varepsilon_k)$. For each shortcut $(p_i, p_j) \in G(\mathcal{C}, \varepsilon_k)$, we compute $c^k_{i,j}$ by finding a shortest path π in $G(\mathcal{C}, \varepsilon_{k-1})$ from p_i to p_j, minimizing $\sum_{(p_x, p_y) \in \pi} c^{k-1}_{x,y}$ thereby.

We can use any shortest path algorithm, such as Dijkstra's algorithm. On each scale k, we need to run Dijkstra's algorithm on $O(n)$ source nodes of $G(\mathcal{C}, \varepsilon_k)$. This yields a worst case running time of $O(n^3m)$, because Dijkstra's algorithm runs in $O(n^2)$ time on weighted shortcut graphs with integer weights.

We increment $c^k_{i,j} = c^{k-1}_{i,j} + 1$ for any shortcut $(p_i, p_j) \in G(\mathcal{C}, \varepsilon_{k-1})$. By doing so, we avoid recomputations of shortest paths and reuse cost values whenever necessary.

We construct the sequence of simplifications from $\mathcal{S}_m$ down to $\mathcal{S}_1$. First, we compute $\mathcal{S}_m$ by returning the shortest path from p_1 to p_n in $G(\mathcal{C}, \varepsilon_m)$ using the computed cost values at scale m. Next, we compute a shortest path P from p_i to p_j in $G(\mathcal{C}, \varepsilon_{m-1})$ for all shortcuts $(p_i, p_j) \in \mathcal{S}_m$. Simplification $\mathcal{S}_{m-1}$ is then constructed by linking these paths P with each other. We build all other simplifications in this manner until $\mathcal{S}_1$ is constructed.

If (p_i, p_j) is a valid shortcut in $G(\mathcal{C}, \varepsilon_{k-1})$ for any $1 < k \leq m$, then it follows that $c^k_{i,j} = c^{k-1}_{i,j} + 1$. We prove this in [8].

Correctness

We prove that our simplification algorithm returns a valid and minimal solution for the progressive simplification problem. Let $\langle \mathcal{S}_1, \ldots, \mathcal{S}_m \rangle$ be a sequence of simplifications computed

by our algorithm. By constructing the simplifications from scale m down to 1, it follows that for any shortcut $(p_i, p_j) \in \mathcal{S}_k$ with $1 < k \leq m$, there exists a subsequence $\langle p_i, \ldots, p_j \rangle \sqsubseteq \mathcal{S}_{k-1}$, and thus $\mathcal{S}_k \sqsubseteq \mathcal{S}_{k-1}$. Furthermore, each simplification $\mathcal{S}_k$ has a maximum Hausdorff distance ε_k to $\mathcal{C}$ since it contains only edges from $G(\mathcal{C}, \varepsilon_k)$.

It remains to show that we minimize $\sum_{i=1}^{m} |\mathcal{S}_i|$. We therefore define a set of shortcuts $\mathcal{S}_k^{i,j}$ for any $1 \leq i < j \leq n$ and $1 \leq k \leq m$ as $\mathcal{S}_k^{i,j} = \{\ (p_x, p_y) \in \mathcal{S}_k \mid x \leq i < j \leq y\ \}$.

Thus, $\mathcal{S}_k^{i,j}$ includes all line segments of $\mathcal{S}_k$ that span the subcurve $\langle p_i, \ldots, p_j \rangle$ with an error of at most ε_k to $\mathcal{C}$. $|S_k^{i,j}|$ then is the number of shortcuts in simplification $\mathcal{S}_k$ covering (p_i, p_j).

▶ **Lemma 2.1.** *If the line segment (p_i, p_j) is part of simplification $\mathcal{S}_k$, then the associated cost value $c_{i,j}^k = \sum_{\ell=1}^{k} |\mathcal{S}_\ell^{i,j}|$ for any $1 \leq k \leq m$ and $1 \leq i < j \leq n$.*

Proof. We show $c_{i,j}^k = \sum_{\ell=1}^{k} |\mathcal{S}_\ell^{i,j}|$ by induction on k using the following induction hypothesis: For any $(p_x, p_y) \in \mathcal{S}_k$, it holds that $c_{x,y}^k = \sum_{\ell=1}^{k} |\mathcal{S}_\ell^{x,y}|$ (IH).
Base $k = 1$**:** Take any shortcut $(p_i, p_j) \in \mathcal{S}_1$. It follows that $\mathcal{S}_1^{i,j} = \{(p_i, p_j)\}$, and therefore $|\mathcal{S}_1^{i,j}| = 1$. We deduce that $c_{i,j}^1 = 1 = \sum_{\ell=1}^{k} 1 = \sum_{\ell=1}^{k} |\mathcal{S}_1^{i,j}|$.
Step $k > 1$**:** Take any line segment $(p_i, p_j) \in \mathcal{S}_{k+1}$. Thus, we observe $(p_i, p_j) \in G(\mathcal{C}, \varepsilon_{k+1})$, $\mathcal{S}_{k+1}^{i,j} = \{(p_i, p_j)\}$, and $|\mathcal{S}_{k+1}^{i,j}| = 1$.

Consider any $1 \leq \ell \leq k$ and a path $\pi \in \prod^k(p_i, p_j)$ such that $\sum_{(p_x,p_y)\in\pi} |\mathcal{S}_\ell^{x,y}|$ is minimal. We now derive that $\pi = \mathcal{S}_\ell^{i,j}$ such that $\mathcal{S}_\ell^{x,y}$ is minimal for all $(p_x, p_y) \in \pi$. Note that $\pi = \mathcal{S}_\ell^{i,j} \subseteq G(\mathcal{C}, \varepsilon_\ell) \subseteq G(\mathcal{C}, \varepsilon_k)$ since $\varepsilon_k \geq \varepsilon_\ell$. We observe that π is both in $\prod^\ell(p_i, p_j)$ and $\prod^k(p_i, p_j)$. It thus follows that:

$$\min_{\pi \in \prod_{i,j}^{k}} \sum_{(p_x,p_y)\in\pi} |\mathcal{S}_\ell^{x,y}| = \min_{\pi \in \prod_{i,j}^{\ell}} \sum_{(p_x,p_y)\in\pi} |\mathcal{S}_\ell^{x,y}| \tag{1}$$

From $\pi = \mathcal{S}_\ell^{i,j}$ it follows that $\mathcal{S}_\ell^{x,y} \cap \mathcal{S}_\ell^{y,z} = \emptyset$ for any (p_x, p_y) and (p_y, p_z) in π. Combining $\mathcal{S}_\ell^{x,y}$ for all $(p_x, p_y) \in \pi$ yields a non-overlapping sequence of shortcuts from p_i to p_j. This gives us:

$$|\mathcal{S}_\ell^{i,j}| = \min_{\pi \in \prod_{i,j}^{\ell}} \sum_{(p_x,p_y)\in\pi} |\mathcal{S}_\ell^{x,y}| \tag{2}$$

We now derive the following:

$$c_{i,j}^{k+1} \overset{\text{(IH)}}{=} 1 + \min_{\pi \in \prod_{i,j}^{k}} \sum_{(p_x,p_y)\in\pi} \sum_{\ell=1}^{k} |\mathcal{S}_\ell^{x,y}| \overset{(1)}{=} 1 + \sum_{\ell=1}^{k} \min_{\pi \in \prod_{i,j}^{\ell}} \sum_{(p_x,p_y)\in\pi} |\mathcal{S}_\ell^{x,y}| \overset{(2)}{=} 1 + \sum_{\ell=1}^{k} |\mathcal{S}_\ell^{i,j}|$$
$$\overset{|\mathcal{S}_{k+1}^{i,j}|=\{(p_i,p_j)\}}{=} \sum_{\ell=1}^{k+1} |\mathcal{S}_\ell^{i,j}|$$

◀

▶ **Theorem 2.2.** *Given a polygonal curve with n points in the plane, and $0 \leq \varepsilon_1 < \ldots < \varepsilon_m$, a minimal progressive simplification can be computed in $O(n^3 m)$ time under distance measures for which the validity of a shortcut can be computed in $O(n)$ time. This includes the Fréchet, Hausdorff and area-based measures.*

Proof. It remains to be proven that the combined size of the simplifications computed by our algorithm is minimal. Let $\langle \mathcal{S}'_1, \ldots, \mathcal{S}'_m \rangle$ be a sequence of simplifications of a minimal progressive simplification, and let $\langle \mathcal{S}_1, \ldots, \mathcal{S}_m \rangle$ be the sequence computed by our algorithm.

Let us derive the following:

$$\min_{\pi\in\prod_{1,n}^{m}} \sum_{(p_x,p_y)\in\pi} c_{x,y}^{m} \overset{(2.1)}{=} \min_{\pi\in\prod_{1,n}^{m}} \sum_{(p_x,p_y)\in\pi} \sum_{k=1}^{m} |\mathcal{S}_k^{x,y}| \overset{(1)}{=} \sum_{k=1}^{m} \min_{\pi\in\prod_{1,n}^{\ell}} \sum_{(p_x,p_y)\in\pi} |\mathcal{S}_k^{x,y}| \overset{(2)}{=} \sum_{k=1}^{m} |\mathcal{S}_k|$$

Hence, the algorithm produces a simplification that minimizes the cumulative cost of shortcuts in $\mathcal{S}_m$. Because $\mathcal{S}_{i+1} \sqsubseteq \mathcal{S}_i$; the algorithm produces a set of simplifications in which each simplification consists of edges from the corresponding shortcut graph such that the cumulative number of vertices is minimized.

We further know that any minimal simplification $\mathcal{S}'_k$ is a path in $G(\mathcal{C}, \varepsilon_k)$ since it strictly connects shortcuts with an error of at most ε_k.

We conclude that $\sum_{k=1}^{m} |\mathcal{S}_k| \leq \sum_{k=1}^{m} |\mathcal{S}'_k|$ holds. ◀

3 Continuous and Weighted Progressive Simplification

We now consider two versions of the progressive simplification problem: the *weighted* progressive simplification, where the objective is to minimize $\sum_{k=1}^{m} w_k|\mathcal{S}_k|$ (with $w_k \geq 0$), thus the weighted cumulative size of the simplifications; and the *continuous* progressive simplification, which is an instance of the weighted progressive simplification where $\int_0^m |\mathcal{S}_i|\, d\varepsilon$ is minimal. For both problems, we can employ our preceding algorithm to compute simplifications progressively. We first show how to adapt our algorithm for the weighted progressive simplification problem; then we prove how to solve the continuous simplification problem.

For the weighted progressive simplification, we use the following cost function for each shortcut $(p_i, p_j) \in G(\mathcal{C}, \varepsilon_k)$: if $k = 1$, $c_{i,j}^{k} = w_1$ else $c_{i,j}^{k} = w_k + \min_{\pi\in\prod_{i,j}^{k-1}} \sum_{(p_x,p_y)\in\pi} c_{x,y}^{k-1}$. Note that the proofs above are trivially extended to apply to this updated cost function. The main reason to consider the weighted case is that it helps us solving the continuous progressive simplification problem.

▶ **Theorem 3.1.** *Given a polygonal curve with n points in the plane, a minimal continuous progressive simplification can be computed in $O(n^5)$ time under distance measures for which the validity of a shortcut can be computed in $O(n)$ time. This includes the Fréchet, Hausdorff and area-based measures.*

Proof. Consider the maximal errors $\varepsilon(p_i, p_j)$ of all possible line segments (p_i, p_j) with $i < j$ with respect to the Hausdorff distance (or another distance measure). Then let $\mathcal{E} := \langle \varepsilon_1, \ldots, \varepsilon_{\binom{n}{2}} \rangle$ be the sorted sequence of these errors based on their value. Let M be the index of the corresponding ε_M in this sorted sequence $\mathcal{E}$ for the line segment (p_1, p_n); thus $\varepsilon_M = \varepsilon(p_1, p_n)$. Note that it is possible that $M < \binom{n}{2}$, but there is no reason to use any $\varepsilon > \varepsilon_M$, since at this point we already have simplified the curve to a single line segment, (p_1, p_n).

In a minimal-size progressive simplification it holds that $\mathcal{S}_\varepsilon = \mathcal{S}_{\varepsilon_i}$ for all $\varepsilon \in [\varepsilon_i, \varepsilon_{i+1})$. This can be shown by contradiction: if $\mathcal{S}_\varepsilon$ would be smaller, we could decrease the overall size by setting all $\mathcal{S}_{\varepsilon'}$ with $\varepsilon' \in [\varepsilon_i, \varepsilon]$ to $\mathcal{S}_\varepsilon$. Therefore, in a minimal continuous progressive simplification we have $\int_0^{\varepsilon_M} |\mathcal{S}_\varepsilon|\, d\varepsilon = \sum_{k=1}^{M-1} (\varepsilon_{k+1} - \varepsilon_k)|\mathcal{S}_{\varepsilon_k}|$. Thus, we can solve the continuous progressive simplification problem by reducing it to the weighted progressive simplification problem with $O(n^2)$ values ε_k. ◀

4 Discussion

We present the first algorithm to compute minimum-complexity progressive simplifications given a polygonal curve with n points in the plane. Our algorithm runs in $O(n^3m)$ time for m discrete scales and $O(n^5)$ time for continuous scaling.

In the following, we survey further results from [8]. To facilitate progressive simplifications on many scales, in [8] we present a technique for computing all $\varepsilon(p_i, p_j)$ efficiently in $O(n^2 \log n)$ time instead of $O(n^3)$ time [3]. This is in particular useful for continuous progressive simplification, where we would otherwise need to compute a quadratic number of shortcut graphs, thus spending $O(n^4)$ time on computing shortcut graphs.

Furthermore, we developed a storage-efficient representation of the shortcut graph that is capable of finding shortest paths in $O(n \log n)$ time, which is also applicable to any simplification algorithm that uses a shortcut graph.

The experimental evaluation on a trajectory of a migrating griffon vulture shows that our progressive algorithm is effective, yet too slow for larger trajectory data, and provides similar cumulative simplification sizes as an optimal non-progressive simplification algorithm. We discuss all experiments, algorithms, and results in [8].

As future work, it would be of interest to improve the running time of the minimal progressive simplification algorithm to facilitate real-world application.

Acknowledgments. We thank Michael Horton for our discussions on this topic. Kevin Buchin and Maximilian Konzack are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.207.

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *IJCGA*, 5(1–2):78–99, 1995.

2 Hu Cao, Ouri Wolfson, and Goce Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDB J*, 15(3):211–228, 2006. doi:10.1007/s00778-005-0163-7.

3 Wing Shiu Chan and F Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *IJCGA*, 6(01):59–77, 1996.

4 Shervin Daneshpajouh, Mohammad Ghodsi, and Alireza Zarei. Computing polygonal path simplification under area measures. *Graphical Models*, 74(5):283–289, 2012.

5 David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.

6 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. Elsevier, 1988.

7 Guo Qingsheng, Christoph Brandenberger, and Lorenz Hurni. A progressive line simplification algorithm. *Geo-spatial Information Science*, 5(3):41–45, 2002.

8 Wim Reddingius. Progressive minimum-link simplification of polygonal curves. Master's thesis, Eindhoven University of Technology, 2017. URL: https://tue.on.worldcat.org/oclc/1016161122.

9 Maheswari Visvalingam and James D Whyatt. Line generalisation by repeated elimination of points. *Cartogr J*, 30(1):46–51, 1993.

Drawing Connected Planar Clustered Graphs on Disk Arrangements *†

Tamara Mchedlidze[1], Marcel Radermacher[1], Ignaz Rutter[2], and Nina Zimbel[1]

1 **Department of Computer Science, Karlsruhe Institute of Technology, Germany** mched@iti.uka.de, radermacher@kit.edu

2 **Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands,** i.rutter@tue.nl

Abstract

Let $G = (V, E)$ be a planar graph and let $\mathcal{V}$ be a partition of V whose *clusters*, i.e., the graphs induced by the vertex sets in $\mathcal{V}$, are connected. Let $\mathcal{D}_\mathcal{C}$ be an arrangement of disks with a bijection between the disks and the clusters. Akitaya et al. [1] give an algorithm to test whether $(G, \mathcal{V})$ can be embedded onto $\mathcal{D}_\mathcal{C}$ with the additional constraint that edges are routed through an additional set of pipes between the disks. Based on such an embedding, we prove that every clustered graph with connected clusters and every disk-arrangement with non-overlapping disks has a planar straight-line drawing where every vertex is embedded in the disk corresponding to its cluster. This result can be seen as an extension of the result by Alam et al. [2] who solely consider biconnected clusters.

1 Introduction

In this paper, we study the problem of drawing a large plane clustered graph G on a prescribed disk arrangement $\mathcal{D}_\mathcal{C}$. More formally, a *(flat) clustering* of a graph $G = (V, E)$ is a partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of the vertex set V. We refer to the pair $\mathcal{C} = (G, \mathcal{V})$ as a *clustered graph* and the graphs G_i induced by V_i as *clusters*. A *disk arrangement* $\mathcal{D} = \{d_1, \ldots, d_k\}$ is a set of pairwise disjoint disks in the plane together with a bijective mapping $\mu(V_i) = d_i$ between the clusters $\mathcal{C}$ and the disks $\mathcal{D}$. We refer to a disk arrangement $\mathcal{D}$ with a bijective mapping μ as a *disk arrangement of* $\mathcal{C}$, denoted by $\mathcal{D}_\mathcal{C}$. A *$\mathcal{D}_\mathcal{C}$-framed drawing of* $\mathcal{C}$ is a planar drawing of a clustered graph $\mathcal{C}$ where each cluster G_i is drawn within its corresponding disk d_i. We study the following problem: given a clustered planar graph G with an embedding ψ and a disk arrangement $\mathcal{D}_\mathcal{C}$ of $\mathcal{C}$, does G admit a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing homeomorphic to ψ?

A *pipe p_{ij} of two clusters V_i, V_j* is the *convex hull* of the disks d_i and d_j, i.e., the smallest convex set of points containing d_i and d_j; see Fig. 1. A disk arrangement $\mathcal{D}_\mathcal{C}$ of $\mathcal{C}$ is *planar* if (i) the pairwise intersections of all disks are empty, and (ii) if $(V_i \times V_j) \cap E \neq \emptyset$, then the intersection of p_{ij} with all disks d_k (corresponding to V_k) is empty (i, j, k pairwise distinct) and, (iii) if $(V_i \times V_j) \cap E \neq \emptyset$ and $(V_k \times V_l) \cap E \neq \emptyset$ (i, j, k, l pairwise distinct), then the intersection of the pipes p_{ij} and p_{kl} is empty. A planar disk arrangement can be seen as a thickening of the graph obtained by contracting all clusters in $\mathcal{C}$. An *embedding* ψ of G, i.e., a topological planar drawing of G, is *compatible* with a planar disk arrangement $\mathcal{D}_\mathcal{C}$ if ψ is homeomorphic to a $\mathcal{D}_\mathcal{C}$-framed embedding of $\mathcal{C}$ such that edges of a cluster are routed within the corresponding disks, and edges between distinct clusters are routed through the

* Work was partially supported by grant WA 654/21-1 of the German Research Foundation (DFG).
† This research was funded in part by Humility & Conviction in Public Life, a project of the University Connecticut sponsored by the John Templeton Foundation.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

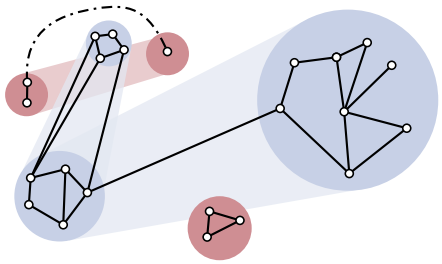


Figure 1 The blue disk arrangement is planar. The red disk arrangement disrupts the planarity of the entire arrangement. The dash dotted edge is not embedded in a pipe, hence the embedding is not compatible with the disk arrangement.

corresponding pipes. Throughout the paper we assume the disk arrangement, provided as part of the input, is planar.

Related Work

Feng et al. [7] introduced the notion of *clustered graphs* and *c-planarity*. A graph G together with a recursive partitioning of the vertex set is considered to be a clustered graph. An embedding of G is a *c-planar embedding* if (i) each cluster c is drawn within a connected region R_c, (ii) two regions R_c, R_d intersect if and only if the cluster c contains the cluster d or vice versa, and (iii) every edge intersects the boundary of a region at most once. They prove that a c-planar embedding of a connected clustered graph can be computed in $O(n^2)$ time. It is an open question whether it is possible to extend this result to disconnected clustered graphs. Many special cases of this problem have been considered [4].

Concerning drawings of c-planar clustered graphs, Eades et al. [6] prove that every c-planar graph has a c-planar straight-line drawing where each cluster is drawn in a convex region. Angelini et al. [3] strengthen the result of Eades et al. by showing that every c-planar graph has a c-planar straight-line drawing in which every cluster is drawn in an axis-parallel rectangle. The result of Akitaya et al. [1] implies that in $O(n \log n)$ time one can decide whether an abstract graph with a flat clustering has an embedding where each vertex lies in a prescribed topological disk and every edge is routed through a prescribed topological pipe. In general their algorithm decides whether a simplicial map φ of G onto a 2-manifold M is a *weak embedding*, i.e., for every $\epsilon > 0$, φ can be perturbed into an embedding ψ_ϵ with $||\varphi - \psi_\epsilon|| < \epsilon$.

Alam et al. [2] prove that it is $\mathcal{NP}$-hard to decide whether a clustered graph has a c-planar straight-line drawing where every cluster is contained in a prescribed rectangle and edges have to pass through a defined part of the boundary of the rectangle. Further, they prove that all instances with biconnected clusters always admit a solution. Their result implies that graphs of this class have $\mathcal{D}_\mathcal{C}$-framed straight-line drawings.

Contribution

In this paper, we prove that every *connected clustered graph* $(G, \mathcal{V})$, i.e., each cluster G_i is connected, with an embedding ψ compatible with a prescribed planar disk arrangement $\mathcal{D}_\mathcal{C}$, has a $\mathcal{D}_\mathcal{C}$-framed planar straight-line drawing homeomorphic to ψ. Taking the result of Akitaya et al. [1] into account, our result can be used to test whether an abstract clustered graph with connected clusters has a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing. Our result is an extension of the result of Alam et al. [2] from biconnected to connected clusters.

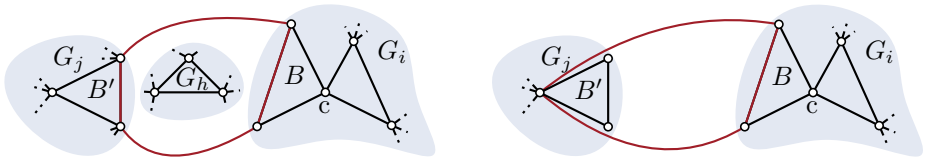


Figure 2 (a) A planar clustered graph $\mathcal{C}$ that is not simple. (b) The block B is leaf block of G_i. The block B' of G_j obstructs B, B' itself is free. The cycles mentioned in the definitions are highlighted in red.

2 Preliminaries

A clustered graph $\mathcal{C} = (G, \mathcal{V})$ is *simple* if for every i, j, there is no cluster $G_h (i, j \neq h)$ embedded in the interior of the subgraph induced by $V_i \cup V_j$; see Fig. 2a. Note that this is a necessary condition in our model, as otherwise the corresponding disk arrangement would not be planar. The set of edges E_i of a cluster G_i are *intra-cluster edges* and the set of edges with endpoints in different clusters *inter-cluster edges*. The vertex u of an inter-cluster edge uv is the *inter-cluster neighbor of* v.

We refer to a maximal biconnected component B of G_i as a *block of* G_i. Removing a *cut vertex* from G_i, splits G_i into two connected components. A block is a *leaf block* if it is incident to at most one cut vertex of G_i; see Fig. 2b. A block B' of a cluster G_j *obstructs* a leaf block of G_i in ψ if there is a cycle C using only vertices of B and at most a single vertex of B' such that B' is in the interior of the graph induced by $C \cup B \cup B'$. A block B that is not obstructed by another block is *free*. We denote the graph after the contraction of a block B by G/B and refer to the resulting vertex b as the *contraction vertex of* G/B. The contraction of a block in a graph with an embedding ψ *induces an embedding* $\psi_{G/B}$ of G/B.

▶ **Lemma 2.1.** *Let $\mathcal{C} = (G, \mathcal{V})$ be a connected simple clustered graph with an embedding ψ that is compatible with a disk arrangement $\mathcal{D}_\mathcal{C}$. Then the embedding induced by the contraction of a free leaf block is compatible with $\mathcal{D}_\mathcal{C}$.*

3 Drawing Planar Clustered Graphs on Disk Arrangements

In this section, we prove that every connected simple clustered graph $\mathcal{C}$ has a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing, see Theorem 3.6. Our proof strategy is as follows. We iteratively contract free leaf blocks B of $\mathcal{C}$ until every cluster contains exactly one vertex, see Lemma 3.1. In this case, the center points of the disks in the disk arrangement $\mathcal{D}_\mathcal{C}$ induce a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing of $\mathcal{C}$. In order to undo a contraction of a free leaf block B, we consider a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing $\Gamma_{\mathcal{C}/B}$ of the contracted graph $\mathcal{C}/B$, see Fig 3b. We start by defining a safe convex polygon σ, that allows us to extend the drawing $\Gamma_{\mathcal{C}/B}$ to a drawing Γ of $\mathcal{C}$, by placing vertices on the boundary of B on the boundary of σ, and the interior vertices of B in the interior of σ. The result of Chambers et al. [5] ensures that the drawing of B, where the vertices on the boundary of B have prescribed placements on the boundary of a convex polygon, is a planar straight-line drawing homeomorphic to the embedding of B. The challenging part is to guarantee that the inter-cluster edges do not intersect with edges of B; see Lemma 3.2 to Lemma 3.5. We first prove that unless the clustered graph is not sufficiently small, there is a free leaf block B.

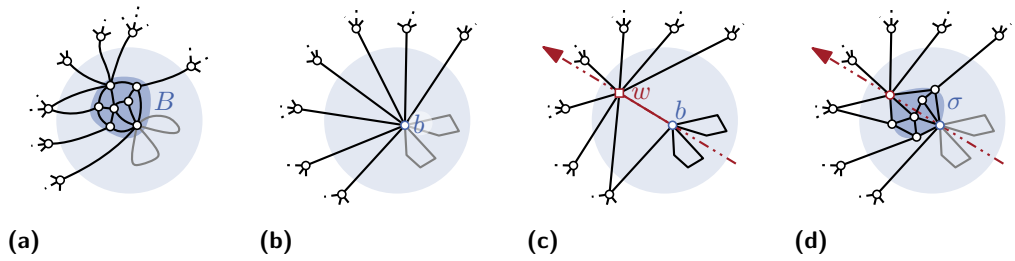


Figure 3 (a) A block B (black) with inter-cluster neighbors outside of the blue disk. (b) A straight-line drawing of the B-contracted graph. (c) A U_b-similar segment $\overline{bw}$ with its supporting line (red). (d) $\mathcal{D}_\mathcal{C}$-framed straight-line drawing with B drawn in the dark blue convex polygon σ.

▶ **Lemma 3.1.** *Every connected simple clustered graph $\mathcal{C} = (G, \mathcal{V})$ has a cluster G_i with a free leaf block or every cluster has exactly one vertex.*

Let B be a free leaf block of a cluster G_i and consider a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing $\Gamma_{\mathcal{C}/B}$ of a B-contracted clustered graph $\mathcal{C}/B$. Observe that we cannot take an arbitrary convex polygon σ to extend the drawing $\Gamma_{\mathcal{C}/B}$ to a drawing Γ, since for this polygon it might not be possible to avoid intersections between inter-cluster edges and edges of B. To avoid these intersections, we construct the polygon σ in two phases. First, we will prove the existence of a special segment s (see Fig 3c), that we will later use to construct two polygons σ_L and σ_R. Then the union of σ_L and σ_R will be the desired polygon σ.

We formalize the concept of a safe point set as follows. Denote by U_b the inter-cluster neighbors of the contraction vertex b and let $L \subseteq U_b$ be a set of vertices that is consecutive in the clockwise order around b. We construct an *L-split drawing* Γ_p from $\Gamma_{\mathcal{C}/B}$ by removing the inter-cluster edges $\{bu \mid u \in L\}$ from $\Gamma_{\mathcal{C}/B}$ and adding a *split vertex* w at position $p \in \mathbb{R}^2$ and connecting w to all vertices in $L \cup \{b\}$ with straight-line edges. We say a set $P \subseteq \mathbb{R}^2$ is *L-similar* if for every point $p \in P$ the L-split drawing Γ_p of $\Gamma_{\mathcal{C}/B}$ is planar, and the contraction of the edge bw induces an embedding homeomorphic to $\Gamma_{\mathcal{C}/B}$.

▶ **Lemma 3.2.** *Let B be a free leaf block of a cluster G_i and let $d_i \in \mathcal{D}_\mathcal{C}$ be the corresponding disk. Let $\Gamma_{\mathcal{C}/B}$ be a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing of $\mathcal{C}/B$. Let b be the contraction vertex of $\mathcal{C}/B$ with inter-cluster neighbors U_b. Then there is a U_b-similar straight-line segment $s \subset d_i$.*

Proof sketch. There is a small disk $\delta \subset d_i$ around b such that moving b within δ preserves the topological properties of b. Let e_l be and e_r be the edges that precede and succeed B, respectively. Then, the two lines containing e_l and e_r divide δ into four regions of which one region R is U_b-similar. Thus, every segment $\overline{ba}$, with $a \in R$, is U_b-similar. □

A *supporting line* of a U_b-similar segment $s = \overline{ba}$ is the line that contains s and is directed from b towards a. This line l separates the set U_b into sets L and R, such that the vertices in L are to left of l in the drawing $\Gamma_{\mathcal{C}/B}$, and the vertices in R to the right of l. Depending on the set, we show that there are convex polygons σ_L and σ_R that are monotone with respect to s. For a segment $s = \overline{ba}$, a convex polygon $\langle p_0, p_1, \ldots, p_k, p_{k+1} \rangle$, with $p_0 = a$ and $p_{k+1} = b$, is *s-monotone* if the projections of all p_i onto the supporting line of s, lie on s.

▶ **Lemma 3.3.** *Let $\Gamma_{\mathcal{C}/B}$ be a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing of $\mathcal{C}/B$ and let U_b be the inter-cluster neighbors of the contraction vertex b and let $s = \overline{ba}$ be a U_b-similar segment. Let $L \subseteq U_b$ be the set of vertices that are to the left of the supporting-line of s. Then there is a convex s-monotone polygon σ_L contained in $d_i \in \mathcal{D}_\mathcal{C}$ such that the boundary $\mathcal{BD}(\sigma_L)$ of σ_L is L-similar, and for every point p on $\mathcal{BD}(\sigma_L) \setminus s$ and every vertex $u \in L$, the open segment $\overline{pu}$ and σ_L do not intersect.*

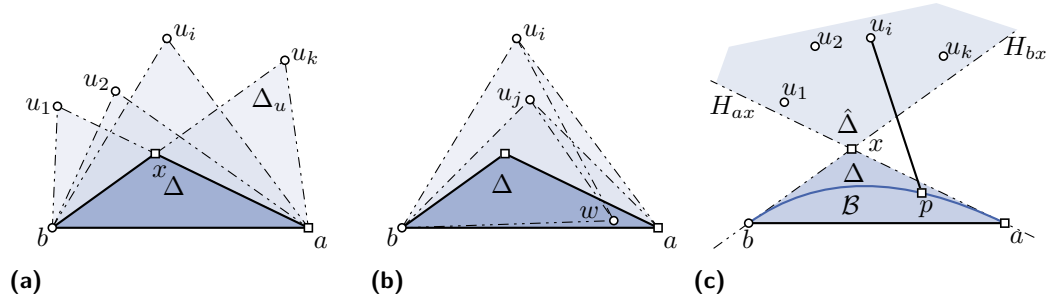


Figure 4 (a) Triangle Δ is the intersection of all triangles Δ_u. (b) Δ is not L-similar. (c) $\mathcal{B}$ is a Bézier-curve within Δ.

Proof sketch. Consider the non-empty set L and the triangle Δ_u with vertices b, u, a for a vertex $u \in L$; see Fig. 4a. Let Δ be the intersection of all triangles Δ_u. Since the segment $s = \overline{ba}$ is U_b-similar and the set L contains all vertices to the left of l, Δ is L-similar. Unfortunately, the triangle $\Delta = (b, x, a)$ is not the desired polygon σ_L, yet. To ensure that the polygon σ_L is s-monotone and entirely contained in d_i, we place the vertex x in the intersection of Δ and d_i, such that the projection of x lies on s. Such a point exists, since s is contained in d_i. Finally, we have to guarantee that for every point p in $\mathcal{BD}(\sigma_L) \setminus s$ and every vertex u in L, the open segment $\overline{pu}$ and σ_L do not intersect. Indeed the Bézier-curve $\mathcal{B}$ with b, x, a as its control points satisfies this property. Hence, the desired polygon σ_L can be constructed by discretizing the curve $\mathcal{B}$. □

Observe that this lemma can be restated in terms of the set R right of the supporting line l of s. We then obtain an s-monotone polygon σ_R. Merging the two polygons σ_L and σ_R results in the final polygon σ. Before we are able to actually draw the block B on σ, it is crucial that the notion of vertices to the left and right of a supporting line l transfers to the vertices on the boundary of B. We formalize this with the concept of an *apex vertex of* B. Let $v_0, v_1, \ldots, v_k, v_{k+1}$ be the vertices on the boundary of B, with $v_0 = v_{k+1}$ the cut vertex of B. A vertex v_i is called an *apex vertex of* B *with respect to* $\Gamma_{\mathcal{C}/B}$ *and* l if all inter-cluster neighbors of the vertices in $v_1, \ldots v_{i-1}$ are to left of l in $\Gamma_{\mathcal{C}/B}$ and the inter-cluster neighbors of the vertices $v_{i+1}, \ldots, v_k$ are to the right of l in $\Gamma_{\mathcal{C}/B}$.

▶ **Lemma 3.4.** *Let B be a free leaf block of a clustered graph $\mathcal{C}$ with an embedding ψ. Let $\Gamma_{\mathcal{C}/B}$ be a planar straight-line drawing homeomorphic to the induced embedding of $\mathcal{C}/B$ and let l be the supporting line of a U_b-similar segment. Then there is an apex vertex of B with respect to $\Gamma_{\mathcal{C}/B}$ and l.*

Proof sketch. Since $\Gamma_{\mathcal{C}/B}$ is a straight-line drawing homeomorphic to the embedding induced by the contraction of B, the neighbors of b in $\mathcal{C}/B$ appear in the same clockwise order as in a clockwise traversal of all neighbors of vertices on the boundary of B in $\mathcal{C}$. Thus, the partitioning of the neighborhood of b into the left and right of l transfers to the vertices on the boundary of B. □

With this framework at hand, we are now able to prove that $\mathcal{C}$ has $\mathcal{D}_{\mathcal{C}}$-framed straight-line drawing, if the B-contracted clustered graph $\mathcal{C}/B$ has a $\mathcal{D}_{\mathcal{C}}$-framed straight-line drawing $\Gamma_{\mathcal{C}/B}$. Thus, let L be the set of inter-cluster neighbors to the left of the supporting-line l of a U_b-similar segment s, and let R be the corresponding set to the right of l. We obtain two polygons σ_L and σ_R by the application of Lemma 3.3. We obtain a convex polygon σ by merging σ_L and σ_R at the common side s. An apex vertex v_i splits the vertices on the boundary of B. We place the vertices $v_0, \ldots, v_{i-1}$ on the boundary of the polygon σ_L and

$v_{i+1}, \ldots, v_k$ on the boundary of σ_R. The apex v_i is placed at the end a of the U_b-similar segment $s = \overline{ba}$ where it can be connected to vertices in L and in R. Since σ is a convex polygon, we can extend this drawing to a drawing Γ of $\mathcal{C}$ by drawing the remaining vertices of B in the interior of σ with the result of Chambers et al. [5]. We get the following result.

▶ **Lemma 3.5.** *Let $\mathcal{C} = (G, \mathcal{V})$ be a connected simple clustered graph with an embedding ψ that is compatible with a disk arrangement $\mathcal{D}_\mathcal{C}$. If B is a free leaf block of $\mathcal{C}$ and $\mathcal{C}/B$ has $\mathcal{D}_\mathcal{C}$-framed straight-line drawing homeomorphic to the embedding induced by the contraction of B, then $\mathcal{C}$ has a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing.*

Note that, if every cluster contains exactly one vertex, then the center points of the disks in the planar disk arrangement $\mathcal{D}_\mathcal{C}$ induce a planar straight-line drawing of $\mathcal{C}$. Thus, we can inductively apply the previous lemma to prove our main theorem.

▶ **Theorem 3.6.** *Every connected simple clustered graph $\mathcal{C} = (G, \mathcal{V})$ with a planar embedding ψ that is compatible with a disk arrangement $\mathcal{D}_\mathcal{C}$ has a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing that is homeomorphic to ψ.*

4 Conclusion

We proved that every clustered planar graph with an embedding compatible with a planar disk arrangement has a $\mathcal{D}_\mathcal{C}$-framed straight-line drawing. If the requirement of the disk arrangement to be planar is dropped, not every clustered-planar graph has $\mathcal{D}_\mathcal{C}$-framed straight-line drawing. Thus, we ask what is the complexity of deciding whether a clustered planar embedded graph has $\mathcal{D}_\mathcal{C}$-framed straight-line drawing for a given non-planar disk arrangement $\mathcal{D}_\mathcal{C}$?

References

1 Hugo A. Akitaya, Radoslav Fulek, and Csaba D. Tóth. Recognizing Weak Embeddings of Graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 274–292. Society for Industrial and Applied Mathematics, 2018. doi:10.1137/1.9781611975031.20.

2 Md. Jawaherul Alam, Michael Kaufmann, Stephen G. Kobourov, and Tamara Mchedlidze. Fitting Planar Graphs on Planar Maps. *Journal of Graph Algorithms and Applications*, 19(1):413–440, 2015. doi:10.7155/jgaa.00367.

3 Patrizio Angelini, Fabrizio Frati, and Michael Kaufmann. Straight-Line Rectangular Drawings of Clustered Graphs. *Discrete & Computational Geometry*, 45(1):88–140, 2011. doi:10.1007/s00454-010-9302-z.

4 Thomas Bläsius and Ignaz Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. *Theoretical Computer Science*, 609(2):306 – 315, 2016. doi:10.1016/j.tcs.2015.10.011.

5 Erin W. Chambers, David Eppstein, Michael T. Goodrich, and Maarten Löffler. Drawing Graphs in the Plane with a Prescribed Outer Face and Polynomial Area. *Journal of Graph Algorithms and Applications*, 16(2):243–259, 2012. doi:10.7155/jgaa.00257.

6 Peter Eades, Qingwen Feng, Xuemin Lin, and Hiroshi Nagamochi. Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. *Algorithmica*, 44(1):1–32, 2006. doi:10.1007/s00453-004-1144-8.

7 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for Clustered Graphs. In Paul Spirakis, editor, *Proceedings of the Third Annual European Symposium on Algorithms (ESA'95)*, pages 213–226. Springer Berlin/Heidelberg, 1995. doi:10.1007/3-540-60313-1_145.

Arrangements of Pseudocircles: On Circularizability*

Stefan Felsner[1] and Manfred Scheucher[1]

1 **Technische Universität Berlin, Germany**
{felsner,scheucher}@math.tu-berlin.de

Abstract

An arrangement of pseudocircles is a collection of simple closed curves on the sphere or in the plane such that every pair is either disjoint or intersects in exactly two crossing points. We call an arrangement intersecting if every pair of pseudocircles intersects twice. An arrangement is circularizable if there is a combinatorially equivalent arrangement of circles.

Kang and Müller showed that every arrangement of at most 4 pseudocircles is circularizable. Linhart and Ortner found an arrangement of 5 pseudocircles which is not circularizable.

We show that there are exactly four non-circularizable arrangements of 5 pseudocircles, exactly one of them is intersecting. For $n = 6$, we show that there are exactly three non-circularizable digon-free intersecting arrangements. We also have some additional examples of non-circularizable arrangements of 6 pseudocircles.

The claims that we have all non-circularizable arrangements with the given properties are based on a program that generated all connected arrangements of $n \leq 6$ pseudocircles and all intersecting arrangements of $n \leq 7$ pseudocircles. Given the complete lists of arrangements, we used heuristics to find circle representations. Examples where the heuristics failed had to be examined by hand.

1 Introduction

Arrangements of pseudocircles generalize arrangements of circles in the same vein as arrangements of pseudolines generalize arrangements of lines. The study of arrangements of pseudolines was initiated 1918 with an article of Levi [10]. Since then arrangements of pseudolines were intensively studied and the handbook article on the topic [2] lists more than 100 references. The study of arrangements of pseudocircles was initiated by Grünbaum [8].

A *pseudocircle* is a simple closed curve in the plane or on the sphere. An *arrangement of pseudocircles* is a collection of pseudocircles with the property that the intersection of any two of the pseudocircles is either empty or consists of two points where the curves cross. The *graph of an arrangement* $\mathcal{A}$ of pseudocircles has the intersection points of pseudocircles as *vertices*, the vertices split each of the pseudocircles into arcs, these are the *edges* of the graph. Note that this graph may have multiple edges and loop edges without vertices. The graph of an arrangement of pseudocircles comes with a plane embedding, the faces of this embedding are the *cells* of the arrangement. A cell with k crossings on its boundary is a *k-cell*. A 2-cell is also called a *digon* (some authors call it a *lense*), and a 3-cell is also called a *triangle*. An arrangement $\mathcal{A}$ of pseudocircles is

simple, if no three pseudocircles of $\mathcal{A}$ intersect in a common point.
connected, if the graph of the arrangement is connected.

* Partially supported by DFG Grant FE 340/11-1. Manfred Scheucher was partially supported by the ERC Advanced Research Grant no. 267165 (DISCONV). The authors gratefully acknowledge the computing time granted by TBK Automatisierung und Messtechnik GmbH and by the Institute of Software Technology, Graz University of Technology.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

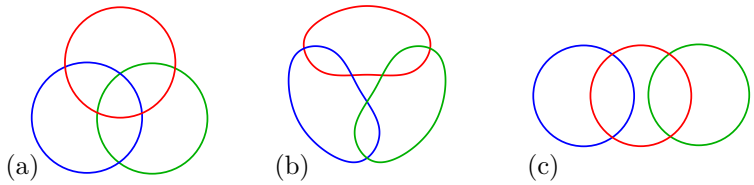


Figure 1 The 3 arrangements of $n = 3$ pseudocircles: (a) *Krupp*, (b) *NonKrupp*, (c) *3-Chain*.

intersecting, if any two pseudocircles of $\mathcal{A}$ intersect.
cylindrical, if there are two cells in $\mathcal{A}$ which are separated by each of the pseudocircles.
Note that every intersecting arrangement is connected. In this paper we assume that arrangements are simple and connected.

Two arrangements $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic* if they induce homeomorphic cell decompositions of the plane respectively the sphere. Figure 1 shows the three connected arrangements of three pseudocircles. We call the unique digon-free intersecting arrangement of three (pseudo)circles the *Krupp*[1]. The second intersecting arrangement of three pseudocircles is the *NonKrupp*, this arrangement has digons. The non-intersecting arrangement is the *3-Chain*.

Every triple of great-circles on the sphere induces a Krupp arrangement, hence, we call an intersecting arrangement of pseudocircles an *arrangement of great-pseudocircles* if every subarrangement induced by three pseudocircles is a Krupp.

Some authors think of arrangements of great-pseudocircles when they speak about arrangements of pseudocircles, this is e.g. common practice in the theory of oriented matroids. In fact, arrangements of great-pseudocircles serve to represent rank 3 oriented matroids.

▶ Definition. An arrangement of pseudocircles is *circularizable* if there is an isomorphic arrangement of circles.

Circularizability of arrangements of pseudocircles has not been studied extensively. This paragraph describes the state of the art. Edelsbrunner and Ramos [1] proved non-circularizability of an arrangement of 6 pseudocircles with digons. Linhart and Ortner [11] found a non-intersecting arrangement of 5 pseudocircles with digons which is non-circularizable (Figure 2b). They also proved that every intersecting arrangement of at most 4 pseudocircles is circularizable. Kang and Müller [9] extended the result by showing that all arrangements with at most 4 pseudocircles are circularizable. They also proved that deciding circularizability of connected arrangements is NP-hard. Since stretchability is $\exists\mathbb{R}$-complete, their proof actually implies $\exists\mathbb{R}$-completeness of circularizability.

In our last year's EuroCG contribution [6] we have sketched non-circularizability of two further intersecting arrangements on 5 and 6 pseudocircles, respectively, namely $\mathcal{N}_5^1$ and $\mathcal{N}_6^\Delta$ (see Figures 2a and 3a). Since then, we have extended our results and got the following.

▶ **Theorem 1.1.** *The four equivalence classes of arrangements $\mathcal{N}_5^1$, $\mathcal{N}_5^2$, $\mathcal{N}_5^3$, and $\mathcal{N}_5^4$ (shown in Figure 2) are the only non-circularizable ones among the 984 equivalence classes of connected arrangements of $n = 5$ pseudocircles.*

▶ **Theorem 1.2.** *The three equivalence classes of arrangements $\mathcal{N}_6^\Delta$, $\mathcal{N}_6^2$, and $\mathcal{N}_6^3$ (shown in Figure 3) are the only non-circularizable ones among the 2131 equivalence classes of digon-free intersecting arrangements of $n = 6$ pseudocircles.*

[1] This name refers to the logo of the Krupp AG, a German steel company. Krupp was the largest company in Europe at the beginning of the 20th century. There is also a disease with the German name Pseudo-Krupp, we have no corresponding arrangement.

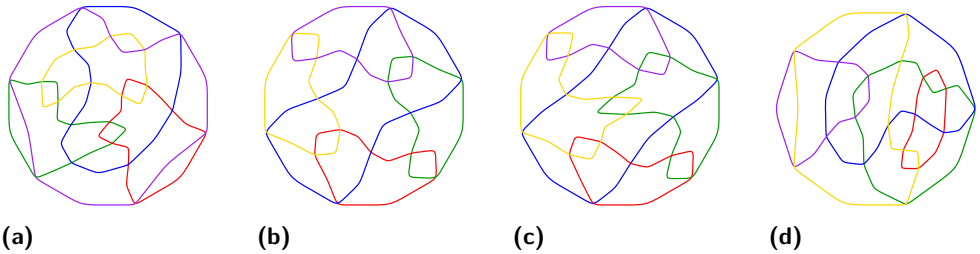


Figure 2 The four non-circularizable arrangements on $n = 5$ pseudocircles. (a) $\mathcal{N}_5^1$. (b) $\mathcal{N}_5^2$. (c) $\mathcal{N}_5^3$. (d) $\mathcal{N}_5^4$.

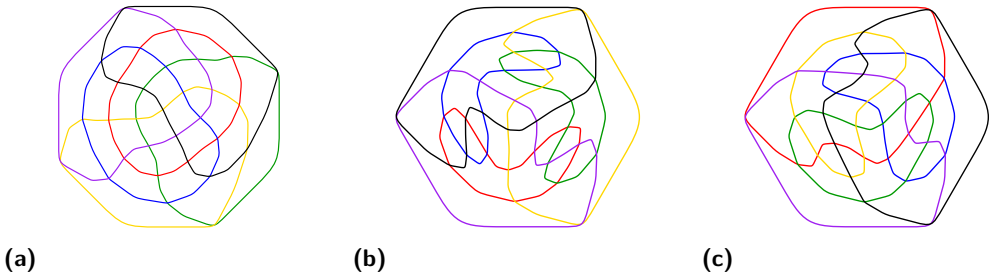


Figure 3 The three non-circularizable digon-free intersecting arrangements for $n = 6$. (a) $\mathcal{N}_6^\Delta$. (b) $\mathcal{N}_6^2$. (c) $\mathcal{N}_6^3$.

Full proofs of Theorems 1.1 and 1.2 can be found in the full version [4], where we also prove non-circularizability of some further interesting arrangements on $n = 6$ pseudocircles and provide some further results for certain classes of arrangements. The non-circularizability proofs use various techniques, most depend on incidence theorems, others use arguments involving metric properties of arrangements of planes, or angles in planar figures.

Our results strongly depend on the generation of the complete lists of connected arrangements of $n \leq 6$ pseudocircles and of intersecting arrangements of $n \leq 7$ pseudocircles. The respective numbers are shown in Table 1. The encoded lists of arrangements up to $n = 6$ are available on our webpage [3]. We remark that the list of intersecting arrangements was already mentioned in our at last year's EuroCG contribution [6]. Computational issues are deferred until Section 5. There we describe the algorithmic ideas behind the computation of the lists.

Particularly interesting is the arrangement $\mathcal{N}_6^\Delta$ (Figure 3a). This is the unique intersecting digon-free arrangement of 6 pseudocircles which attains the minimum 8 for the number of triangles. From our computer search we know that $\mathcal{N}_6^\Delta$ occurs as a subarrangement of every

n	4	5	6	n	4	5	6	7
connected	21	984	609 423	**intersecting**	8	278	145 058	447 905 202
+digon-free	3	30	4 509	+digon-free	2	14	2 131	3 012 972
con.+cylindrical	20	900	530 530	int.+cylindrical		278	144 395	435 367 033
+digon-free		30	4 477	+digon-free			2 131	3 012 906
				great-p.c.s		1	4	11

Table 1 Number of combinatorially different arrangements of n pseudocircles.

digon-free arrangement for $n = 7, 8, 9$ with $p_3 < 2n - 4$ triangles, hence, also neither of those arrangements is circularizable. Therefore, it seems plausible that for every arrangement of n circles $p_3 \geq 2n - 4$. This is the Weak Grünbaum Conjecture. [5, 6]

For the non-circularizability of $\mathcal{N}_6^{\Delta}$ we have two proofs. Due to didactical reasons, we exchanged "first" and "second" in the full version against the actual chronological order.

Our first proof is based on an incidence theorem in 3-space and was already sketched in our last year's EuroCG contribution [6].

Here we sketch our new second proof, which is based on a sweeping argument in 3-D (see Subsection 4). With a similar idea we also show the following theorem, which has some nice corollaries, e.g., it yields a very direct and easy proof that deciding circularizability is $\exists\mathbb{R}$-complete (see Section 3).

▶ **Theorem 1.3** (The Great-Circle Theorem). *An arrangement of great-pseudocircles is circularizable (i.e., has a circle representation) if and only if it has a great-circle representation.*

2 Preliminaries

Stereographic projections map circles to circles (if we consider a line to be a circle containing the point at infinity), therefore, circularizability on the sphere and in the plane is the same concept. Arrangements of circles can be mapped to isomorphic arrangements of circles via Möbius transformations.

Let $\mathcal{C}$ be an arrangement of circles represented on the sphere. Each circle of $\mathcal{C}$ spans a plane in 3-space, hence, we obtain an arrangement $\mathcal{E}(\mathcal{C})$ of planes in $\mathbb{R}^3$. In fact, a fixed sphere S conveys a bijection between (not necessarily connected) circle arrangements on S and arrangements of planes with the property that each plane of the arrangement intersects S.

Consider two circles C_1, C_2 of a circle arrangement $\mathcal{C}$ on S and the corresponding planes E_1, E_2 of $\mathcal{E}(\mathcal{C})$. The intersection of E_1 and E_2 is either empty (i.e., E_1 and E_2 are parallel) or a line ℓ. The line ℓ intersects S if and only if C_1 and C_2 intersect, in fact, $\ell \cap S = C_1 \cap C_2$.

With three pairwise intersecting circles C_1, C_2, C_3 we obtain three planes E_1, E_2, E_3 intersecting in a vertex v of $\mathcal{E}(\mathcal{C})$. It is notable that v is in the interior of the ball bounded by S if and only if the three circles form a Krupp in $\mathcal{C}$.

3 Arrangements of (pseudo) great-circles

Central projections map between arrangements of great-circles on a sphere S and arrangements of lines on a plane. Changes of the plane preserve the isomorphism class of the projective arrangement of lines.

An Euclidean arrangement of n pseudolines can be represented by x-monotone pseudolines, a special representation of this kind is the wiring diagram, see e.g [2]. An x-monotone representation can be glued with a horizontally mirrored copy of itself to form an arrangement of n pseudocircles, see Figure 4. The resulting arrangement is intersecting and has no NonKrupp subarrangement, i.e., it is a great-pseudocircle arrangement.

Indeed the above construction yields a bijection between projective arrangements of n pseudolines in the plane and arrangements of n great-pseudocircles.

Projective arrangements of pseudolines are also known as projective abstract order types or oriented matroids. Their number is known for $n \leq 11$, hence the numbers of great-pseudocircle arrangements given in Table 1 are not new. For more information see [4].

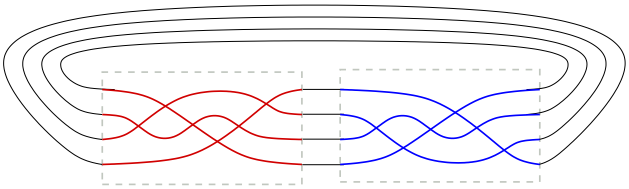


Figure 4 Obtaining an arrangement of pseudocircles from an Euclidean arrangement $\mathcal{A}$ of pseudolines. Arrangement $\mathcal{A}$ and its mirrored copy are shown in red and blue, respectively.

Let $\mathcal{C}$ be an arrangement of great-pseudocircles and let $\mathcal{A}$ be the corresponding projective arrangement of pseudolines. Central projections show that, if $\mathcal{A}$ is realizable with straight lines, then $\mathcal{C}$ is realizable with great-circles, and conversely.

In fact, it is enough that $\mathcal{C}$ is circularizable to conclude that $\mathcal{C}$ is realizable with great-circles and $\mathcal{A}$ is realizable with straight lines.

Proof of Theorem 1.3. Consider an arrangement of circles $\mathcal{C}$ on the unit sphere S that realizes an arrangement of great-pseudocircles. Let $\mathcal{E}(\mathcal{C})$ be the arrangement of planes spanned by the circles of $\mathcal{C}$. Since $\mathcal{C}$ realizes an arrangement of great-pseudocircles, every triple of circles forms a Krupp, hence, the point of intersection of any three planes of $\mathcal{E}(\mathcal{C})$ is in the interior of S.

Imagine the radius of the sphere growing with the time t, to be precise, let $S_1 = S$ and $S_t = t \cdot S$. Since all the intersection points of the arrangement $\mathcal{E}(\mathcal{C})$ are in the interior of S_1, the circle arrangement obtained by intersecting $\mathcal{E}(\mathcal{C})$ with the growing sphere remains the same (isomorphic). Also every circle of the arrangement is moving towards a great-circle while the sphere is growing. When t is large enough it is possible to push all circles a small amount to make them great-circles without changing the arrangement. ◀

▶ **Corollary 3.1.** *Every non-stretchable arrangement of pseudolines has a corresponding non-circularizable arrangement of pseudocircles.*

In particular, the hardness of stretchability directly carries over to hardness of circularizability. Moreover, since there are infinite families of minimal non-stretchable arrangements of pseudolines [7], the same is true for pseudocircles.

It is known that Mnëv's Universality Theorem [12] has strong implications for pseudoline arrangements and stretchability. This together with results from Suvorov [13] directly translates to:

▶ **Corollary 3.2.** *The problem of deciding circularizability is $\exists\mathbb{R}$-complete. Moreover, there exist circularizable arrangements of pseudocircles with a disconnected realization space.*

4 Non-circularizability of $\mathcal{N}_6^\Delta$

Our second proof of non-circularizability of $\mathcal{N}_6^\Delta$ is an immediate consequence of the following theorem, which resembles the proof of the Great-Circle Theorem (Theorem 1.3).

▶ **Theorem 4.1.** *Let $\mathcal{A}$ be a connected digon-free arrangement of pseudocircles with the property that every triple of pseudocircles, which forms a triangles in $\mathcal{A}$, is NonKrupp. Then $\mathcal{A}$ is not circularizable.*

Proof (second proof of non-circularizability of $\mathcal{N}_6^\Delta$). The arrangement $\mathcal{N}_6^\Delta$ is intersecting, digon-free, and each of the eight triangles of $\mathcal{N}_6^\Delta$ is a NonKrupp, hence, Theorem 4.1 implies that $\mathcal{N}_6^\Delta$ is not circularizable. ◀

5 Computational Part

To produce the database of all intersecting arrangements of up to $n = 7$ pseudocircles, we used the dual graphs and a procedure, which generates the duals of all possible extensions by one additional pseudocircle of a given arrangement, starting with the unique arrangement of two intersecting pseudocircles [4, 5]. Another way to obtain the database for a fixed value of n, is to perform a recursive search in the flip graph using the triangle flip operation.

For connected arrangement the dual graph might contain multiple edges. To avoid problems with non-unique embeddings, we modeled connected arrangements with their primal-dual-graphs where vertices, segments, and faces of the arrangement are represented by a vertex in the graph and two vertices share an edge if the corresponding entities are incident and one of them corresponds to an edge. To generate the database of all connected arrangements for $n \leq 6$, we used the fact that the flip graph is connected when triangle flips and digon flips are used. The enumeration was done by a recursive search on the flip graph.

Having generated the database of arrangements of pseudocircles, we were then interested in identifying the circularizable and the non-circularizable ones. To find circle representations we used computer assistance. Examples where our programs failed to find realizations had to be examined by hand. For more information, we refer to the full version [4].

References

1 H. Edelsbrunner and E. A. Ramos. Inclusion-exclusion complexes for pseudodisk collections. *Discrete & Computational Geometry*, 17:287–306, 1997.

2 S. Felsner and J. E. Goodman. Pseudoline Arrangements. In Toth, O'Rourke, and Goodman, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 3 edition, 2018.

3 S. Felsner and M. Scheucher. Homepage of Pseudocircles. `http://www3.math.tu-berlin.de/pseudocircles`.

4 S. Felsner and M. Scheucher. Arrangements of Pseudocircles: On Circularizability. arXiv/1712.02149, 2017.

5 S. Felsner and M. Scheucher. Arrangements of Pseudocircles: Triangles and Drawings. arXiv/1708.06449, 2017.

6 S. Felsner and M. Scheucher. Triangles in Arrangements of Pseudocircles. In *Proc. EuroCG 2017*, pages 225–228, 2017.

7 J. E. Goodman and R. Pollack. Allowable sequences and order types in discrete and computational geometry. In Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 103–134. Springer, 1993.

8 B. Grünbaum. *Arrangements and Spreads*, volume 10 of *Regional Conf. Ser. Math.* AMS, 1972 (reprinted 1980).

9 R. J. Kang and T. Müller. Arrangements of pseudocircles and circles. *Discrete & Computational Geometry*, 51:896–925, 2014.

10 F. Levi. Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade. *Ber. Math.-Phys. Kl. sächs. Akad. Wiss. Leipzig*, 78:256–267, 1926.

11 J. Linhart and R. Ortner. An arrangement of pseudocircles not realizable with circles. *Beiträge zur Algebra und Geometrie*, 46:351–356, 2005.

12 N. E. Mnëv. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In *Topology and Geometry — Rohlin Seminar*, volume 1346 of *Lect. Notes in Math.*, pages 527–543. Springer, 1988.

13 P. Suvorov. *Isotopic but not rigidly isotopic plane systems of straight lines*, volume 1346 of *Lect. Notes in Math.*, pages 545–556. Springer, 1988.

On Romeo and Juliet Problems: Minimizing Distance-to-Sight*

Hee-Kap Ahn[1], Eunjin Oh[2], Lena Schlipf[3], Fabian Stehn[4], and Darren Strash[5]

1 **Department of Computer Science and Engineering, POSTECH, South Korea**
heekap@postech.ac.kr
2 **Department of Computer Science and Engineering, POSTECH, South Korea**
jin9082@postech.ac.kr
3 **Theoretische Informatik, FernUniversität in Hagen, Germany**
lena.schlipf@fernuni-hagen.de
4 **Institut für Informatik, Universität Bayreuth**
fabian.stehn@uni-bayreuth.de
5 **Department of Computer Science, Colgate University, US.**
dstrash@cs.colgate.edu

Abstract

We introduce a variant of the watchman route problem, which we call the *quickest pair-visibility* problem. Given two persons standing at points s and t in a simple polygon P with no holes, we want to minimize the distance these persons travel in order to see each other in P. We solve two variants of this problem, one minimizing the longer distance the two persons travel (min-max) and one minimizing the total travel distance (min-sum), optimally in linear time.

1 Introduction

In the watchman route problem, a watchman takes a route to *guard* a given region—that is, any point in the region is visible from at least one point on the route. It is desirable to make the route as short as possible so that the entire area can be guarded as quickly as possible. The problem was first introduced in 1986 by Chin and Ntafos [4] and has been extensively studied in computational geometry [3, 10]. Though the problem is NP-hard for polygons with holes [4, 5, 7], an optimal route can be computed in time $O(n^3 \log n)$ for simple n-gons [6] when the tour must pass through a specified point, and $O(n^4 \log n)$ time otherwise.

In this paper, we study a variant we call the *quickest pair-visibility* problem, which can be stated as follows.

▶ **Problem** (quickest pair-visibility problem). *Given two points s and t in a simple polygon P, compute the minimum distance that s and t must travel in order to see each other in P.*

This problem may sound similar to the shortest path problem between s and t, in which the objective is to compute the shortest path for s to *reach* t. However, they differ even for a simple case: for any two points lying in a convex polygon, the distance in the quickest pair-visibility problem is zero while in the shortest path problem it is their Euclidean distance.

The quickest pair-visibility problem occurs in optimization tasks. For example, mobile robots that use a line-of-sight communication model are required to move to mutually-visible

* This work by Ahn and Oh was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP-2017-0-00905) supervised by the IITP (Institute for Information & communications Technology Promotion.).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

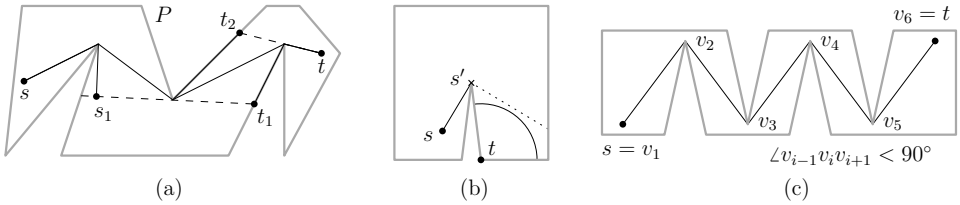


Figure 1 (a) The quickest pair-visibility problem finds two paths $\pi(s, s_1)$ and $\pi(t, t_1)$ such that $\overline{s_1t_1} \subset P$ and $\max\{|\pi(s, s_1)|, |\pi(t, t_1)|\}$ or $|\pi(s, s_1)| + |\pi(t, t_1)|$ is minimized. The quickest visibility problem for query point t finds a shortest $\pi(s, t_2)$ with $\overline{tt_2} \subset P$. (b) **min-max:** Every pair (s', t^*), where t^* is some point within the geodesic disk centered in t with radius $\pi(s, s')$, is an optimal solution to the *min-max* problem. (c) **min-sum:** Every pair (v_i, v_{i+1}) for $1 \leq i < 6$ is an optimal solution to this instance.

positions to establish communication [8]. An optimization task here is to find shortest paths for the robots to meet the visibility requirement for establishing communication among them.

Wynters et al. [12] studied this problem for two agents acting in a polygonal domain in the presence of polygonal obstacles and gave an $O(nm)$-time algorithm for the min-sum variant (where m is the number of edges of the visibility graph of all corners) and an $O(n^3 \log n)$-time algorithm for the min-max variant. A query version of the quickest visibility problem has also been studied [1, 9, 11]. In the query problem, a polygon and a source point lying in the polygon are given, and the goal is to preprocess them and construct a data structure that allows, for a given query point, to find the shortest path taken from the source point to see the query point efficiently. Khosravi and Ghodsi [9] considered the case for a simple n-gon and presented an algorithm to construct a data structure of $O(n^2)$ space so that given a query, it finds the shortest visibility path in $O(\log n)$ time. Later, Arkin et al. [1] improved the result and presented an algorithm for the problem in a polygonal domain. Very recently, Wang [11] presented an improved algorithm for this problem for the case that the number of the holes in the polygon is relatively small. Figure 1(a) illustrates differences in these problems for a simple polygon and two points, s and t, in the polygon.

1.1 Our results

In this paper, we consider two variants of the quickest pair-visibility problem for a simple polygon: either we want to minimize the maximum length of a traveled path (*min-max variant*) or we want to minimize the sum of the lengths of both traveled paths (*min-sum variant*). We give a sweep-line-like approach that "rotates" the lines-of-sight along vertices on the shortest path between the start positions, allowing us to evaluate a linear number of candidate solutions on these lines. Throughout the sweep, we encounter solutions to both variants of the problem. We further show that our technique can be implemented in linear time.

2 Preliminaries

Let P be a simple polygon and ∂P be its boundary. The vertices of P are given in counter-clockwise order along ∂P. We denote the shortest path within P between two points $p, q \in P$ by $\pi(p, q)$ and its length by $|\pi(p, q)|$. We say a point $p \in P$ is visible from another point $q \in P$ (and q is visible from p) if and only if line segment $\overline{pq}$ is completely contained in P.

For two starting points s and t, our task is to compute a pair (s', t') of points such that s'

and t' are visible to each other, where we wish to minimize the lengths of $\pi(s, s')$, and $\pi(t, t')$. In the *min-max* setting, we wish to minimize $\max\{|\pi(s, s')|, |\pi(t, t')|\}$. For the *min-sum* setting, we wish to minimize $|\pi(s, s')| + |\pi(t, t')|$. Note that, for both variants, the optimum is not necessarily unique; see Figure 1(b) and (c).

For our discussion, let (s^*, t^*) be an optimal solution for the instance at hand. Let $V(p)$ denote the visible region for a point p in P, that is, the portion of P that is visible from p. Clearly, $V(p)$ is a *star-shaped* polygon. Moreover, every boundary edge of $V(p)$ is either (part of) an edge of P or a segment $\overline{vq}$ that is contained in P and parallel to $\overline{pv}$, where v is a vertex of P visible from p and q is a point on the boundary of P. We call an edge of the latter type a *window edge* of the visibility region. The structure of $V(p)$ may change as p moves along a path contained in P. It is known that a change to the structure of $V(p)$ occurs if and only if two vertices of P become collinear with p [2].

▶ **Lemma 2.1.** *Unless s and t are visible to each other, the segment $\overline{s^*t^*}$ contains a vertex v of the shortest path $\pi(s, t)$ from s to t.*

It is easy to see by contradiction that $\overline{s^*t^*}$ must contain a vertex v of the boundary of P; using shortest path properties, one can show that v is a vertex of $\pi(s, t)$. The full proof is omitted due to space constraints.

3 Computing All Events for a Sweep-Line-Like Approach

For each vertex v on $\pi(s, t)$ we compute a finite collection of lines through v, each being a configuration at which the combinatorial structure of the shortest paths $\pi(s, s^*)$ and/or $\pi(t, t^*)$ changes. To be more precise, at these lines either the vertices of $\pi(s, s^*)$ or $\pi(t, t^*)$ (except for s^* and t^*) change or the edge of ∂P changes that is intersected by the extension of $\overline{s^*t^*}$. To explain how to compute these lines, we introduce the concept of a *line-of-sight*.

▶ **Definition 3.1** (line-of-sight)**.** We call a segment ℓ a *line-of-sight* if (i) $\ell \subset P$, (ii) both endpoints of ℓ lie on ∂P, and (iii) ℓ is tangent to $\pi(s, t)$ at a vertex $v \in \pi(s, t)$.

We say a segment g is tangent to a path π at a vertex v if $v \in g \cap \pi$ and the local neighborhood of π at all intersections $g \cap \pi$ is on the same side of g. The algorithm we present is in many aspects similar to a sweep-line strategy, except that we do not sweep over the scene in a standard fashion but rotate a *line-of-sight* ℓ in P around the vertices of the shortest path $\pi(s, t) := (s = v_0), v_1, \ldots, v_{k-1}, (t = v_k)$. The process will be initialized with a line-of-sight that contains s and v_1 and is then rotated around v_1 (while remaining tangent to v_1) until it hits v_2, see Figure 2(a). In general, the current line-of-sight is rotated around v_i in a way so that it remains tangent to v_i (it is rotated in the interior of P) until the line-of-sight contains v_i and v_{i+1}, then the process is iterated with v_{i+1} as the new rotation center. The process terminates as soon as the line-of-sight contains v_{k-1} and t.

While performing these rotations around the shortest path vertices, we encounter all combinatorially different lines-of-sight. As for a standard sweep-line approach, we will compute and consider events at which the structure of a solution changes: this is either because the interior vertices of $\pi(s, s^*)$ or $\pi(t, t^*)$ change or because the line-of-sight starts or ends at a different edge of ∂P. These events will be represented by points on ∂P (actually, we introduce the events as vertices on ∂P unless they are already vertices). Between two consecutive lines-of-sight, we compute the local minima of the relevant distances for the variant at hand in constant time and hence encounter all global minima eventually.

There are three event-types to distinguish:

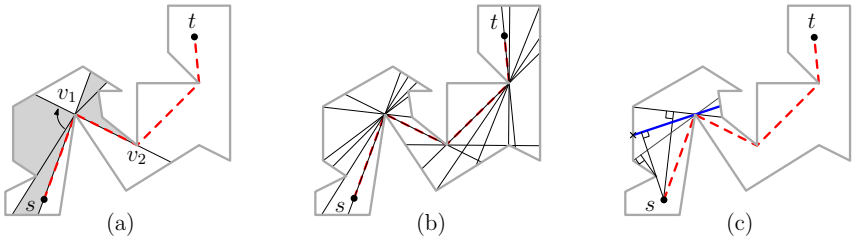


Figure 2 Path- and boundary-events. (a) The first path-event is the line-of-sight through $\overline{sv_1}$. The line-of-sight rotates until it hits the next path-event: the segment through $\overline{v_1v_2}$. (b) All path- and boundary-events: the event-queue is initialized with these events. (c) A bend-event (marked with a cross) occurs between the two boundary-events. The shortest path from s to these segments changes at the bend-event.

1. **Path-Events** are endpoints of lines-of-sight that contain two consecutive vertices of the shortest path $\pi(s,t)$. See Figure 2(a).
2. **Boundary-Events** are endpoints of lines-of-sight that are tangent at a vertex of $\pi(s,t)$ and contain at least one vertex of $P \setminus \pi(s,t)$ (potentially as an endpoint). See Figure 2(b).
3. **Bend-Events** are encountered when, the shortest path of s (or t) to the line-of-sight gains or loses a vertex while rotating the line-of-sight around a vertex v. See Figure 2(c). Note that bend-events can coincide with path- or boundary-events.

We will need to explicitly know both endpoints of the line-of-sight on ∂P at each event and the corresponding vertex of $\pi(s,t)$ on which we rotate.

▶ **Lemma 3.2** (Computing path- and boundary-events). *For a simple polygon P with n vertices and points $s,t \in P$, the queue $\mathcal{Q}$ of all path- and boundary-events of the rotational sweep process, ordered according to the sequence in which the sweeping line-of-sight encounters them, can be initialized in $O(n)$ time.*

Path events coincide with specific vertices of the shortest path map of s (or of t) in P, whereas boundary events are endpoints of specific edges of the shortest path tree of s (or of t) in P. These structures can be constructed and classified in linear time, a full proof is omitted due to space constraints.

Once we initialized the event queue $\mathcal{Q}$, we can now compute and process bend-events as we proceed in our line-of-sight rotations.

▶ **Lemma 3.3.** *All bend-events can be computed in $O(n)$ time, sorted in the order as they appear on the boundary of P.*

Due to space limitations, the proof of Lemma 3.3 is omitted.

4 Algorithm Based on a Sweep-Line-Like Approach

In this section, we present a linear-time algorithm for computing the minimum distance that two points s and t in a simple polygon P travel in order to see each order. We compute all events defined in Section 3 in linear time. The remaining task is to handle the lines-of-sight lying between two consecutive events.

▶ **Lemma 4.1.** *For any two consecutive events, the line-of-sight ℓ lying between them that minimizes the sum of the distances from s and t to ℓ can be found in constant time.*

Proof. Let $\mathcal{L}$ be the set of all lines-of-sights lying between the two consecutive events. Every line-of-sight in $\mathcal{L}$ contains a common vertex v of $\pi(s,t)$. We assume that $\mathcal{L}$ contains no vertical line-of-sight. Otherwise, we consider the set containing all lines-of-sight of $\mathcal{L}$ with positive slopes, and then the set containing all lines-of-sight of $\mathcal{L}$ with negative slopes.

By construction, the second to the last vertex u of $\pi(s,\ell)$ (and $\pi(t,\ell)$) for any $\ell \in \mathcal{L}$ remains the same. We already obtained v and u while computing the events. We will give an algebraic function for the length of $\pi(s,\ell)$ for $\ell \in \mathcal{L}$. An algebraic function for the length of $\pi(t,\ell)$ can be obtained by changing the roles of s and t.

Since the topology of $\pi(s,\ell)$ for every $\ell \in \mathcal{L}$ remains the same, we consider only the length of $\pi(u,\ell)$. Observe that $\pi(u,\ell)$ is a line segment for any $\ell \in \mathcal{L}$, and thus its length is the same as the Euclidean distance between u and ℓ. The length is either the Euclidean distance between u and the line containing ℓ, or the Euclidean distance between u and the endpoint of ℓ closest to u. We show how to handle the first case only because the second case can be handled analogously.

To use this observation, we use $\ell(\alpha)$ to denote the line of slope α passing through v for any $\alpha > 0$. There is an interval I such that $\ell(\alpha)$ contains a line-of-sight in $\mathcal{L}$ if and only if $\alpha \in I$. The Euclidean distance between u and $\ell(\alpha)$ is the same as the distance between u and the line-of-sight contained in $\ell(\alpha)$. Thus, in the following, we consider the distance between u and $\ell(\alpha)$ for every $\alpha \in I$.

Since $\ell(\alpha)$ passes through a common vertex, the line $\ell(\alpha)$ can be represented as the form of $y = \alpha x + f(\alpha)$, where $f(\alpha)$ is a function linear in α. Then, the distance between u and $\ell(\alpha)$ can be represented as the form of $|c_1\alpha + c_2|/\sqrt{\alpha^2+1}$, where c_1 and c_2 are constants depending only on v and u.

Then our problem reduces to the problem of finding a minimum of the function of the form of $(|c_1\alpha + c_2| + |c_1'\alpha + c_2'|)/\sqrt{\alpha^2+1}$ for four constants c_1, c_2, c_1' and c_2', and for all $\alpha \in I$. We can find a minimum in constant time using an elementary analysis. ◀

▶ **Lemma 4.2.** *For any two consecutive events, the line-of-sight ℓ lying between the them that minimizes the maximum of the distances from s and t to ℓ can be found in constant time.*

▶ **Theorem 4.3.** *Given a simple n-gon P with no holes and two points $s, t \in P$, a point-pair (s^*, t^*) such that* i) $\overline{s^*t^*} \subset P$ *and* ii) *either $|\pi(s,s^*)| + \pi(t,t^*)|$ or $\max\{|\pi(s,s^*)|, |\pi(t,t^*)|\}$ is minimized can be computed in $O(n)$ time.*

Proof. Our algorithm first computes all path- and boundary-events as described in Lemma 3.2. The number of events introduced during this phase is bounded by the number of vertices of the shortest path maps, M_s and M_t, respectively, which are $O(n)$. In the next step, it computes the bend-events on ∂P as described in Lemma 3.3, which can be done in $O(n)$ time. Finally, our algorithm traverses the sequence of events. Between any two consecutive events, it computes the respective local optimum in constant time by Lemma 4.1. It maintains the smallest one among the local optima computed so far, and return it once all events are processed. Therefore the running time of the algorithm is $O(n)$.

For the correctness, consider the combinatorial structure of a solution and how it changes. The path-events ensure that all vertices of $\pi(s,t)$ are considered as being the vertex lying on the segment connecting the solution (s^*, t^*). While the line-of-sight rotates around one fixed vertex of $\pi(s,t)$, either the endpoints of line-of-sight sweep over or become tangent to a vertex of ∂P. These are exactly the boundary-events. Or the combinatorial structure of $\pi(s,s^*)$ or $\pi(t,t^*)$ changes as interior vertices of $\pi(s,s^*)$ or $\pi(t,t^*)$ appear or disappear. These happen exactly at bend events. Therefore, our algorithm returns an optimal point-pair. ◀

Acknowledgments

This research was initiated at the 19th Korean Workshop on Computational Geometry in Würzburg, Germany.

References

1 E. M. Arkin, A. Efrat, C. Knauer, J. S. B. Mitchell, V. Polishchuk, G. Rote, L. Schlipf, and T. Talvitie. Shortest path to a segment and quickest visibility queries. *Journal of Computational Geometry*, 7(2):77–100, 2016.

2 B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. Visibility queries and maintenance in simple polygons. *Discrete Comput. Geom.*, 27(4):461–483, 2002.

3 S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete Comput. Geom.*, 22(3):377–402, 1999.

4 W. Chin and S. Ntafos. Optimal watchman routes. In *Proceedings of the 2nd ACM Symposium on Computational Geometry*, pages 24–33, 1986.

5 W. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, 28(1):39–44, 1988.

6 M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th ACM Symposium on Theory of Computing*, pages 473–482, 2003.

7 A. Dumitrescu and C. D. Tóth. Watchman tours for polygons with holes. *Computational Geometry*, 45(7):326–333, 2012.

8 A. Ganguli, J. Cortes, and F. Bullo. Visibility-based multi-agent deployment in orthogonal environments. In *Proc. Am. Control Conf.*, pages 3426–3431, 2007.

9 R. Khosravi and M. Ghodsi. The fastest way to view a query point in simple polygons. In *Proc. European Workshop on Computational Geometry*, pages 187–190, 2005.

10 J. S. B. Mitchell. Approximating watchman routes. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 844–855, 2013.

11 H. Wang. Quickest visibility queries in polygonal domains. In *Proceedings of the 33rd International Symposium on Computational Geometry*, volume 77, pages 61:1–61:16, 2017.

12 E. L. Wynters and J. S. B. Mitchell. Shortest paths for a two-robot rendez-vous. In *Proc. 5th Canadian Conference on Computational Geometry*, pages 216–221, 1993.

The Topology of Skeletons and Offsets

Stefan Huber[1]

1 **B&R Industrial Automation**
stefan.huber@br-automation.com

Abstract

Given a polygonal shape with holes, we investigate the topology of two types of skeletons (straight skeleton, Voronoi diagram) and the evolution of the inward offsets they induce. It is shown that both skeletons are homotopy equivalent to the shape and an $O(n \log n)$ algorithm to compute the persistent homology of the filtration of the inset polygons w.r.t. to their reversed offsetting process is given. We conclude with a brief discussion on possible applications.

1 Introduction

The straight skeleton and the Voronoi diagram of a polygonal shape capture certain topological and geometrical information. For instance, the maximum inscribed circle of the shape has its center at a vertex of of Voronoi diagram. In terms of homotopy both skeletons encode *the* topology of the shape, but their geometry is different. The different geometry manifests in different offset curves: Mitered offsets for straight skeletons and Minkowski offsets for Voronoi diagrams. The evolution of offset curves again tells something about the topology of the shape. The mathematical tool to investigate this observation is persistent homology.

Lieutier [8] showed that the medial axis of an open bounded set in $\mathbb{R}^d$ is homotopy equivalent to its medial axis by an involved proof not based on constructing a deformation retraction. Further related work concerns the homotopy of the medial axis, its stability, and its relation to the Voronoi diagram of a point set. Halperin et al. [4] investigated the outer (Minkowski) offset filtration of convex polyhedra in two and three dimensions, i.e., they generalize from (alpha filtrations of) point sets to sets of disjoint convex polyhedra and presented an $O(n \log n)$ algorithm for the persistent homology.

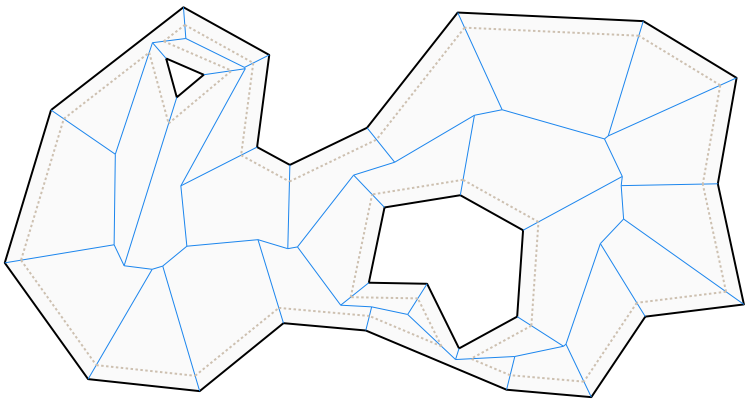


Figure 1 The straight skeleton $S(P)$ in blue of a polygon with holes, P, in black. The wavefront (a mitered offset curve) is shown as dotted lines.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2 Topology of skeletons

Let P denote a polygon with holes in the plane, i.e., $\operatorname{bd} P$ forms a set of disjoint closed polygonal curves. The straight skeleton $S(P)$ of P is defined by a wavefront propagation process where the edges of P move inwards at unit speed. Two kind of structural changes occur to the wavefront: (i) edges may collapse and vanish and (ii) reflex vertices may hit another part of the wavefront and split it into parts.[1] The line structure that is traced out by the wavefront vertices was introduced in [1] as the straight skeleton $S(P)$ of P, see Fig. 1. We call the area swept out by one edge f of P the straight-skeleton cell $C_S(f)$ of f.

The Voronoi diagram $V(P)$ of P is defined by a nearest-neighbor cell decomposition of P by the faces of P, i.e., its vertices and edges. We follow [5] by defining the cone of influence $I(f)$ of a vertex f to be $\mathbb{R}^2$ and of an edge f to be the orthogonal strip spanned by f. Then the Voronoi cell $C_V(f)$ is defined as the set of points in $I(f)$ at least as close to f than to any other face of P. We define the Voronoi diagram $V(P)$ of P as the line structure formed by the boundaries of the Voronoi cells, restricted to P, see Fig. 2.

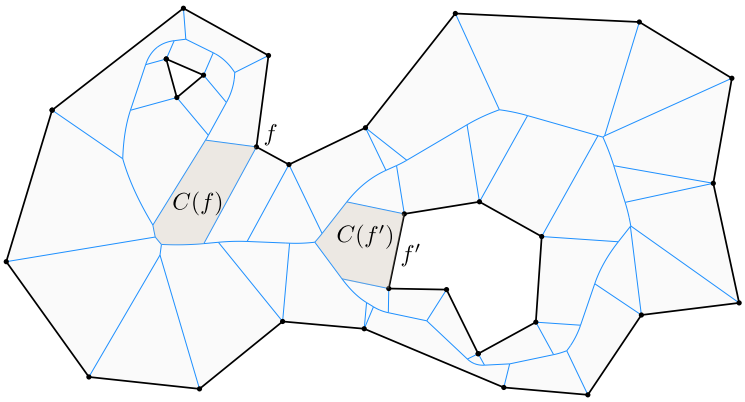


Figure 2 The Voronoi diagram $V(P)$ in blue of a polygon with holes, P, in black. Two cells $C_V(f)$ and $C_V(f')$ of the faces f and f' shaded in gray.

Two remarks on the above definition: First, for our purpose we would like to emphasize the notion of a Voronoi diagram *of a polygon* in analogy to $S(P)$ and in contrast to the typical notion of the Voronoi diagram of a collection of sites (which could form a polygon). Secondly, the two edges of $V(P)$ emanating at each reflex vertex are considered to be *topologically* disjoint, i.e., the two distinct endpoints only geometrically overlap. Furthermore, we split conic Voronoi edges at the apex, including those between two vertices of P, see [7]. This is (i) algorithmically handy, e.g., when computing offset curves, and (ii) turns out to be natural from a topological perspective.

Both, $S(P)$ and $V(P)$, capture geometrical and topological features of the underlying shape P. For instance, they form a tree for simple polygons. Moreover, for each hole that we punch into P both get a new (generator) cycle (in a group of cycles). That is, in terms of homotopy theory, they both capture *the* topology of the shape:

▶ **Theorem 2.1.** *Let P denote a polygon with holes in the plane. The following homotopy equivalences hold:*

$$P \simeq S(P) \simeq V(P).$$

[1] See [7] for a survey on straight skeletons including a taxonomy on the different wavefront events.

It suffices to show that $S(P)$ and $V(P)$ are each deformation retracts of P.

▶ **Lemma 2.2.** *$S(P)$ is a deformation retract of P.*

Proof. Consider the cell decomposition of P induced by $S(P)$. The cell $C_S(f)$ of an edge f is a topological disk [7]. Let us denote by $S(P,f) = S(P) \cap C_S(f)$ the boundary of $C_S(f)$ without f, which is a connected part of $\operatorname{bd} C_S(f)$. The topological disk $C_S(f)$ can be trivially deformation retracted to $S(P,f)$. We can even require that the deformation retraction stays constant on $S(P,f)$. This allows us to plug together the per-cell deformation retractions to a deformation retraction of $P = \bigcup_f C(f)$ to $\bigcup_f S(P,f) = S(P)$. ◀

Note that the above proof also applies to straight-skeletons of positively weighted straight skeletons. However, in the presence of negative weights Thm. 2.1 fails as $S(P)$ of a simple polygon P may have cycles as shown in [2].

▶ **Lemma 2.3.** *$V(P)$ is a deformation retract of P.*

We could use the more general result of Lieutier [8] for the medial axis, augment it with certain line segments to obtain $V(P)$ and argue that the homotopy type did not change. However, the simple proof scheme of Lem. 2.2 basically applies here, too. Moreover, Voronoi cells of circular arcs meet the above topological requirements as well [6], and hence Lem. 2.3 also applies to shapes P bounded by straight-line segments and circular arcs.

There is only a technicality at reflex vertices (for both approaches), where we remind the reader that the two emanating Voronoi edges are considered topologically disjoint. The topological space $V(P)$ could be obtained by glueing together Voronoi edges, but we do not glue at reflex vertices of P. Put in different words, let us consider P' as the Minkowski-difference[2] $P \ominus B_\epsilon$ of P by an $\epsilon > 0$, where B_ϵ denotes the o-centered ball of radius ϵ. Then $V(P') = V(P) \cap P'$, i.e., $V(P)$ is $V(P')$ with little line segments attached at the tips of $V(P')$. The shape P' is structurally the same as P, only the reflex vertices of P are replaced by tiny circular arcs of radius ϵ. We consider $V(P)$ and $V(P')$ to be topologically identical, only the tips of $V(P')$ are geometrically perturbed. In particular, we consider $V(P,f) = V(P) \cap C_V(f)$ being a topological line instead of a circle at reflex vertices.

▶ **Corollary 2.4.** *P, $S(P)$, and $V(P)$ are homologous and, by the theorem of Euler-Poincaré, have the same Euler characteristics.*

3 Persistence of offset curves

3.1 Mitered and Minkowski offsets

A skeleton and its offset curves are dual in the following sense: We can easily compute offset curves from the skeleton and, vice versa, the skeleton can be obtained from the evolution of offset curves. For the latter direction this is the original definition of straight skeletons, where the wavefront propagation is the evolution of the offset curves. The definition of Voronoi diagrams based the evolution of the offset curves is related to the so-called grassfire model.

For the former direction, the computation of mitered offset curves by means of straight skeletons resp. Minkowski offset curves by means of Voronoi diagrams are one of many

[2] For sets A, B in a vector space let $A \oplus B = \{x + y \colon x \in A, y \in B\}$ denote the Minkowski-sum and let $A \ominus B = \{x \colon \{x\} \oplus B \subseteq A\} = (A^c \oplus (-B))^c$ denote the Minkowski-difference.

applications of skeletons, e.g. in GIS (buffer zone computation) or CAD/CAM (tool-path planing in NC-machining), cf. [5, 7].

Let us denote by $Q_V(r)$ the polygon P inset by radius r according Minkowski offsetting, i.e., $Q_V(r) = P \ominus B_r$. Similarly, denote by $Q_S(r)$ the polygon P inset by radius r according to mitered offsetting. The so-called roof model projects the evolution of offset curves in three-space with the third dimension being time, see Fig. 3. We denote by $R_V(P) = \bigcup_{r \geq 0} \operatorname{bd} Q_V(r) \times \{r\}$ and likewise for $R_S(P)$. In the following we write $Q_*(r)$ resp. $R_*(P)$ when we refer to both $Q_V(r)$ and $Q_S(r)$ resp. $R_V(P)$ and $R_S(P)$. Aichholzer and Aurenhammer [1] showed the following property for $R_S(P)$, which is also true for $R_V(P)$:

▶ **Lemma 3.1.** *$R_*(P)$ does not possess local minima, except all points of $\operatorname{bd} P \times \{0\}$.*

Proof. Assume $R_*(P)$ would possess a local minimum at $q \in \operatorname{int} P$ at level $t > 0$. Then $P \setminus Q_*(t+\epsilon)$ possesses an arbitrarily small component around q for small enough $\epsilon > 0$. This basically means that offset curves pop up without being emanated from $\operatorname{bd} P$. ◀

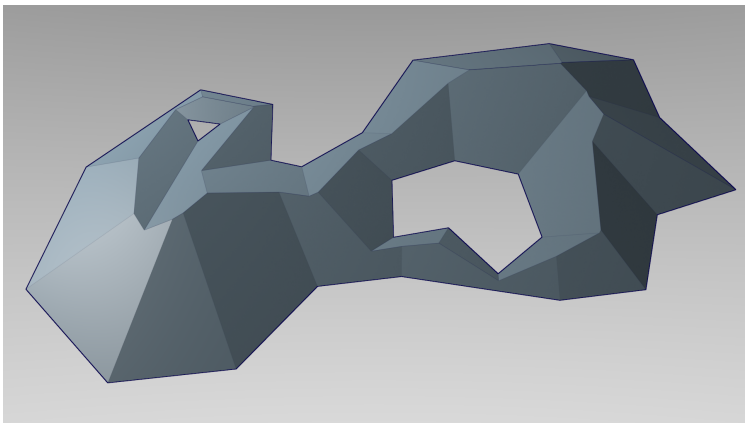


Figure 3 The straight-skeleton roof model $R_S(P)$ of P. The vertices, crests and ridges of $R_S(P)$ projected onto $\mathbb{R}^2 \times \{0\}$ give $S(P)$ again. Offset curves are lifted to isolines on $R_S(P)$.

3.2 Computing persistent homology of offset curve filtrations

Persistent homology is a mathematical framework that investigates the evolution of homology groups in a so-called filtration of topological spaces. In the following we apply this framework to a growing sequence of nested sets, where the growth is given by the offset curves with a decreasing offset radius. This gives insight into the topology of the underlying shape that goes beyond the homotopy type of P because the topological changes in the offset curves pull in geometric information from the offsetting process itself.

Let us consider r to decrease from a large enough r_0 to 0, while $Q_*(r)$ grows from the empty set to P. We ask for the persistent homology groups (over $\mathbb{Z}_2$) of this *offset filtration* of P. Using the roof model $R_*(P)$ we can apply the water shed picture [3] here: Assume the sea has level r_0 and then continuously lowers to level 0. At local maxima of $R_*(P)$ islands pop up (0-dimensional homology classes are born), at certain other levels islands merge with others (0-dimensional homology classes die) or atolls are formed (1-dimensional homology classes are born). However, from Lem. 3.1 follows this:

▶ **Lemma 3.2.** *In an offset filtration 1-dimensional homology classes never die.*

3.2.1 Direct approach on a simplicial complex

A straightforward approach to compute persistent homology could be to apply the boundary matrix algorithm [3]. To do so, we switch to the setting of a filtration on a simplicial complex. First, we consider a finite filtration: Note that the topological changes of $Q_*(r)$ only occur at levels of $R_*(P)$ where the isoline touches a roof vertex. Let us denote by $r_1 > r_2 > \cdots > r_k = 0$ the sequence of levels at which the vertices of $R_*(P)$ sit, which gives us the nested sets $Q_*(r_1) \subset \cdots \subset Q_*(r_k)$. (We may add a level $r_0 > r_1$ in order to start with the empty set $Q_*(r_0)$.) Next we construct a simplicial complex $\mathcal{C}$ that covers P by (i) adding the offset curves $\operatorname{bd} Q_*(r_1), \ldots, \operatorname{bd} Q_*(r_k)$ to the skeleton and (ii) triangulating the onion layers $Q_*(r_{i+1}) \setminus \operatorname{int} Q_*(r_i)$ for $1 \leq i \leq k-1$. Note that in step (i), we split skeleton edges at the intersection points with the offset curves and in step (ii), we only need a topological triangulation, i.e., edges do not need to be straight. Then we define a simplicial function $\mathcal{C} \to [0, \infty)$ by assigning each simplex of $\mathcal{C}$ the level of its lowest point in $R_*(P)$. We take the super-level set filtration according to this simplicial function, which corresponds to the offset filtration initially presented, i.e., it contains triangulations of all $Q_*(r_i)$ as subcomplexes.

The boundary matrix reduction runs in $O(m^3)$ time where $m \in O(kn) \subseteq O(n^2)$ is the size of $\mathcal{C}$. The construction of $\mathcal{C}$ involves the computation of k offset curves, each taking $O(n)$ time after the skeleton has been computed, and the triangulation in $O(m \log m)$ time.

3.2.2 A skeleton-based algorithm

Note that the Voronoi diagram of $Q_V(r)$ is $V(P) \cap Q_V(r)$ and similarly $S(Q_S(r)) = S(P) \cap Q_S(r)$ for straight skeletons. So $Q_V(r)$ is homotopy equivalent to $V(P) \cap Q_V(r)$ and therefore homologous. That is, instead of considering the growing sets $Q_V(r_1) \subset \cdots \subset Q_V(r_k)$ we can consider the growing subsets $Q_V(r_1) \cap V(P) \subset \cdots \subset Q_V(r_k) \cap V(P)$ of the Voronoi diagram and likewise for the straight skeleton. So it suffices to track the birth and death of components and cycles in the growing graph structure of the skeleton.

By Lem. 3.2 we can exclude the death of cycles from our considerations. So we sort the vertices of the skeleton by decreasing level in the roof model and keep adding vertex by vertex in the growing graph structure. For each new vertex v we have the following cases:

1. No neighbor of v was inserted already. Then v is a peak and a new component is born.
2. The neighbors $u_1, \ldots, u_d$ were already inserted. For every connected component that is involved with c vertices in $\{u_1, \ldots, u_d\}$ we get $c-1$ new cycles closed at v. All involved components are merged with the oldest component and then v joins this component, too.

After the skeleton of P has been computed and the vertices were sorted in $O(n \log n)$ time, where n is the number of vertices of P, one can compute the birth and death of the homology classes in $O(n\,\alpha(n))$ time by means of a union-find data structure [3]. (There are $O(n)$ find resp. union operations in case 2.) If one is interested in the homology classes itself each can be dumped in $O(n \log n)$ time by a simple graph traversal. For instance the cycle that is born at level 0.109 in Fig. 4 can be obtained by a depth-first search along one emanating edge of the vertex v at level 0.109, restricted to vertices inserted so far, until v is reached again. (The traversal stays within the component in which the cycle is closed.)

4 Conclusion

Computational topology has prominent applications in topological data analysis. We believe that also classical problems in computational geometry profit from methods of computational topology.

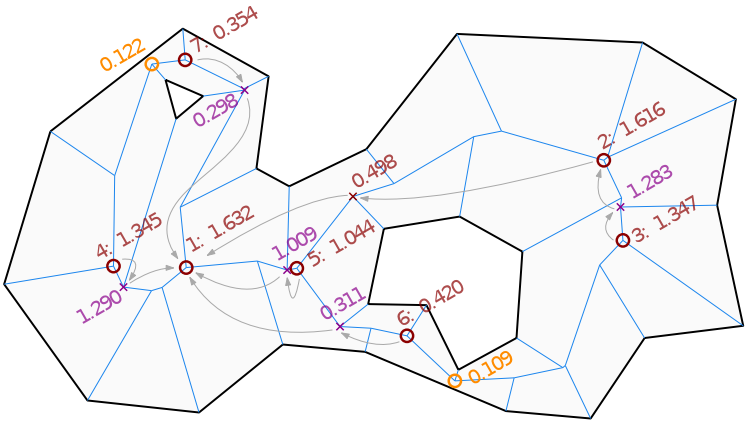


Figure 4 Birth and death of homology classes in the mitered-offset filtration by inserting vertices at given levels. Red circles are of case 1. (Peak 1 is on level 1.632.) Orange (birth of cycle) and violet (death of component) vertices are of case 2. Unlabeled vertices are trivial instances of case 2. The grey arrows tell the merge direction. Peaks in decreasing persistence: 1, 2, 6, 3, 7, 4.

Take for instance the maximum inscribed circle of P whose center is known to be a vertex of $V(P)$ with highest distance to its defining faces, i.e., the highest peak in $R_V(P)$. In other words, the maximum inscribed circle corresponds to the 0-dimensional homology class of highest persistence, where the persistence of a homology class is defined by the level difference of birth and death. We can quantify all peaks by its persistence and obtain a notion of "significance" of a locally maximum inscribed circle. This could again be useful for shape decomposition algorithms, e.g. for motion planing in NC machining. In Fig. 4 the peaks 1 and 2 have a significant persistence above 1.1, while the other peaks possess a comparable small persistence below 0.1. Those two peaks represent the "main parts" of P in this sense.

References

1 O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In A.M. Samoilenko, editor, *Voronoi's Impact on Modern Science, Book 2*, pages 7–21. Inst. of Math. of the National Academy of Sciences of Ukraine, Kiev, Ukraine, 1998.

2 T. Biedl, M. Held, S. Huber, D. Kaaser, and P. Palfrader. Weighted straight skeletons in the plane. *Comp. Geom. Theory & Appl.*, 48(2):120–133, February 2015.

3 H. Edelsbrunner and J. Harer. *Computational Topology – An Introduction.* American Mathematical Society, 2010. ISBN 978-0-8218-4925-5.

4 D. Halperin, M. Kerber, and D. Shaharabani. The offset filtration of convex objects. In *Proc. 23rd (ESA '15)*, pages 705–716, 2015.

5 M. Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *Lecture Notes Comput. Sci.* Springer-Verlag, June 1991. ISBN 3-540-54103-9.

6 M. Held and S. Huber. Topology-oriented incremental computation of Voronoi diagrams of circular arcs and straight-line segments. *Comp. Aided Design*, 41(5):327–338, May 2009.

7 S. Huber. *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice.* Shaker Verlag, April 2012. ISBN 978-3-8440-0938-5.

8 A. Lieutier. Any open bounded subset of $\mathbb{R}^n$ has the same homotopy type than its medial axis. *Comp. Aided Design*, 36(11):1029–1046, 2004.

Balanced Dynamic Loading and Unloading*

Sándor P. Fekete[1], Sven von Höveling[1], Joseph S. B. Mitchell†[2], Christian Rieck[1], Christian Scheffer[1], Arne Schmidt[1], and James R. Zuber[2]

1 **Department of Computer Science, TU Braunschweig, Germany.**
{s.fekete, v.sven, c.rieck, c.scheffer, arne.schmidt}@tu-bs.de

2 **Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794, USA.**
joseph.mitchell@stonybrook.edu, zuber139@gmail.com

Abstract

We consider dynamic loading and unloading problems for heavy geometric objects. The challenge is to maintain balanced configurations at all times: minimize the maximal motion of the overall center of gravity. While this problem has been studied from an algorithmic point of view, previous work only focuses on balancing the *final* center of gravity; we give a variety of results for computing schemes that minimize the maximal motion of the center of gravity during the entire process.

In particular, we consider the one-dimensional case and distinguish between *loading* and *unloading*. In the unloading variant, the positions of the intervals are given, and we search for an optimal unloading order of the intervals. We prove that the unloading variant is NP-complete and give a 2.7-approximation algorithm. In the loading variant, we have to compute both the positions of the intervals and their loading order. We give optimal approaches for several variants that model different loading scenarios that may arise, e.g., in the loading of a container ship.

1 Introduction

Packing a set of objects is a classic challenge that has been studied extensively, from a variety of perspectives. The basic question is: how can the objects be arranged to fit into a container? Packing problems are important for many practical applications, such as loading items into a storage space, or containers onto a ship. They are also closely related to scheduling and sequencing, which may include additional temporal considerations. Packing and scheduling are closely intertwined in *loading* and *unloading* problems, where the challenge is not just to compute a good *final* configuration, but also to *dynamically build* this configuration, such that intermediate states are both achievable and stable, e.g., when loading and unloading container ships, for which maintaining *balance* throughout the process is crucial.

In this paper, we consider algorithmic problems of balanced loading and unloading. For unloading, this means planning an optimal sequence for removing a given set of objects, one at a time; for loading, this requires planning both the position and order of the objects.

The practical constraints of loading and unloading motivate a spectrum of relevant scenarios. As ships are symmetric around their main axis, we focus on one-dimensional settings, in which the objects correspond to intervals. Containers may be of uniform size, but stackable up to a certain limited height; because sliding objects on a moving ship are major safety hazards, stability considerations may prohibit gaps between containers.

* A full version of the paper is available at [2]. An extended abstract will appear in the 13th Latin American Theoretical INformatics Symposium (LATIN 2018), April 16–19, 2018.

† Work of this author is partially supported by the National Science Foundation (CCF-1526406).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

1.1 Related Work

Previous work on cargo loading covers a wide range of specific aspects, constraints and objectives. The general Cargo Loading Problem (CLP) asks for an optimal packing of (possibly heterogeneous) rectangular boxes into a given bin, equivalent to the Cutting Stock Problem [4]. Most of the proposed methods are heuristics based on (mixed) integer programming and have been studied both for heterogeneous and homogeneous items.

Amiouny et al. [1] consider the problem of packing a set of one-dimensional boxes of different weights and different lengths into a flat bin (so they are not allowed to stack these boxes), in such a way that after placing the last box, the center of gravity is as close as possible to a fixed target point. They prove strong NP-completeness by a reduction from 3-Partition and give a heuristic with a guaranteed accuracy within $\ell_{max}/2$ of a given target point, where ℓ_{max} is the largest box length.

Gehring et al. [3] consider the general CLP, in which (rectangular) items may be stacked and placed in any possible position. Mongeau and Bes [5] consider a similar variant in which the objective is to maximize the loaded weight. In addition, there may be other paramaters, e.g., each item may have a different priority [8].

While all of this work is related to our problem, it differs in not requiring the center of gravity to be under control for each step of the loading or unloading process. A problem in which a constraint is imposed at each step of a process is Compact Vector Summation (CVS), which asks for a permutation of a set of k-dimensional vectors in order to control their sum, keeping each partial sum within a bounded k-dimensional ball. See Sevastianov [6, 7] for a summary of results in CVS and its application in job scheduling.

2 Preliminaries

An *item* is a unit interval $I := [m - \frac{1}{2}, m + \frac{1}{2}]$ with midpoint m. A set $\{I_1, \ldots, I_n\}$ of n items with midpoints $m_1, \ldots, m_n$ is *valid* if $m_i = m_j$ or $|m_i - m_j| \geq 1$ holds for all $i, j = 1, \ldots, n$. The *center of gravity* $C(I_1, \ldots, I_n)$ of a valid set $\{I_1, \ldots, I_n\}$ of items is defined as $\frac{1}{n}\sum_{i=1}^{n} m_i$.

Given a valid set $\{I_1, \ldots, I_n\}$ of items, we seek orderings in which each item I_j is removed or placed such that the maximal deviation for all points in time $j = 1, \ldots, n$ is minimized. Formally, for $j = 1, \ldots, n$ and a permutation $\pi : j \mapsto \pi_j$, let $C_j := C\left(I_{\pi_j}, \ldots, I_{\pi_n}\right)$.

The Unloading Problem (Unload) seeks to minimize the maximal deviation during an unloading process of $I_1, \ldots, I_n$. In particular, given an input set $\{I_1, \ldots, I_n\}$ of items, we seek a permutation π such that $\max_{i,j=1,\ldots,n} |C_i - C_j|$ is minimized.

In the Loading Problem (Load) we relax the constraint that the positions of the considered items are part of the input. In particular, we seek an ordering and a set of midpoints for the containers such that the containers are disjoint and the maximal deviation for all points in time of the loading process is minimized; see Section 4 for a formal definition.

3 Unloading

We show that the problem Unload is NP-complete and give a polynomial-time 2.7-approximation algorithm for Unload. We first show that there is a polynomial-time reduction from the discrete version of Unload, the Discrete Unloading Problem (dUnload), to Unload; this leads to a proof that Unload is NP-complete, followed by a 2.7-approximation algorithm for Unload.

In the Discrete Unloading Problem (dUnload), we consider a discrete set $X := \{x_1, \ldots, x_n\}$ of points. The center of gravity $C(X)$ of X is defined as $\frac{1}{n}\sum_{i=1}^{n} x_i$. For

$j = 1, \ldots, n$ and a permutation $\pi : j \mapsto \pi_j$, let $C_j = C\left(x_{\pi_1,\ldots,x_{\pi_j}}\right)$. Again, we seek a permutation such that $\max_{i,j=1,\ldots,n} |C_i - C_j|$ is minimized.

▶ **Lemma 3.1.** UNLOAD *and* DUNLOAD *are polynomial-time equivalent.*

3.1 NP-Completeness of the Discrete Case

We can establish NP-completeness of the discrete problem DUNLOAD.

▶ **Theorem 3.2.** DUNLOAD *is* NP*-complete.*

The proof of Theorem 3.3 is based on a reduction of 3-PARTITION and omitted for lack of space, just like any other formal proof; see the full version of this paper [2]. Because of the polynomial-time equivalence of DUNLOAD and UNLOAD, we conclude the following.

▶ **Corollary 3.3.** UNLOAD *is* NP*-complete.*

3.2 Lower Bounds and an Approximation Algorithm

When unloading a set of items, their positions are fixed, so (after reversing time) unloading is equivalent to a loading problem with predetermined positions. For easier and uniform notation throughout the paper, we use this latter description.

In order to develop and prove an approximation algorithm for DUNLOAD, we begin by examining lower bounds on the span, $R - L$, of a minimal interval, $[L, R]$, containing the centers of gravity at all stages in an optimal solution.

Without loss of generality, we assume that the input points x_i sum to 0 (i.e., $\sum_i x_i = 0$), so that the center of gravity, C_n, of all n input points is at the origin. We let $R = \max_i C_i$ and $L = \min_i C_i$. Our first simple lemma leads to a first (fairly weak) bound on the span.

▶ **Lemma 3.4.** *Let $(x_1, x_2, x_3, \ldots)$ be any sequence of real numbers, with $\sum_i x_i = 0$. Let $C_j = (\sum_{i=1}^{j} x_i)/j$ be the center of gravity of the first j numbers, and let $R = \max_i C_i$ and $L = \min_i C_i$. Then, $|R - L| \geq \frac{|x_i|}{i}$, for all $i = 1, 2, \ldots$.*

▶ **Corollary 3.5.** *For any valid solution to* DUNLOAD*, the minimal interval $[L, R]$ containing the center of gravity at every stage must have length $|R - L| \geq \frac{|u_i|}{i}$ where u_i is the input point with the i-th smallest magnitude.*

We note that the naive lower bound given by Corollary 3.5 can be far from tight: Consider the sequence $1, 2, 3, 4, 5, 6, 7, -7, -7, -7, -7$. In the optimal order, the first -7 is placed fourth, after $2, 1, 3$. The optimal third and fourth centers, $\{2, -\frac{1}{4}\}$ are the largest magnitude positive and negative centers seen, and show a span 2.25 times greater than the naive bound of 1. By placing the first -7 in the third position, $R \geq \frac{3}{2}$, and $L \leq -\frac{4}{3}$. By placing it fifth, $R \geq \frac{5}{2}$. Our observation is that failing to place our first -7 if the cumulative sum is > 7 would needlessly increase the span.

This generalizes to the sequence $(x_1 = 1, x_2 = 2, \ldots, x_k = k, x_{k+1} = -k, x_{k+2} = -k, \ldots, x_N)$, with an appropriate x_N to make $\sum x_i = 0$. If we place positive weights in increasing order until, the current center of mass $C_j \geq \frac{k}{j}$, placing $-k$ instead of a positive point at position j would decrease the center of gravity well below $\frac{k}{j}$. The first negative point should be placed when $\min_j \frac{j^2-j}{2} \geq k$, which is when $j \approx \sqrt{2k}$. In this example, our optimal center of gravity span is at least $\frac{k}{j} \approx \sqrt{\frac{k}{2}}$, not the 1 from the naive bound of Corollary 3.5.

We now describe our heuristic, $\mathcal{H}$, which leads to a provable approximation algorithm. It is convenient to relabel and reindex the input points as follows. Let $(P_1, P_2, \ldots)$ denote the

positive input points, ordered (and indexed) by increasing value. Similarly, let $(N_1, N_2, \ldots)$ denote the negative input points, orders (and indexed) by increasing magnitude $|N_i|$ (i.e., ordered by decreasing value).

The heuristic $\mathcal{H}$ orders the input points as follows. The first point is simply the one closest to the origin (i.e., of smallest absolute value). Then, at each step of the algorithm, we select the next point in the order by examining three numbers: the partial sum, S, of all points placed in the sequence so far, the smallest magnitude point, α, not yet placed that has the same sign as S, and the smallest magnitude point, β, not yet placed that has the opposite sign of S. If $S+\alpha+\beta$ is of the same sign as S, then we place β next in the sequence; otherwise, if $S+\alpha+\beta$ has the opposite sign as S, then we place α next in the sequence. The intuition is that we seek to avoid the partial sum from drifting in one direction; we switch to the opposite sign sequence of input points in order to control the drift, when it becomes expedient to do so, measured by comparing the sign of S with the sign of $S+\alpha+\beta$, where α and β are the smallest magnitude points available in each of the two directions. We call the resulting ordering the $\mathcal{H}$-permutation. The $\mathcal{H}$-permutation puts the j-th largest positive point, P_j, in position π_j^+ in the order, and puts the j-th largest in magnitude negative point, N_j, in position π_j^- in the order, where

$$\pi_j^+ = j + \max_k\{k \ : \ \sum_{i=1}^{k} |N_i| \leq \sum_{l=1}^{j} P_l\} \quad \text{and} \quad \pi_j^- = j + \max_k\{k \ : \ \sum_{i=1}^{k} P_i < \sum_{l=1}^{j} |N_l|\}.$$

We obtain an improved lower bound based on our heuristic, $\mathcal{H}$, which orders the input points according to the $\mathcal{H}$-permutation.

▶ **Lemma 3.6.** *A lower bound on the optimal span of* dUnload *is given by* $|R-L| \geq \frac{P_i}{\pi_i^+}$ *and* $|R-L| \geq \frac{|N_i|}{\pi_i^-}$.

▶ **Claim 1.** *For any input set to the discrete unloading problem, where s_i is a member of S, the set of all terms with the same sign sorted by magnitude, a permutation π that minimizes the maximum value of the ratio $\frac{|s_i|}{\pi_i}$ must satisfy $\pi_k < \pi_i$, for all $k < i$.*

▶ **Theorem 3.7.** *The $\mathcal{H}$-permutation minimizes the maximum (over i) value of the ratio $\frac{|x_i|}{\pi_i}$, and thus yields a lower bound on $|R-L|$.*

For the worst-case ratio, we get the following.

▶ **Theorem 3.8.** *The $\mathcal{H}$ heuristic yields an ordering having span $R-L$ at most 2.7 times larger than the $\mathcal{H}$-lower bound.*

▶ **Corollary 3.9.** *There is a polynomial-time 2.7-approximation algorithm for* Unload.

4 Loading

We consider loading problems, where the positions of the objects are part of the optimization. Therefor some additional definitions are necessary:

An *item* is given by a real number ℓ. By assigning a *position* $m \in \mathbb{R}$ to an item, we obtain an interval I with length ℓ and midpoint m. For $n \geq 1$, we consider a set $\{\ell_1, \ldots, \ell_n\}$ of n items and assume $\ell_1 \geq \cdots \geq \ell_n$. Furthermore, $\{\ell_1, \ldots, \ell_n\}$ is *uniform* if $\ell := \ell_1 = \ldots = \ell_n$.

A *state* is a set $\{(I_1, h_1), \ldots, (I_n, h_n)\}$ of pairs, each one consisting of an interval I_j and an integer $h_j \geq 1$, the *layer* in which I_j lies. A state satisfies the following: (1) Two different

intervals that lie in the same layer do not overlap and (2) for $j = 2, \ldots, n$, an interval in layer j is a subset of the union of the intervals in layer $j-1$.

A state $\{(I_1, h_1), \ldots, (I_n, h_n)\}$ is *plane* if all intervals lie in the first layer.

To simplify the following notations, we let m_j denote the midpoint of the interval I_j, for $j = \{1, \ldots, n\}$. The *center of gravity* $C(s)$ of a state $s = \{(I_1, h_1), \ldots, (I_n, h_n)\}$ is defined as $\frac{1}{M}\sum_{j=1}^{n} \ell_j m_j$, where M is defined as $\sum_{j=1}^{n} \ell_j$.

A *placement* p of n items $\ell_1, \ldots, \ell_n$ is a sequence $\langle I_{\pi_1}, \ldots, I_{\pi_n} \rangle$ such that (1) there is a permutation π with $\ell_i = |I_{\pi_i}|$ for all $i \in \{1, \ldots, n\}$ and (2) $\{(I_{\pi_1}, h_{\pi_1}), \ldots, (I_{\pi_j}, h_{\pi_j})\}$ is a state, the *j-th state* s_j, for each $j = 1, \ldots, n$. The 0-th state s_0 is defined as $\emptyset$ and its center of gravity $C(s_0)$ is defined as 0.

▶ **Definition 4.1.** The LOADING PROBLEM (LOAD) is defined as follows: Given a set of n items, determine a placement p such that the $n+1$ centers of gravity of the $n+1$ states of p lie close to 0. In particular, the *deviation* $\Delta(p)$ of a placement p is defined as $\max_{j=0,\ldots,n} |C(s_j)|$. We seek a placement of S with minimal deviation among all possible placements for S.

We say that *stacking is not allowed* if we require that all intervals are placed in layer 1. Otherwise, we say that *stacking is allowed*. For a given integer $\mu \geq 1$ we say that μ is the *maximum stackable height* if we require that all used layers are no larger than μ.

Note that in the loading case, minimizing the deviation is equivalent to minimizing the diameter, i.e., minimizing the maximal distance between the smallest and largest extent of the centers.

4.1 Optimally Loading Unit Items With Stacking

Now we consider the case where you have given a set of unit items which has to be loaded and you are allowed to stack these items up the a certain height. A simple and straightforward strategy for this scenario is to build a stack of maximum height first (call this stack $\mathcal{S}_0$) and place items as close as possible to $\mathcal{S}_0$ on alternating sides afterwards. By selecting the position of $\mathcal{S}_0$ carefully, this strategy guarantees the following:

▶ **Theorem 4.2.** *There is a polynomial-time algorithm for loading a set of unit items so that the deviation of the center of gravity is in* $[0, \frac{1}{1+\mu}]$*, where* μ *is the maximum stackable height.*

Furthermore it can be shown by a contradiction argument that there is no strategy that can guarantee a smaller deviation of the center of gravity than the strategy described above.

▶ **Theorem 4.3.** *The strategy given in Theorem 4.2 is optimal for* $n > \mu$*, i.e., there is no strategy such that the center of gravity deviates in* $[0, \frac{1}{1+\mu})$.

Combining Theorem 4.2 and Theorem 4.3 shows that our approach is optimal.

▶ **Corollary 4.4.** *With the given strategy for a uniform system where each item has length* ℓ*, the center of gravity deviates in* $[0, \frac{\ell}{1+\mu}]$*, which is optimal.*

4.2 Optimally Loading Without Stacking but With Minimal Space

Assume that the height of the ship to be loaded does not allow stacking items. This makes it necessary to ensure that the space consumption of the packing is minimal. We restrict ourselves to plane placements such that each state is connected. For simplicity, we assume w.l.o.g. that $\ell_1 \geq \cdots \geq \ell_n$ holds. First one can simply argue that $\Delta(p) \geq \frac{\ell_2}{4}$ holds for an arbitrary connected plane placement p of S. Subsequently we give an algorithm that realizes this lower bound.

A fundamental key for this subcase is that the center of gravity of a connected plane state is the midpoint of the induced overall interval.

▶ **Observation 1.** *Let s be a plane state such that the union of the corresponding intervals is an interval $[a, b] \subset \mathbb{R}$. Then $C(s) = \frac{a+b}{2}$.*

The algorithm works as follows. First, sort the items by decreasing value and place the item ℓ_1 at position $\frac{\ell_2}{4}$. After that, place all other items successively on alternating sides such that there are no gaps in the each intermediate placement. This approach yields a deviation of the center of gravity in $[-\frac{\ell_2}{4}, \frac{\ell_2}{4}]$.

▶ **Lemma 4.5.** *For each plane placement p of S, we have $\Delta(p) \geq \frac{\ell_2}{4}$.*

▶ **Lemma 4.6.** *We can compute a placement p of S such that $\Delta(p) \leq \frac{\ell_2}{4}$.*

The combination of Lemma 4.5 and Lemma 4.6 implies that our approach for connected placements is optimal.

▶ **Corollary 4.7.** *Given an arbitrary system, there is a polynomial-time algorithm for optimally loading a general set of items without stacking and under the constraint of minimal space consumption for all intermediate stages.*

5 Conclusion

We have introduced a new family of problems that seek to balance objects, controlling the variation of their center of gravity during the loading and unloading of the objects. We have provided hardness results and optimal or constant-factor approximation algorithms.

There are various related challenges. These include sequencing problems with multiple loading and unloading stops (which arise in vehicle routing or tour planning for container ships); variants in which items can be shifted in a continuous fashion; batch scenarios in which multiple items are loaded or unloaded at once (making it possible to maintain better balance, but also increasing the space of possible choices); and higher-dimensional variants, possibly with inhomogeneous space constraints. All these are left for future work.

References

1 Samir V. Amiouny, John J. Bartholdi, John H. Vande Vate, and Jixian Zhang. Balanced loading. *Operations Research*, 40(2):238–246, 1992.

2 S. P. Fekete, S. von Höveling, J. S. B. Mitchell, C. Rieck, C. Scheffer, A. Schmidt, and J. R. Zuber. Don't Rock the Boat: Algorithms for Balanced Dynamic Loading and Unloading. *CoRR*, abs/1712.06498, 2017. http://arxiv.org/abs/1712.06498.

3 Hermann Gehring, K Menschner, and M Meyer. A computer-based heuristic for packing pooled shipment containers. *Eur. J. Oper. Res.*, 44(2):277–288, 1990.

4 PC Gilmore and Ralph E Gomory. Multistage cutting stock problems of two and more dimensions. *Operations research*, 13(1):94–120, 1965.

5 Marcel Mongeau and Christian Bes. Optimization of aircraft container loading. *IEEE Transactions on Aerospace and Electronic Systems*, 39(1):140–150, 2003.

6 Sergey Sevastianov. On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics*, 55(1):59–82, 1994.

7 Sergey Sevastianov. Nonstrict vector summation in multi-operation scheduling. *Annals of Operations Research*, 83:179–212, 1998.

8 Wouter Souffriau, Peter Demeester, G Vanden Berghe, and Patrick De Causmaecker. The aircraft weight and balance problem. *Proceedings of ORBEL*, 22:44–45, 1992.

Almost-equidistant sets*

Martin Balko[1,5], Attila Pór[2], Manfred Scheucher[3], Konrad Swanepoel[4], and Pavel Valtr[1]

1 **Department of Applied Mathematics and Institute for Theoretical Computer Science, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic,** balko@kam.mff.cuni.cz
2 **Department of Mathematics, Western Kentucky University, Bowling Green, KY 42101,** attila.por@wku.edu
3 **Institute of Mathematics, Technische Universität Berlin, Germany,** scheucher@math.tu-berlin.de
4 **Department of Mathematics, London School of Economics and Political Science, London, United Kingdom,** k.swanepoel@lse.ac.uk
5 **Department of Computer Science, Faculty of Natural Sciences, Ben-Gurion University of the Negev, Beer Sheva, Israel**

Abstract

For a positive integer d, a set of points in d-dimensional Euclidean space is called *almost-equidistant* if for any three points from the set, some two are at unit distance. Let $f(d)$ denote the largest size of an almost-equidistant set in d-space.

It is known that $f(2) = 7$, $f(3) = 10$, and that the extremal almost-equidistant sets are unique. We have independent, computer-assisted proofs of these statements. It is also known that $f(5) \geq 16$. We further show that $12 \leq f(4) \leq 13$, $f(5) \leq 20$, $18 \leq f(6) \leq 26$, $20 \leq f(7) \leq 34$, and $f(9) \geq f(8) \geq 24$. Up to dimension 7, our work is based on various computer searches, and in dimensions 6 to 9, we have constructions based on the known construction for $d = 5$.

For every dimension $d \geq 3$, we have an example of an almost-equidistant set of $2d + 4$ points in the d-space and we prove the asymptotic upper bound $f(d) \leq O(d^{3/2})$.

1 Introduction and our results

For a positive integer d, we denote the d-dimensional Euclidean space by $\mathbb{R}^d$. A set V of (distinct) points in $\mathbb{R}^d$ is called *almost-equidistant* if among any three of them, some pair is at distance 1. Let $f(d)$ be the maximum size of an almost-equidistant set in $\mathbb{R}^d$. For example, the vertex set of the well-known Moser spindle (Figure 1(a)) is an almost-equidistant set of 7 points in the plane and thus $f(2) \geq 7$.

In this paper we study the growth rate of the function f. We first consider the case when the dimension d is small and give some almost tight estimates on $f(d)$ for $d \leq 9$. Then we turn to higher dimensions and show $2d + 4 \leq f(d) \leq O(d^{3/2})$.

It is trivial that $f(1) = 4$ and that, up to congruence, there is a unique almost-equidistant set on 4 points in $\mathbb{R}$. Bezdek, Naszódi, and Visy [5] showed that an almost-equidistant set in the plane has at most 7 points. Talata (personal communication) showed in 2007 that there is a unique extremal set. We have a simple, computer-assisted proof of this result [3].

* M. Balko has received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement no. 678765. The authors all acknowledge the support of the ERC Advanced Research Grant no. 267165 (DISCONV). M. Balko and P. Valtr acknowledge support of the grant P202/12/G061 of the Czech Science Foundation (GAČR). M. Scheucher acknowledges support from TBK Automatisierung und Messtechnik GmbH.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

(a) (b)

Figure 1 (a) The Moser spindle. (b) An almost-equidistant set in $\mathbb{R}^3$ on 10 points.

▶ **Theorem 1.1** (Talata, 2007). *The largest number of points in an almost-equidistant set in $\mathbb{R}^2$ is 7, that is, $f(2) = 7$. Moreover, up to congruence, there is only one planar almost-equidistant set with 7 points, namely the Moser spindle.*

Figure 1(b) shows an example of an almost-equidistant set of 10 points in $\mathbb{R}^3$. It is made by taking a so-called *biaugmented tetrahedron*, which is a non-convex polytope formed by gluing three unit tetrahedra together at faces, and rotating a copy of it along the axis through the two simple vertices so that two additional unit-distance edges are created. This unit-distance graph is used in a paper of Nechushtan [12] to show that the chromatic number of $\mathbb{R}^3$ is at least 6. Györey [8] showed, by an elaborate case analysis, that this is the unique largest almost-equidistant set in $\mathbb{R}^3$. We have an independent, computer-assisted proof [3].

▶ **Theorem 1.2** (Györey [8]). *The largest number of points in an almost-equidistant set in $\mathbb{R}^3$ is 10, that is, $f(3) = 10$. Moreover, up to congruence, there is only one almost-equidistant set in $\mathbb{R}^3$ with 10 points.*

In dimension 4, we have only been able to obtain the following bounds.

▶ **Theorem 1.3.** *The largest number of points in an almost-equidistant set in $\mathbb{R}^4$ is either 12 or 13, that is, $f(4) \in \{12, 13\}$.*

The lower bound comes from a generalization of the example in Figure 1(b); see also Theorem 1.6. The proofs of the upper bounds in the above theorems are computer assisted. Based on some numerical work to find approximate realisations of graphs, we believe, but cannot prove rigorously, that an almost-equidistant set of 13 points in $\mathbb{R}^4$ does not exist.

▶ **Conjecture 1.4.** *The largest number of points in an almost-equidistant set in $\mathbb{R}^4$ is 12.*

In dimension 5, Larman and Rogers [11] showed that $f(5) \geq 16$ by a construction based on the so-called Clebsch graph. In dimensions 6 to 9, we use their construction to obtain lower bounds that are stronger than the lower bound $2d + 4$ stated below in Theorem 1.6. We again complement this with some computer-assisted upper bounds.

▶ **Theorem 1.5.** *The largest number of points in an almost-equidistant set in $\mathbb{R}^5$, $\mathbb{R}^6$, $\mathbb{R}^7$, $\mathbb{R}^8$ and $\mathbb{R}^9$ satisfy the following: $16 \leq f(5) \leq 20$, $18 \leq f(6) \leq 26$, $20 \leq f(7) \leq 34$, $24 \leq f(8) \leq 41$, and $24 \leq f(9) \leq 49$.*

The *unit-distance graph* of an almost-equidistant point set P in $\mathbb{R}^d$ is obtained from P by letting P be its vertex set and by placing an edge between pairs of points at unit distance.

For every $d \in \mathbb{N}$, a unit-distance graph in $\mathbb{R}^d$ does not contain K_{d+2} (see Corollary 2.2) and the complement of the unit-distance graph of an almost-equidistant set is triangle-free.

Dimension d	1	2	3	4	5	6	7	8	9	$d \geq 9$
Lower bounds on $f(d)$	4	7	10	12	16	18	20	24	24	$2d+4$
Upper bounds on $f(d)$	4	7	10	13	20	26	34	41	49	$4(d^{3/2}+\sqrt{d})$

Table 1 Lower and upper bounds on the largest size of an almost-equidistant set in $\mathbb{R}^d$.

Thus we have $f(d) \leq R(d+2,3)-1$, where $R(d+2,3)$ is the *Ramsey number* of K_{d+2} and K_3, that is, the smallest positive integer N such that for every graph G on N vertices there is a copy of K_{d+2} in G or a copy of K_3 in the complement of G.

Ajtai, Komlós, and Szemerédi [1] showed $R(d+2,3) \leq O(d^2/\log d)$ and this bound is known to be tight [9]. We thus have an upper bound $f(d) \leq O(d^2/\log d)$, which, as we show below, is not tight. For small values of d where the Ramsey number $R(d+2,3)$ is known or has a reasonable upper bound, we obtain an upper bound for $f(d)$. In particular, we get $f(5) \leq 22$, $f(6) \leq 27$, $f(7) \leq 35$, $f(8) \leq 41$, and $f(9) \leq 49$ [16]. For $d \in \{5,6,7\}$, we slightly improve these estimates to the bounds from Theorem 1.5 using our computer-assisted approach [3].

We now turn to higher dimensions. The obvious generalization of the Moser spindle gives an example of an almost-equidistant set of $2d+3$ points in $\mathbb{R}^d$. The next theorem improves this by 1. It is a generalization of the almost-equidistant set on 10 points in $\mathbb{R}^3$ from Figure 1(b).

▶ **Theorem 1.6.** *For every $d \geq 3$, there is an almost-equidistant set in $\mathbb{R}^d$ with $2d+4$ points.*

Rosenfeld [17] showed that an almost-equidistant set on a sphere in $\mathbb{R}^d$ of radius $1/\sqrt{2}$ has size at most $2d$, which is best possible. Rosenfeld's proof, which uses linear algebra, was adapted by Bezdek and Langi [4] to spheres of other radii. They showed that an almost-equidistant set on a sphere in $\mathbb{R}^d$ of radius $\leq 1/\sqrt{2}$ has at most $2d+2$ elements, which is attained by the union of two d-simplices inscribed in the same sphere.

Pudlák [15] and Deaett [6] gave simpler proofs of Rosenfeld's result. Our final result is an asymptotic upper bound for the size of an almost-equidistant set, based on Deaett's proof [6].

▶ **Theorem 1.7.** *An almost-equidistant set of points in $\mathbb{R}^d$ has cardinality $O(d^{3/2})$.*

We note that Polyanskii [13] recently found an upper bound of $O(d^{13/9})$ for the size of an almost-equidistant set in $\mathbb{R}^d$ and Kupavskii, Mustafa, and Swanepoel [10] and Polyanskii [14] improved this to $O(d^{4/3})$. Both papers use ideas from our proof of Theorem 1.7.

In this paper, we use $\|v\|$ to denote the Euclidean norm of a vector v from $\mathbb{R}^d$. For a subset S of $\mathbb{R}^d$, we use $\mathrm{span}(S)$ to denote the linear hull of S.

In the rest of the paper we sketch the proof of Theorem 1.7. The proofs of the remaining statements, as well as some auxiliary claims, can be found in the full version of this paper [3]. The full version also contains a computer program that enumerates all graphs that are unit-distance graphs of almost-equidistant sets up to a certain size and dimension. The source code of our programs and the files are available on a separate website [18].

2 Proof of Theorem 1.7

In this section, we sketch the proof of Theorem 1.7 by showing the upper bound $f(d) \leq O(d^{3/2})$. As a first step towards this proof, we state the following lemma that characterizes sets of points lying at the unit distance from vertices of a regular simplex with unit-length edges. For the statement of the lemma, we recall that a sphere of dimension d is a surface of a $(d+1)$-dimensional ball.

▶ **Lemma 2.1.** *For $d, k \in \mathbb{N}$, let C be a set of k points in $\mathbb{R}^d$ such that the distance between any two of them is 1. Let $c := \frac{1}{k}\sum_{p\in C} p$ be the centroid of C and let $A := \operatorname{span}(C-c)$. Then the set of points equidistant from all points of C is the affine space $c + A^\perp$ orthogonal to A and passing through c. Furthermore, the intersection of all unit spheres centred at the points in C is the $(d-k)$-dimensional sphere of radius $\sqrt{(k+1)/(2k)}$ centred at c and contained in $c + A^\perp$.*

▶ **Corollary 2.2.** *For $d \in \mathbb{N}$, every subset of $\mathbb{R}^d$ contains at most $d+1$ points that are pairwise at unit distance.*

The following lemma is a well-known result that bounds the rank of a square matrix from below in terms of the entries of the matrix [2, 6, 15].

▶ **Lemma 2.3.** *Let $A = [a_{i,j}]$ be a non-zero symmetric $m \times m$ matrix with real entries. Then*

$$\operatorname{rank} A \geq \Big(\sum_{i=1}^{m} a_{i,i}\Big)^2 \Big/ \sum_{i=1}^{m}\sum_{j=1}^{m} a_{i,j}^2.$$

The last lemma before the proof of Theorem 1.7 can be proved by a calculation, using its assumption that the vectors v_i have pairwise inner products ε, so they differ from an orthogonal set by some skewing.

▶ **Lemma 2.4.** *For $n, t \in \mathbb{N}$ with $t \leq n$, let $w_1, \ldots, w_t$ be unit vectors in $\mathbb{R}^n$ such that $\langle w_i, w_j\rangle = \varepsilon$ for all i, j with $1 \leq i < j \leq t$, where $\varepsilon \in [0, 1)$. Then the set $\{w_1, \ldots, w_t\}$ can be extended to $\{w_1, \ldots, w_n\}$ such that $\langle w_i, w_j\rangle = \varepsilon$ for all i, j with $1 \leq i < j \leq n$, and such that for some orthonormal basis $e_1, \ldots, e_n$ we have $w_i = \frac{e_i+\lambda e}{\|e_i+\lambda e\|}$ $(i = 1, \ldots, n)$, where*

$$\lambda := \frac{-1 + \sqrt{1 + \varepsilon n/(1-\varepsilon)}}{n} \quad \text{and} \quad e := \sum_{j=1}^{n} e_j = \frac{1}{\sqrt{1+(n-1)\varepsilon}} \sum_{j=1}^{n} w_j.$$

Moreover, $\|e_i + \lambda e\|^2 = (1-\varepsilon)^{-1}$ for each $i \in \{1, \ldots, n\}$ and for every $x \in \mathbb{R}^n$ we have

$$\sum_{j=1}^{n} (\langle x, w_j\rangle - \varepsilon)^2 = (1-\varepsilon)(\|x\|^2 - \varepsilon) + \varepsilon\left(\langle x, e\rangle - \sqrt{1+(n-1)\varepsilon}\right)^2.$$

We are now ready to prove Theorem 1.7. For $d \geq 2$, let $V \subset \mathbb{R}^d$ be an almost-equidistant set. Let $G = (V, E)$ be the unit-distance graph of V and let $k := \lfloor 2\sqrt{d}\rfloor$. Note that $2 \leq k \leq d$.

Let $S \subseteq V$ be a set of k points such that the distance between any two of them is 1. If such a set does not exist, then, since the complement of G does not contain a triangle, we have $|V| < R(k, 3)$, where $R(k, 3)$ is the Ramsey number of K_k and K_3. Using the bound $R(k, 3) \leq \binom{k+3-2}{3-1}$ obtained by Erdős and Szekeres [7], we derive $|V| < \binom{2\sqrt{d}+1}{2} = 2d + \sqrt{d}$. Thus we assume in the rest of the proof that S exists.

Let B be the set of common neighbours of S, that is, $B := \{x \in V \mid \|x - s\| = 1\ \forall s \in S\}$. Since V is almost-equidistant, the set of non-neighbours of any vertex of G is a clique and so it has size at most $d + 1$ by Corollary 2.2. Every vertex from $V \setminus B$ is a non-neighbour of some vertex from S and thus it follows that $|V \setminus B| \leq k(d + 1)$.

We now estimate the size of B. By Lemma 2.1 applied to S, the set B lies on a sphere of radius $\sqrt{(k+1)/2k}$ in an affine subspace of dimension $d - k + 1$. We may take the centre of this sphere as the origin, and rescale by $\sqrt{2k/(k+1)}$ to obtain a set B' of m unit vectors $v_1, \ldots, v_m \in \mathbb{R}^{d-k+1}$ where $m := |B|$. For any three of the vectors from B', the

distance between some two of them is $\sqrt{2k/(k+1)}$. For two such vectors v_i and v_j with $\|v_i - v_j\|^2 = 2k/(k+1)$, the facts $\|v_i - v_j\|^2 = \|v_i\|^2 + \|v_j\|^2 - 2\langle v_i, v_j\rangle$ and $\|v_i\|^2 = \|v_j\|^2 = 1$ imply $\langle v_i, v_j\rangle = \varepsilon$, where $\varepsilon := 1/(k+1)$. Note that the opposite implication also holds. That is, if $\langle v_i, v_j\rangle = \varepsilon$, then v_i and v_j are at distance $\sqrt{2k/(k+1)}$.

Let $A = [a_{i,j}]$ be the $m \times m$ matrix defined by $a_{i,j} := \langle v_i, v_j\rangle - \varepsilon$. Clearly, A is a symmetric matrix with real entries. If $m \geq d - k + 2$, then A is also non-zero, as G contains no K_{d+2} and every vertex from B is adjacent to every vertex from S in G. We recall that $\operatorname{rank} XY \leq \min\{\operatorname{rank} X, \operatorname{rank} Y\}$ and $\operatorname{rank}(X+Y) \leq \operatorname{rank} X + \operatorname{rank} Y$ for two matrices X and Y. Since $B' = \{v_1, \ldots, v_m\} \subset \mathbb{R}^{d-k+1}$ and

$$A = \begin{bmatrix} v_1 & v_2 & \cdots & v_m \end{bmatrix}^\top \begin{bmatrix} v_1 & v_2 & \cdots & v_m \end{bmatrix} - \varepsilon J,$$

where J is the $m \times m$ matrix with each entry equal to 1, we have

$$\operatorname{rank} A \leq d - k + 2. \tag{1}$$

By Lemma 2.3,

$$\operatorname{rank} A \geq \frac{\left(\sum_{i=1}^{m} a_{i,i}\right)^2}{\sum_{i=1}^{m}\sum_{j=1}^{m} a_{i,j}^2} = \frac{m^2(1-\varepsilon)^2}{\sum_{i=1}^{m}\sum_{j=1}^{m} (\langle v_i, v_j\rangle - \varepsilon)^2}. \tag{2}$$

For $i \in \{1, \ldots, m\}$, let N_i be the set of vectors from B' that are at distance $\sqrt{2k/(k+1)}$ from v_i. That is, $N_i := \{v_j \in B' \mid \langle v_i, v_j\rangle = \varepsilon\}$. Then for each fixed v_i we have

$$\sum_{j=1}^{m} (\langle v_i, v_j\rangle - \varepsilon)^2 = (1-\varepsilon)^2 + \sum_{v_j \in N_i} 0 + \sum_{v_j \in B' \setminus (N_i \cup \{v_i\})} (\langle v_i, v_j\rangle - \varepsilon)^2. \tag{3}$$

Note that the vectors from $B' \setminus (N_i \cup \{v_i\})$ have pairwise inner products ε, as neither of them is at distance $\sqrt{2k/(k+1)}$ from v_i, and thus $|B' \setminus (N_i \cup \{v_i\})| \leq d - k + 2$. In fact, we even have $|B' \setminus (N_i \cup \{v_i\})| \leq d - k + 1$, since B' contains only unit vectors and any subset of $d - k + 2$ points from B' with pairwise distances $\sqrt{2k/(k+1)}$ would form the vertex set of a regular $(d-k+1)$-simplex with edge lengths $\sqrt{2k/(k+1)}$ centred at the origin. However, then the distance from the centroid of such a simplex to its vertices would be equal to $\sqrt{k(d-k+1)/((k+1)(d-k+2))} \neq 1$, which is impossible.

Thus setting $n := d-k+1$ and $t := |B' \setminus (N_i \cup \{v_i\})|$, we have $t \leq n$. Applying Lemma 2.4 to the t vectors from $B' \setminus (N_i \cup \{v_i\}) \subseteq \mathbb{R}^n$ with $\varepsilon = (k+1)^{-1}$ and $x = v_i$, we see that the last sum in (3) is at most

$$(1-\varepsilon)^2 + \varepsilon\left(\langle v_i, e\rangle - \sqrt{1 + (d-k)\varepsilon}\right)^2,$$

where $e = \sum_{j=1}^{d-k+1} e_j$ for some orthonormal basis $e_1, \ldots, e_{d-k+1}$ of $\mathbb{R}^{d-k+1}$.

By the Cauchy–Schwarz inequality,

$$\begin{aligned}\left(\langle v_i, e\rangle - \sqrt{1 + (d-k)\varepsilon}\right)^2 &< \left(\sqrt{d-k+1} + \sqrt{1 + (d-k)\varepsilon}\right)^2 \\ &= d - k + 1 + 2\sqrt{d-k+1}\sqrt{1+(d-k)\varepsilon} + 1 + (d-k)\varepsilon < 4(d-k+1).\end{aligned}$$

Recall that $k \geq 2$. Using $\varepsilon = (k+1)^{-1}$, we obtain

$$\sum_{j=1}^{m} \big(\langle v_i, v_j\rangle - \varepsilon\big)^2 < 2(1-\varepsilon)^2 + 4\varepsilon(d-k+1) = 4\varepsilon d + 2(1+\varepsilon)^2 - 4 < 4\varepsilon d.$$

If we substitute this upper bound back into (2), then with (1) we obtain that $d-k+2 > m^2(1-\varepsilon)^2/(4m\varepsilon d)$ and thus $m < (4\varepsilon d)(d-k+2)/(1-\varepsilon)^2$. Using the choice $k = \lfloor 2\sqrt{d} \rfloor$ and the expression $\varepsilon = (k+1)^{-1}$, we obtain $(d-k+2)/(1-\varepsilon)^2 < d$, if $d \geq 8$, and thus $m < 4d^2/(k+1)$. Altogether, we have $m \leq \max\{d-k+1, 4d^2/(k+1)\} = 4d^2/(k+1)$. It follows that $|V| \leq k(d+2)+4d^2/(k+1)$. Again, using the choice $k = \lfloor 2\sqrt{d} \rfloor \in (2\sqrt{d}-1, 2\sqrt{d}]$, we conclude that $|V| < 2\sqrt{d}(d+2) + 4d^2/(2\sqrt{d}) = 4d^{3/2} + 4\sqrt{d}$. This finishes the proof of Theorem 1.7.

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi, *A note on Ramsey numbers*, J. Combin. Theory Ser. A **29** (1980), no. 3, 354–360.

2 Noga Alon, *Problems and results in extremal combinatorics. I*, Discrete Math. **273** (2003), no. 1-3, 31–53, EuroComb'01 (Barcelona).

3 Martin Balko, Attila Pór, Manfred Scheucher, Konrad Swanepoel, and Pavel Valtr, *Almost-equidistant sets*, 2017, `http://arxiv.org/abs/1706.06375`

4 Károly Bezdek and Zsolt Lángi, *Almost equidistant points on S^{d-1}*, Period. Math. Hungar. **39** (1999), no. 1-3, 139–144, Discrete geometry and rigidity (Budapest, 1999).

5 Károly Bezdek, Márton Naszódi, and Balázs Visy, *On the mth Petty numbers of normed spaces*, Discrete geometry, Monogr. Textbooks Pure Appl. Math., vol. 253, Dekker, New York, 2003, pp. 291–304.

6 Louis Deaett, *The minimum semidefinite rank of a triangle-free graph*, Linear Algebra Appl. **434** (2011), no. 8, 1945–1955.

7 Paul Erdős and George Szekeres, *A combinatorial problem in geometry*, Compositio Math. **2** (1935), 463–470.

8 Bernadett Györey, *Diszkrét metrikus terek beágyazásai*, Master's thesis, Eötvös Loránd University, Budapest, 2004, in Hungarian.

9 Jeong H. Kim, *The Ramsey number $R(3,t)$ has order of magnitude $t^2/\log t$*, Random Structures and Algorithms **7** (1995), no. 3, 173–207.

10 Andrey Kupavskii, Nabil Mustafa, and Konrad J. Swanepoel, *Bounding the size of an almost-equidistant set in Euclidean space*, 2017, `http://arxiv.org/abs/1708.01590`

11 David G. Larman and Claude A. Rogers, *The realization of distances within sets in Euclidean space*, Mathematika **19** (1972), 1–24.

12 Oren Nechushtan, *On the space chromatic number*, Discrete Math. **256** (2002), no. 1–2, 499–507.

13 Alexandr Polyanskii, *On almost-equidistant sets*, 2017, `http://arxiv.org/abs/1707.00295`

14 Alexandr Polyanskii, *On almost-equidistant sets - II*, 2017, `http://arxiv.org/abs/1708.02039`

15 Pavel Pudlák, *Cycles of nonzero elements in low rank matrices*, Combinatorica **22** (2002), no. 2, 321–334, Special issue: Paul Erdős and his mathematics.

16 Stanisław P. Radziszowski, *Small Ramsey numbers*, Electron. J. Combin. **1** (1994), Dynamic Survey 1, 30 pp.

17 Moshe Rosenfeld, *Almost orthogonal lines in E^d*, Applied geometry and discrete mathematics. The Victor Klee Festschrift (Peter Gritzmann and Bernd Sturmfels, eds.), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 4, Amer. Math. Soc., Providence, RI, 1991, pp. 489–492.

18 Manfred Scheucher, *Homepage of almost-equidistant sets*, `http://page.math.tu-berlin.de/~scheuch/supplemental/almost_equidistant_sets`.

Solving Large-Scale Minimum-Weight Triangulation Instances to Provable Optimality*

Andreas Haas[1]

1 **Department of Computer Science, TU Braunschweig, 38106 Braunschweig, Germany.** haas@ibr.cs.tu-bs.de

Abstract

We consider practical methods for the problem of finding a minimum-weight triangulation (MWT) of a planar point set, a classic problem of computational geometry with many applications. While Mulzer and Rote proved in 2006 that computing an MWT is NP-hard, Beirouti and Snoeyink showed in 1998 that computing provably optimal solutions for MWT instances of up to 80,000 *uniformly distributed* points is possible, making use of clever heuristics that are based on geometric insights. We show that these techniques can be refined and extended to instances of much bigger size and different type, based on an array of modifications and parallelizations in combination with more efficient geometric encodings and data structures. As a result, we are able to solve MWT instances with up to 30,000,000 uniformly distributed points in less than 4 minutes to provable optimality. Moreover, we can compute optimal solutions for a vast array of other benchmark instances that are *not* uniformly distributed, including normally distributed instances (up to 30,000,000 points), all point sets in the TSPLIB (up to 85,900 points), and VLSI instances with up to 744,710 points. This demonstrates that from a practical point of view, MWT instances can be handled quite well, despite their theoretical difficulty.

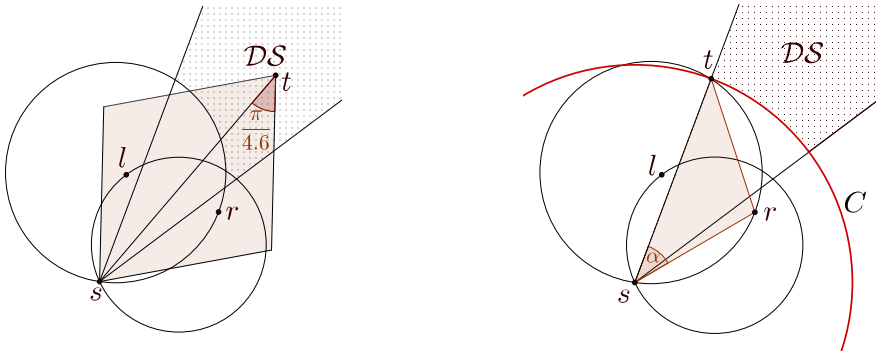
1 Introduction

Triangulating a set of points in the plane is a classic problem in computational geometry: given a planar point set S, find a maximal set of non-crossing line segments connecting the points in S. Triangulations have many real-world applications, for example in terrain modeling, finite element mesh generation and visualization. In general, a point set has exponentially many possible triangulations and a natural question is to ask for a triangulation that is optimal with respect to some optimality criterion. A natural criterion is to minimize the total weight of the resulting triangulation. As Mulzer and Rote [11] showed, it is NP-hard to compute a minimum-weight triangulation (MWT).

Practical approaches for computing an MWT are based on heuristics for including or excluding edges with certain properties from any minimum-weight triangulation. Das and Joseph [4] showed that every edge in an MWT has the *diamond property*. An edge e cannot be in MWT(S) if both of the two isosceles triangles with base e and base angle $\pi/8$ contain other points of S. Drysdale et al. [7] improved the angle to $\pi/4.6$. This can greatly reduce the edge set and works exceedingly well on uniformly distributed point sets, for which only $O(n)$ edges remain in expectation. Dickerson et al. [5,6] proposed the *LMT-skeleton heuristic*, based on a local criterion fulfilled by every edge in MWT(S). The LMT-skeleton algorithm often yields a connected graph, and the remaining polygonal faces can be triangulated with dynamic programming to obtain an MWT. Combining the diamond property and the LMT-skeleton makes it possible to compute the MWT for large, well-behaved point sets. Beirouti and Snoeyink [2] showed an efficient implementation of these two heuristics and they reported that their implementation could compute the exact MWT of 40,000 uniformly

* A full version of the paper is available at [9].

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.



(a) Points l and r induce a region $\mathcal{DS}$ such that all edges $e = st$ with $t \in \mathcal{DS}$ fail the diamond test. $\mathcal{DS}$ is called a dead sector (dotted area).

(b) Simplified dead sector $\mathcal{DS}$ is bounded by two rays and circle C. C is induced by the longer of the two edges sl resp. sr and angle α.

Figure 1 Dead sectors.

distributed points in less than 5 minutes and even up to 80,000 points with the improved diamond property.

We revisit diamond test and LMT-skeleton based on Beirouti's and Snoeyink's [2] ideas and describe several improvements. Our bucketing scheme for the diamond test does not rely on a uniform point distribution and filters more edges. For the LMT-skeleton we provide a number of algorithm engineering modifications. These contain a data partitioning scheme for parallelized implementation and other changes for efficiency. We also use an improvement suggested by Aichholzer et al. [1]. Furthermore, we implemented, streamlined and evaluated our implementation on various point sets. For the uniform case, we computed the MWT of 30,000,000 points in less than 4 minutes on commodity hardware; the limiting factor arose from the memory of a standard machine, not from the runtime. We achieved the same performance for normally distributed point sets. The third class of point sets were benchmark instances from the TSPLIB [12] (based on a wide range of real-world and clustered instances) and the VLSI library. These reached a size up to 744,710 points. This shows that from a practical point of view, a wide range of huge MWT instances can be solved to provable optimality with the right combination of theoretical insight and algorithm engineering.

2 Our Improvements and Optimizations

2.1 Diamond Property

For a uniformly distributed point set S with n points, the expected number of edges to pass the diamond test is only $O(n)$. More precisely, Beirouti and Snoeyink [2] state that the number is less than $3\pi n/\sin(\alpha)$, where α is the base angle for the diamond property. We were able to tighten this value.

▶ **Theorem 2.1.** *For a uniformly distributed point set, the expected number of edges that pass the diamond test is less than* $3\pi n/\tan(\alpha)$.

For $\alpha = \pi/4.6$ less than $11.5847n$ edges are expected to pass the test, which is very close to the values observed and achieved by our implementation; see Table 1 in Section 3. In contrast, the value achieved by the implementation of Beirouti and Snoeyink is $\approx 14.3n$ [2].

2.2 Dead Sectors and Bucketing

Our bucketing scheme is based on the same idea of *dead sectors* (see Figure 1a) as described by Beirouti and Snoeyink [2]. We simplify the shape of dead sectors: Instead of bounding

a sector $\mathcal{DS}$ by two circles (as shown in Figure 1a), we only use a single big circle C with center s at the expense of losing a small part of $\mathcal{DS}$. This allows representing dead sectors by just three numbers: an interval of two polar angles, and a squared radius δ; see Figure 1b.

The main ingredient for our bucketing scheme is a spatial search tree with support for incremental nearest neighbor queries, such as a quadtree. Incremental nearest neighbor search queries allow to traverse all nearest neighbors of a point in order of increasing distance. Such queries can be implemented with a priority queue that stores all tree nodes encountered during tree traversal together with the distances to their resp. bounding box (see Hjaltason and Samet [10]). Pruning tree nodes whose bounding box lie in dead sectors is rather simple as follows: consider a nearest neighbor query for point s: when we are about to push a new node n into the priority queue, we compute the smallest polar angle interval I that encloses the bounding box of n and discard n if I is contained in the dead sectors computed so far.

Because nearest neighbors and tree nodes are processed in order of increasing distance, we can store sectors in two stages. On creation, they are inserted into a FIFO-queue; later only the interval component is inserted in a search filter used by the tree. The queue can be seen as a set of pending dead sectors with attached activation distance δ. As soon as we process a point t with $\mathrm{d}(s,t) > \delta$ we can insert the corresponding interval into our filter.

This leaves deciding which points are used to construct dead sectors. We store all points encountered during an incremental search query in an ordered set N, sorted by their polar angle with respect to s. Each time we find a new point t, we insert it into N; dead sectors are computed with the predecessor and the successor of t in N. Computing δ for new sectors only requires multiplying the current squared distance to t with a precomputed constant. The diamond property of edge st is tested against a subset of N.

If we apply the above procedure to every single point, we generate each edge twice, once on each of the two endpoints. Therefore, we output only those edges $e = st$ such that $s < t$, i.e., s is lexicographically smaller than t. As a consequence, we can exclude a part of the left half-space right from the beginning by inserting an initial dead sector $\mathcal{DS}_0 = (1/2\pi + \alpha, 3/2\pi - \alpha)$ at distance 0. Points in the two wedges $(1/2\pi, 1/2\pi + \alpha]$ and $[3/2\pi - \alpha, 3/2\pi]$ are specially treated because they are still useful to generate dead sectors for the right half-space.

2.3 LMT-Skeleton

For "nicely" distributed point sets, a limiting factor of the heuristic is the space required to store the half-edge data structure in memory. We reduce storage overhead by storing all edges in a single array sorted by source vertex (also known as a *compressed sparse row graph*). All outgoing edges of a single vertex are still radially sorted. In addition to the statuses *possible*, *certain*, *impossible*, we store whether an edge lies on the convex hull.

In essence our implementation is still the same as the one given by Beirouti and Snoeyink [2], however, with some optimizations applied. We refer to the central `while loop` in their implementation as the `LMT-Loop`. First, the convex hull edges are implicitly given during initialization of their half-edge structure and can be marked as such without any additional cost. Determining the convex hull edges beforehand allows to remove the case distinction inside the `LMT-Loop`, i.e., it removes all intersection tests that are applied to impossible edges. Secondly, sorting the stack by edge length destroys spatial ordering and the loss of locality of reference outweighs all gains on modern hardware. Without sorting, it is actually not necessary to push all edges onto the stack upfront. Lastly, with proper partitioning of the edges, the `LMT-Loop` can be executed in parallel – described in more detail in Section 2.4.

Additionally, we incorporated an improvement to the LMT-skeleton suggested by Aichholzer et al. [1]. Because the improved LMT-skeleton is computationally much more expensive, we apply it only to edges surviving an initial round of the normal LMT-heuristic.

n	Edges	Number of visited neighbors per point: Mean	SD	Min	Max	$\mathcal{DS} = 2\pi$
10^1	36.16 ±2.63	9 ±0	0 ±0	9 ±0	9 ±0	0 ±0
10^2	882.8 ±27.69	55.6 ±3.1	16.6 ±2.04	23.72 ±4.82	98.56 ±1.27	30.4 ±4.61
10^3	10,731.7 ±159.9	72.52 ±1.56	23.16 ±1.3	22.68 ±4.55	173 ±14.61	737.84 ±10.91
10^4	$1.1316 \cdot 10^5$ ±471.24	77.64 ±0.69	26.64 ±0.73	19.08 ±2.3	363.72 ±20	9,126.08 ±18.74
10^5	$1.15 \cdot 10^6$ ±1,538.64	72.84 ±0.29	23.76 ±0.47	15.96 ±1.61	846.24 ±24.4	97,200.9 ±40.29
10^6	$1.1562 \cdot 10^7$ ±4,737.67	74 ±0.51	25.76 ±0.39	13.28 ±1.31	2,884.96 ±38.53	$9.9117 \cdot 10^5$ ±61.86
10^7	$1.1579 \cdot 10^8$ ±19,254	77 ±0.6	27.24 ±0.79	11.88 ±0.99	9,567.52 ±78.84	$9.9721 \cdot 10^6$ ±100.61
10^8	$1.1585 \cdot 10^9$ ±56,063.1	72 ±0.94	24.08 ±0.69	10.6 ±0.49	25,017.8 ±107.4	$9.9911 \cdot 10^7$ ±239.64

Table 1 Diamond test on uniformly distributed points. The table shows statistics for 25 different instances. The extreme values are assumed by points at the point set boundary.

2.4 Parallelization

Because the LMT-heuristic performs only local changes, most edges can be processed in parallel without synchronization. Problems occur only if adjacent edges are processed concurrently (for the improved LMT-skeleton this is unfortunately not true, because marking an edge *impossible* affects a larger neighborhood of edges). To parallelize the normal LMT-heuristic, we implemented a solution based on data partitioning without explicit locking.

We recursively cut the vertices V into two disjoint sets $V = V_1 \cup V_2$ and process only those edges with both endpoints in V_1 (resp. V_2) in parallel. Define X as the cut set $\{\{s,t\} \in E \mid s \in V_1, t \in V_2\}$, i.e., all edges with one endpoint in V_1 and the other in V_2. While edges in $E(V_1)$ resp. $E(V_2)$ are processed in parallel by two threads, edges in X are accessed read-only by both threads and are handled after both threads join. This way we never process two edges with a common endpoint in parallel. To avoid a serial scan at the top, we push the actual work of computing X down to the leaves in the recursion tree. Scanning of the half-edge array starts at the leave nodes: processing of half-edges that belong to some cut set is postponed, instead they are passed back to the parent node. The parent in turn scans the edges it got from its two children, processes all edges it can and passes up the remaining ones. In other words, the final cut set X bubbles up in the tree, while all intermediate cuts are never explicitly computed. This way, partitioning on each level of the recursion tree only takes constant time, while the actual work is fully parallelized at the leaf level. After the LMT-heuristic completes, we are left with many polygonal faces that still need to be triangulated. Our implementation traverses the graph formed by the edges with one producer thread in order to collect all faces and multiple consumer threads to triangulate them with dynamic programming.

3 Computational Results

Computations were performed on a machine with an Intel i7-6700K quad-core and 64GB memory. The code was written in C++ and compiled with gcc 5.4.0.

3.1 Uniformly and Normally Distributed Point Sets

Table 1 shows results of our diamond test implementation on uniformly distributed point sets with sizes ranging from 10 to 10^8 points. The table shows the mean values and the standard deviation of 25 different instances. Each instance was generated by choosing n points uniformly from a square centered at the origin. The diamond test performs one incremental nearest neighbor query for each point in order to generate the edges that pass the test. The last column shows the number of queries where all nodes in the spatial tree were discarded because dead sectors covered the whole search space. The numbers show that this is the regular case; the exceptional cases occur at points near the point set "boundary".

n	Possible edges after Diamond		Possible edges after LMT		Possible edges after LMT+		Certain edges after LMT		Certain edges after LMT+		Simple Polygons	
$1 \cdot 10^1$	36.76	±2.78	3.8	±3.84	3.72	±3.62	19.32	±2.22	19.32	±2.22	0.68	±0.61
$1 \cdot 10^2$	871.92	±46.37	84.04	±20.14	74.56	±18.1	251.48	±7.12	252.28	±7.12	10.52	±2.55
$1 \cdot 10^3$	10,687.4	±146.68	1,150.32	±98.05	1,031.96	±86.46	2,540	±32.33	2,548.04	±31.41	128	±9.2
$1 \cdot 10^4$	$1.1322 \cdot 10^5$	±661.16	12,637	±281.25	11,271.76	±251.6	25,193.44	±73.29	25,287.56	±76.43	1,367.08	±24.65
$1 \cdot 10^5$	$1.1503 \cdot 10^6$	±1,696.31	$1.2941 \cdot 10^5$	±1,198.41	$1.1523 \cdot 10^5$	±973.14	$2.5129 \cdot 10^5$	±322.29	$2.5227 \cdot 10^5$	±306.72	13,819.44	±67.93
$1 \cdot 10^6$	$1.1563 \cdot 10^7$	±5,459.02	$1.3044 \cdot 10^6$	±2,708.78	$1.1617 \cdot 10^6$	±2,486.36	$2.5098 \cdot 10^6$	±847.61	$2.5194 \cdot 10^6$	±860.53	$1.3904 \cdot 10^5$	±232.43
$1 \cdot 10^7$	$1.1579 \cdot 10^8$	±17,587.01	$1.3074 \cdot 10^7$	±11,021.75	$1.1645 \cdot 10^7$	±8,825.57	$2.5088 \cdot 10^7$	±2,774.11	$2.5184 \cdot 10^7$	±2,727.23	$1.3931 \cdot 10^6$	±607.95
$3 \cdot 10^7$	$3.4747 \cdot 10^8$	±28,678.6	$3.9239 \cdot 10^7$	±18,919.14	$3.4949 \cdot 10^7$	±15,068.66	$7.5258 \cdot 10^7$	±4,637.8	$7.5547 \cdot 10^7$	±4,563.03	$4.1797 \cdot 10^6$	±969.6

Table 2 LMT-skeleton statistics on uniformly distributed point sets.

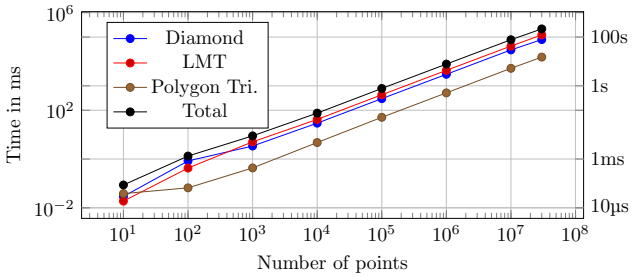


Figure 2 LMT-skeleton runtime on uniformly distributed point sets.

Table 2 shows statistics for the LMT-heuristic on uniformly distributed point sets. The instance sizes range from 10 points up to 30,000,000 points. For each size 25 different instances were generated. For the largest instances, the array storing the half-edges consumes nearly 39 GB of memory on its own. The serial initialization of the half-edge data structure, which basically amounts to radially sorting edges, takes longer than the parallel `LMT-Loop` on uniformly and normally distributed points. The improved LMT-skeleton by Aichholzer et al. is denoted LMT+ in the tables. The resulting skeleton was almost always connected in the computations and the number of remaining simple polygons that needed to be triangulated is shown in the last column. Only one instance of size $3 \cdot 10^7$ contained one small disconnected polygon. As we can see, the LMT-skeleton eliminates most of the possible edges with only $\approx 11\%$ remaining. The certain edges amount to $\approx 83\%$ of the complete triangulation. The improved LMT-skeleton reduces the amount of possible edges by another 10%, but it provides hardly any additional certain edges.

The results on normally distributed point sets are basically identical. Point coordinates were generated by two normally distributed random variables $X, Y \sim \mathcal{N}(\mu, \sigma^2)$, with mean $\mu = 0$ and standard deviation $\sigma \in \{1, 100, 100000\}$. The tables are given in the full version.

3.2 TSPLIB + VLSI

In addition to uniformly and normally distributed instances, we ran our implementation on instances found in the well-known TSPLIB [12], which contains a wide variety of instances with different distributions. The instances are drawn from industrial applications and from geographic problems. All 94 instances have a connected LMT-skeleton and can be fully triangulated with dynamic programming to obtain the minimum-weight triangulation. The total time it took to solve all instances of the TSPLIB was approximately 8.5 seconds.

Additional point sets can be downloaded at `http://www.math.uwaterloo.ca/tsp/vlsi/`. This collection of 102 TSP instances was provided by Andre Rohe, based on VLSI data sets studied at the Universität Bonn. The LMT-heuristic is sufficient to solve all instances, except `lra498378`, which contains two disconnected polygonal faces. Our implementation of the improved LMT-skeleton performs exceedingly bad on some of these instances; see Table 3. These instances contain empty regions with many points on the "boundary". Such regions are the worst-case for the heuristics because most edges inside them have the diamond property, which in turn leads to vertices with very high degree.

Table 3 VLSI instances with long runtime.

	Time in ms					
Instance	Total	DT	LMT-Init	LMT-Loop	LMT+	Dyn. Prog.
ara238025	15,325	4,954	446	496	9,279	148
lra498378	382,932	44,267	1,238	7,532	329,292	599
lrb744710	484,430	7,952	1,377	2,661	471,564	872
sra104815	1,937	559	191	198	922	65

4 Conclusion

We have shown that despite of the theoretical hardness of the MWT problem, a wide range of large-scale instances can be solved to optimality.

Difficulties for other instances arise from two sources. (1) Instances with almost regular k-gons with one or more points near the center can lead to highly disconnected LMT-skeletons (see Belleville et al. [3]) and require exponential time algorithms to complete the MWT. Preliminary experiments suggest that such configurations are best solved with integer programming. The instance by Belleville et al. can be solved with CPLEX in less than a minute, while the dynamic programming implementation of Grantson et al. [8] cannot solve it within several hours. (2) Instances containing empty regions with many points on their "boundary", such as empty k-gons and circles may be solvable in polynomial time, but trigger the worst-case behavior of the heuristics. Dealing with both is left for future work.

Acknowledgments. I want to thank Sándor Fekete and Victor Alvarez for useful discussions and suggestions that helped to improve the presentation of this paper.

References

1 O. Aichholzer, F. Aurenhammer, and R. Hainz. New Results on MWT Subgraphs. *Inf. Process. Lett.*, 69(5):215–219, 1999.
2 R. Beirouti and J. Snoeyink. Implementations of the LMT Heuristic for Minimum Weight Triangulation. In *Proc. Symp. Comp. Geom. (SoCG)*, pages 96–105, 1998.
3 P. Belleville, J. M. Keil, M. McAllister, and J. Snoeyink. On Computing Edges That Are In All Minimum-Weight Triangulations. In *Proc. Symp. Comp. Geom.*, pages 507–508, 1996.
4 G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Optimal Algorithms: Int. Symp. Varna, Bulgaria, May 29–June 2, 1989 Proc.*, pages 168–192. 1989.
5 M. Dickerson, J. M. Keil, and M. H. Montague. A Large Subgraph of the Minimum Weight Triangulation. *Discrete & Computational Geometry*, 18(3):289–304, 1997.
6 M. Dickerson and M. H. Montague. A (Usually?) Connected Subgraph of the Minimum Weight Triangulation. In *Proc. Symp. Comp. Geom. (SoCG)*, pages 204–213, 1996.
7 R. L. S. Drysdale, S. A. McElfresh, and J. Snoeyink. On exclusion regions for optimal triangulations. *Discrete Applied Mathematics*, 109(1-2):49–65, 2001.
8 M. Grantson, C. Borgelt, and C. Levcopoulos. Fixed Parameter Algorithms for the Minimum Weight Triangulation Problem. *Int. J. Comp. Geom. Appl.*, 18(3):185–220, 2008.
9 A. Haas. Solving Large-Scale Minimum-Weight Triangulation Instances to Provable Optimality. 2018. http://arxiv.org/abs/1802.06415.
10 G. R. Hjaltason and H. Samet. Ranking in Spatial Databases. In *Proc. 4th Int. Symp. Adv. Spatial Datab. (SSD)*, pages 83–95, London, UK, 1995.
11 W. Mulzer and G. Rote. Minimum-weight Triangulation is NP-hard. *J. ACM*, 55(2):11:1–11:29, May 2008.
12 G. Reinelt. TSPLIB- a Traveling Salesman Problem Library. *ORSA Journal of Computing*, 3(4):376–384, 1991.

Minimal Geometric Graph Representations of Order Types*

Oswin Aichholzer[1], Martin Balko[2,4], Michael Hoffmann[3], Jan Kynčl[4], Wolfgang Mulzer[5], Irene Parada[1], Alexander Pilz[3], Manfred Scheucher[6], Pavel Valtr[4], Birgit Vogtenhuber[1], and Emo Welzl[3]

1 **Institute of Software Technology, Graz University of Technology, Austria.**
{oaich,iparada,bvogt}@ist.tugraz.at
2 **Department of Computer Science, Ben Gurion University of the Negev, Israel.**
3 **Department of Computer Science, ETH Zürich, Switzerland.**
{michael.hoffmann,alexander.pilz,emo.welzl}@inf.ethz.ch
4 **Department of Applied Mathematics and Institute for Theoretical Computer Science (CE-ITI), Charles University, Prague, Czech Republic,**
{balko,kyncl,valtr}@kam.mff.cuni.cz
5 **Institut für Informatik, Freie Universität Berlin, Germany.**
mulzer@inf.fu-berlin.de
6 **Institute of Mathematics, Technische Universität Berlin, Germany.**
scheucher@math.tu-berlin.de

Abstract

We consider the problem of characterizing small geometric graphs whose structure uniquely determines the order type of its vertex set. We describe a set of edges that prevent the order type from changing by continuous movement and identify properties of the resulting graphs.

1 Introduction

Let $S, T \subset \mathbb{R}^2$ be two sets of n labeled points in the plane, not all on a common line. We say that S and T *have the same order type* if there is a bijection $\varphi : S \to T$ such that any triple $(p, q, r) \in S^3$ has the same orientation (clockwise, counterclockwise, or collinear) as the image $(\varphi(p), \varphi(q), \varphi(r)) \in T^3$ [7]. This induces an equivalence relation on planar sets of n points, with a finite number of equivalence classes, the *order types*. For example, the order type of a point set S determines which geometric graphs can be drawn on S without crossings. This makes order types relevant for extremal problems on geometric graphs.

Suppose we have discovered an interesting order type, and we want to illustrate it in a publication. One solution might be to give explicit coordinates of a representative point set S. This is unlikely to satisfy most readers. Thus, we would rather present S as a set of dots in a figure. For some point sets (particularly those with extremal properties), the reader may find it difficult to discern the orientation of an almost collinear point triple. To mend this, we could draw all lines spanned by two points in S. In fact, it suffices to show only the *segments* between the point pairs (the complete geometric graph on S). The orientation of a triple can then be obtained by inspecting the corresponding triangle; see Figure 1(a). In

* A.P. is supported by a Schrödinger fellowship of the Austrian Science Fund (FWF): J-3847-N35. O.A., I.P., and B.V. are supported by the Austrian Science Fund (FWF) grant W1230. M.B., J.K., and P.V. are supported by the grant no. 18-19158S of the Czech Science Foundation (GAČR). M.B. has received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement no. 678765.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

general, our drawing will be rather dense, and we may have trouble following an edge from one point to the next. Some edges, however, are redundant. Without them, we can still "see" the order type of the underlying point set.

We would like to understand which edges are essential for order type representation. To this end we provide a formal definition of this concept, identify a superset of these *non-redundant* edges, and provide a classification and some properties. While these edges prevent changing the order type by moving the points continuously (intuitively justified by the motivation above), we fall short of proving that their structure fully determines the order type.

Definitions. Let S be a set of n labeled points in the plane. A *geometric graph* on S is a graph with vertex set S whose edges are represented as line segments between their endpoints. A geometric graph is thus a drawing of an abstract graph. Two geometric graphs G and H are *topologically equivalent* if there is a homeomorphism of the plane transforming G into H. Each class of this equivalence relation may be described combinatorially by the cyclic orders of the edge segments around vertices and crossings, and by the incidences of vertices, crossings, edge segments, and faces. In the following we will consider topology-preserving deformations. An *ambient isotopy* of the real plane is a continuous map $f : \mathbb{R}^2 \times [0,1] \to \mathbb{R}^2$ such that $f(\cdot, t)$ is a homeomorphism for every $t \in [0,1]$ and $f(\cdot, 0) = \mathrm{Id}$.

▶ **Definition 1.1.** Let G be a geometric graph on a point set S. We say that G *supports* S (or that G is *supporting*) if every ambient isotopy of $\mathbb{R}^2$ that keeps the images of the edges of G straight and preserves topological equivalence to the drawing G also preserves the order type of the vertex set.

Every complete geometric graph is supporting. A supporting graph need not be connected, and two distinct minimal supporting graphs can be drawings of the same abstract graph; see Figure 2 (b,c), and Figure 4. Thus, the structure of the drawing is crucial.

▶ **Definition 1.2.** Let G be a geometric graph on a set S of n points. We say that G *forces* S (or that G is *forcing*) if every n-point set S' that is the vertex set of a geometric graph topologically equivalent to G has the same order type as S.

Clearly, every forcing geometric graph is also supporting.

Related work and outline. The connection between order types and straight-line drawings has been studied intensively, both for planar drawings and for drawings minimizing the number of crossings. For example, it is NP-complete to decide whether a planar graph can be embedded on a given point set [4]. Continuous movements of the vertices of plane geometric graphs have also been considered [1]. The continuous movement of points maintaining the order type was considered by Mnëv [12], who showed that there are point sets with the same order type such that there is no ambient isotopy between them preserving the order type (settling a conjecture by Ringel [13]). The orientations of triples that have to be fixed to determine the order type are strongly related to the concept of *minimal reduced systems* [3].

We describe a notion of *exit edges* for a given point set. Although the resulting *exit graphs* are always supporting, they are not necessarily minimal with this property. One reason is that the topological structure of a geometric graph is not completely determined by the order type of its vertex set, whereas the exit edges are derived solely from the order type. Furthermore, some exit edges are rendered unnecessary by nonstretchability of certain pseudoline arrangements. This concept and the subsequent difficulties are discussed in

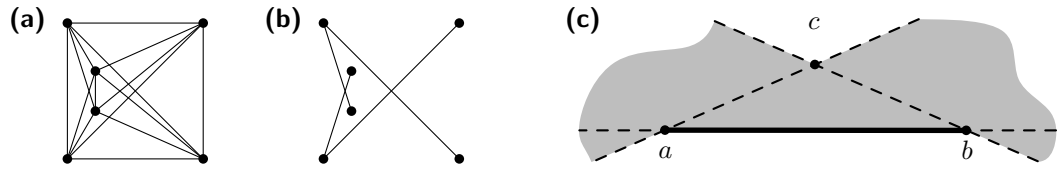


Figure 1 A set of six points with (a) all segments and (b) only exit edges drawn. (c) If the gray region is empty of points, then the edge ab is an exit edge.

Section 2. Despite being non-minimal in general, we argue that exit edges are good candidates for supporting graphs by discussing their dual representation in pseudoline arrangements (Section 3). We provide some further properties in Section 4. We conjecture that graphs based on exit edges are not only supporting but also forcing.

2 Exit edges

To obtain a supporting graph, we select edges so that no vertex of the resulting geometric graph can be moved to change the order type while preserving topological equivalence. In this section we will assume point sets to be in *general position*, that is, with no three collinear points, unless stated otherwise.

▶ **Definition 2.1.** Let S be a finite point set in general position, and let a, b, c be three distinct points from S. We say that ab is an *exit edge* with *witness* c if there is no point $p \in S$ such that the line $\overline{ap}$ separates b from c or the line $\overline{bp}$ separates a from c. The geometric graph on S with edge set formed by the exit edges is called the *exit graph* of S.

Equivalently, ab is an *exit edge* with *witness* c if and only if the double-wedge through a between b and c and the double-wedge through b between a and c contain no point of S in their interior; see Figure 1(c). An exit edge has at most two witnesses. Also, if $|S| \geq 4$ and ab is an exit edge in S with witness c, neither ac nor bc can be an exit edge with witness b or a, respectively. We illustrate the set of exit edges for a set of 6 points in Figure 1(b).

Exit edges can be characterized via 4-holes. For an integer $k \geq 3$, a *k-hole* in a point set S is a simple polygon spanned by k points of S whose interior contains no point of S. A pair ab from S is *extremal* in S if it lies on the boundary of the convex hull of S. A pair of points from S that is not extremal in S is *internal* in S.

▶ **Theorem 2.2.** *The edge ab is not an exit edge of S if and only if the following holds.*

1. *If ab is extremal in S, then it is incident to at least one convex 4-hole in S.*
2. *If ab is internal in S, then it is incident to at least one general 4-hole on each side such that the reflex angle (if any) is incident to ab.*

We remark that an internal exit edge either has a witness on both sides or is incident to at least one general 4-hole on one side. Due to space constraints, the proof of Theorem 2.2 is deferred to the full version of the paper.

▶ **Proposition 2.3.** *Let $S \subset \mathbb{R}^2$ be a finite point set in general position and for every $t \in [0, 1]$, let $S(t)$ be a continuous deformation of S at time t; more formally, let $S(t)$ be the point set $\{f(s, t); s \in S\}$ given by some ambient isotopy $f : \mathbb{R}^2 \times [0, 1] \to \mathbb{R}^2$. Let (a, b, c) be the first triple to become collinear, at time $t_0 > 0$. If c lies on the segment ab in $S(t_0)$, then ab is an exit edge of $S(0)$ with witness c.*

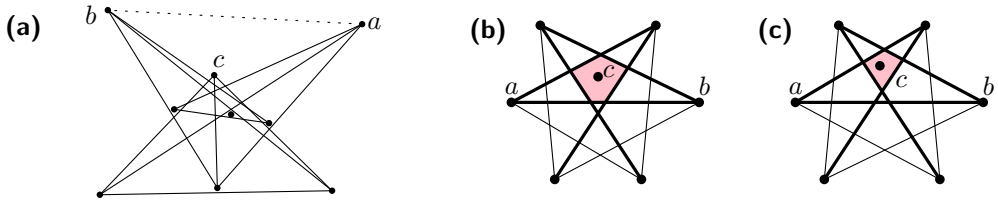


Figure 2 (a) Moving c over ab to make (a, b, c) oriented clockwise, without changing the orientation of other triples, would contradict Pappus's theorem [13]. (The corresponding abstract order type is not realizable.) (b, c) The segment ab is an exit edge with witness c. In (c), we cannot move c continuously to ab without first changing the order type, unless we also move other points.

Proof. For $t \in [0, t_0)$, the triple orientations in $S(t)$ remain unchanged. In $S(t_0)$, the point c lies on ab. Thus, for $t \in [0, t_0)$, there is no line through two points of $S(t)$ that strictly separates the relative interior of ab from c. In particular, there is no such separating line through a or b in $S(0)$. Hence, ab is an exit edge with witness c. ◀

▶ **Corollary 2.4.** *The exit graph of every point set is supporting.*

The proof of Proposition 2.3 also shows that if a line separates c from the relative interior of ab, then there is such a line through a or b. This may suggest that the exit edges are necessary for a supporting graph. However, this is not true in general. For example, in Figure 2(a), we see a construction by Ringel [13]: ab is an exit edge with witness c, but c cannot move over ab without violating Pappus' theorem. There are also point sets where two or more other line segments prevent a witness c from crossing an exit edge ab, for example, see Figure 2(c). In general, this cannot be inferred from the order type of the underlying point set. While c cannot move to ab without changing the order type in Figure 2(c), we could first change the point set to the one in Figure 2(b) and then move c over ab. So ab indeed has to be part of a graph supporting the set. Note that by Definition 1.1, it also prevents the point set to be transformed to the other one.

3 Exit edges and empty triangles

For a point set S in the Euclidean plane, add a line ℓ_∞ to obtain the real projective plane. By taking the projective dual of S and ℓ_∞, we get a projective line arrangement S^* where one cell, the *marked* cell, contains the dual point ℓ_∞^* at vertical infinity. The combinatorial structure of S^*, together with the marked cell, determines the order type of S. Dual to the proof of Proposition 2.3, we continuously move the lines without crossing ℓ_∞^*. The combinatorial structure changes when a line crosses a vertex of S^*. Before that, there is a triangular cell T bounded by three lines, dual to the endpoints of an exit edge and its witness. In S, the witness is the point that is between the other two points when the set becomes collinear. If we project S^* to the Euclidean plane by choosing a *line at infinity* through ℓ_∞^* that does not intersect T, the witness corresponds to the bounding line of T with *median slope*. Alternatively, the witness corresponds to the line containing the leftmost and the rightmost vertex of T.

Hence, the number of *triangles* in a simple projective line arrangement gives an upper bound on the number of exit edges of a point set. One triangle could contain ℓ_∞^*, and there could be pairs of triangles that share a crossing in such a way that leads to only one exit edge for the primal point set. Any projective arrangement of $n \geq 4$ lines has at least n triangles, as each line is incident to at least three triangles [10], which is tight. Therefore, any set of

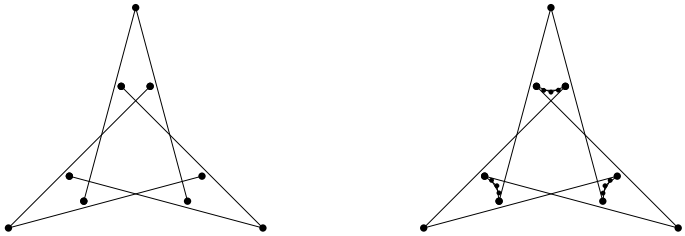


Figure 3 Constructions of point sets with $n-3$ exit edges, currently the smallest known number.

$n \geq 4$ points has at least $\lceil \frac{n-1}{2} \rceil$ exit-edges. A more careful counting of exit edges with one and two witnesses gives a lower bound of $\frac{3n}{5} - O(1)$ for the number of exit edges. The proof can be found in the full version of the paper. This bound is not proven tight, since so far we only know of point sets with $n-3$ exit edges for $n \geq 9$; see Figure 3.

The number of triangles in a simple arrangement is at most $\frac{n(n-1)}{3}$ [8]. Roudneff [14] and Harborth [9] showed that this is also tight. Thus, this is an upper bound on the number of exit edges. Possibly, this upper bound can be improved, as constructions showing tightness of the bound have many pairs of triangles sharing a vertex and corresponding to the same exit edge. However, there are also line arrangements with no such pair of triangles [11]. In the full version of the paper, we adapt a construction from [2] to show that the tight upper bound on the number of supporting edges for n points is in $\Theta(n^2)$.

4 (Counter-)Examples and properties

We present some results on general supporting graphs (and thus on exit graphs).

▶ **Theorem 4.1.** *Any geometric graph supporting a point set S, $|S| \geq 9$, contains a crossing.*

Proof. Let G be a geometric graph with vertex set S without crossings. There is a point set S' with a different order type that also admits G: Dujmović showed that every plane graph admits a plane straight-line embedding with at least $\sqrt{n/2}$ points on a line [5]; as we have a point set with a collinear triple that admits G, there are at least two point sets in general position with a different order type that admit G. Moreover, one can continuously morph S to S' while keeping the corresponding geometric graph planar and topologically equivalent to G (see, for example, [1]). Therefore, G does not support S. ◀

▶ **Proposition 4.2.** *Let S be a point set in general position in $\mathbb{R}^2$ and let G be its exit graph. Every vertex in the unbounded face of G is extremal, that is, it lies on the boundary of the convex hull of S.*

Note that, as shown in Figure 2(a), an analogous statement does not hold for general supporting graphs. The proof of Proposition 4.2 is deferred to the full version of the paper.

So far, we have few results for characterizing graphs that force a point set S, but we conjecture that the graph of exit edges not only supports S, but also forces it. However, even if we are given all the exit edges and their witnesses (in the dual, this means having all triangles of a line arrangement and their orientations), we cannot always infer the order type of S. A counterexample is sketched in Figure 4 as a dual (stretchable) pseudoline arrangement of 14 lines in the projective plane, based on an example by Felsner and Weil [6]. It consists of two arrangements of six lines in the Euclidean plane that are combinatorially different, but share the set of triangles and their orientations. While the exit edges are the same for the two order types, the corresponding exit graphs are not topologically equivalent.

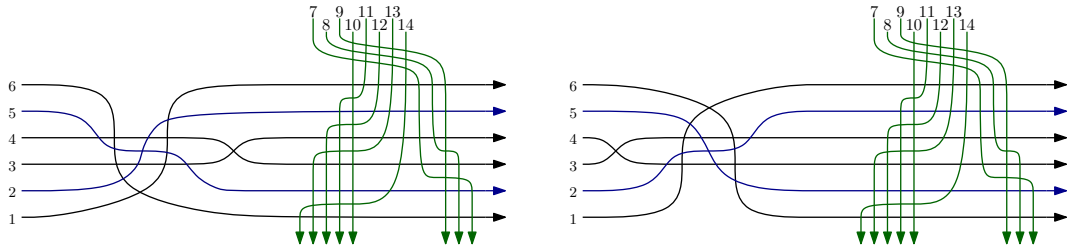


Figure 4 Two arrangements of 14 pseudolines with the same set of triangles (extending [6, Figure 3]). The green arrangements are the same. There is no triangle crossed by the line at infinity.

Acknowledgments. This work was initiated during the *Workshop on Sidedness Queries*, October 2015, Ratsch, Austria. We thank Thomas Hackl, Vincent Kusters, and Pedro Ramos for valuable discussions.

References

1 Soroush Alamdari, Patrizio Angelini, Fidel Barrera-Cruz, Timothy M. Chan, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Penny Haxell, Anna Lubiw, Maurizio Patrignani, Vincenzo Roselli, Sahil Singla, and Bryan T. Wilkinson. How to morph planar graph drawings. *SIAM J. Comput.*, 46(2):824–852, 2017.

2 Martin Balko, Josef Cibulka, and Pavel Valtr. Drawing Graphs Using a Small Number of Obstacles. *Discrete Comput. Geom.*, 59(1):143–164, 2018.

3 Jürgen Bokowski and Bernd Sturmfels. On the coordinatization of oriented matroids. *Discrete Comput. Geom.*, 1:293–306, 1986.

4 Sergio Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *J. Graph Algorithms Appl.*, 10(2):353–363, 2006.

5 Vida Dujmović. The utility of untangling. *J. Graph Algorithms Appl.*, 21(1):121–134, 2017.

6 Stefan Felsner and Helmut Weil. A theorem on higher Bruhat orders. *Discrete Comput. Geom.*, 23(1):121–127, 2000.

7 Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM J. Comput.*, 12(3):484–507, 1983.

8 Branko Grünbaum. *Arrangements and spreads.* AMS, Providence, 1972.

9 Heiko Harborth. Some simple arrangements of pseudolines with a maximum number of triangles. In *Discrete geometry and convexity (New York, 1982)*, volume 440 of *Ann. New York Acad. Sci.*, pages 31–33. New York Acad. Sci., New York, 1985.

10 Friedrich Levi. Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade. *Ber. Math.-Phys. Kl. Sächs. Akad. Wiss. Leipzig*, 78:256–267, 1926. In German.

11 Dragoslav Ljubić, Jean-Pierre Roudneff, and Bernd Sturmfels. Arrangements of lines and pseudolines without adjacent triangles. *J. Combin. Theory. Ser. A*, 50(1):24–32, 1989.

12 Nicolai E. Mnëv. The universality theorems on the classification problem of configuration varieties and convex polytope varieties. In *Topology and Geometry—Rohlin Seminar*, volume 1346 of *Lecture Notes in Math.*, pages 527–544. Springer, 1988.

13 Gerhard Ringel. Teilungen der Ebene durch Geraden oder topologische Geraden. *Math. Z.*, 64:79–102 (1956), 1955.

14 Jean-Pierre Roudneff. On the number of triangles in simple arrangements of pseudolines in the real projective plane. *Discrete Math.*, 60:243–251, 1986.

Beam It Up, Scotty: Angular Freeze-Tag with Directional Antennas*

Sándor P. Fekete[1] and Dominik Krupke[1]

1 **TU Braunschweig**
{s.fekete,d.krupke}@tu-bs.de

Abstract

We consider distributing mission data among the members of a satellite swarm. In this process, spacecraft cannot be reached all at once by a single broadcast, because transmission requires the use of highly focused directional antennas. As a consequence, a spacecraft can transmit data to another satellite only if its antenna is aiming right at the recipient; this may require adjusting the orientation of the transmitter, incurring a time cost proportional to the required angle of rotation. The task is to minimize the total distribution time. This makes the problem similar in nature to the *Freeze-Tag Problem* of waking up a set of sleeping robots, but with angular cost at vertices, instead of distance cost along the edges of a graph. We prove that approximating the minimum length of a schedule for this *Angular Free-Tag Problem* within a factor of less than 5/3 is NP-complete, and provide a 9-approximation for the 2-dimensional case that works even in online settings with incomplete information. Furthermore, we develop an exact method based on Mixed Integer Programming that works in arbitrary dimensions and can compute provably optimal solutions for benchmark instances with about a dozen satellites.

1 Introduction

Providing instructions to all members of a distributed group is a fundamental task for many types of team missions. In terrestrial settings, this can usually be achieved by broadcasting to all recipients in parallel, requiring only a single transmission. However, for long-distance space missions, omnidirectional transmission can no longer be employed, due to significant loss in signal strength. Instead, transferring data is accomplished with the help of directional antennas, requiring a highly focused communication beam that is targeted right at the intended recipient. (See Figure 1 for an illustration.) As a consequence, these transmissions must be performed individually, involving maneuvers for achieving appropriate antenna orientation; the time for such a maneuver is basically proportional to the required angle of rotation, with negligible time for the actual transmission itself. The overall process does allow one parallel component: a team member that has already been "activated" by having received the data may relay this to other partners, motivating the use of intricate communication trees for achieving rapid dissemination of information to all members of a swarm of spacecraft.

This can be utilized if we want to quickly distribute data, e.g., an important update. In the following we consider a basic version of the problem in which the agents are static points in the euclidean space, there are no delays for transmission, and the transmission cone is modeled as a ray. (Also note that more advanced scenarios for space missions may require *both* a transmitting and a receiving antenna that are directed at the communication partner; see the Conclusions in Section 5.)

* Partially supported by the European Space Agency, project ASIMOV, contract number 4000122514/17/F/MOS.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

▶ Problem 1.1. **Angular Freeze Tag (AFT).** Given a set $P = \{p_0, \ldots, p_n\}$ of agent positions in d-dimensional space. Each agent $p_i \in P$ has an initial heading α_i. At time $t = 0$, only p_0 is *active*, while all other agents are *inactive*. An agent p_i is *activated* by an active agent p_j whose heading α_j aims right at p_i; adjusting this heading incurs a cost equal to the required angular change. The objective is to minimize the time T until all agents are activated, i.e., minimize the makespan of the overall activation schedule.

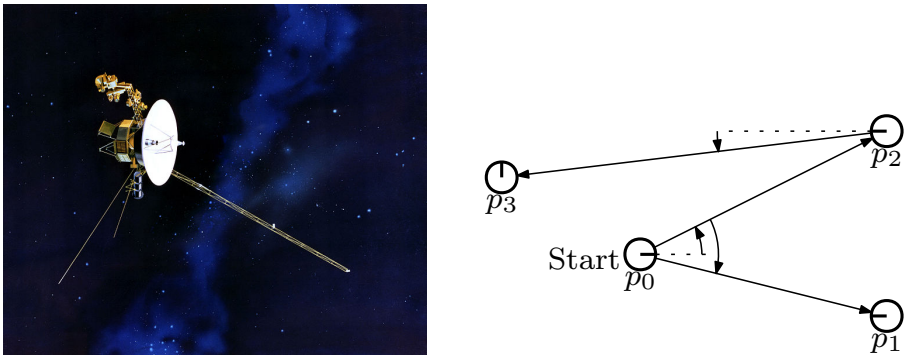


Figure 1 (Left) The space probe Voyager and its directional antenna for transmitting data. (Image CC by NASA.) (Right) Activating all agents by rotations: p_0 first activates p_2 which then activates p_3 while p_0 rotates back to activate p_1.

Related Work. The original *Freeze-Tag Problem* (FTP) was introduced by Arkin et al. [2], who studied the task of waking up a swarm of robots. In the FTP, activating an inactive robot is performed by moving an active robot next to it. The objective (minimize the makespan of the overall schedule) is the same as for our problem, but the cost for an activation (the distance to the robot instead of the angle) is different. This problem is NP-hard even for star graphs, but there are polynomial-time approximation schemes (PTAS) for star graphs and geometrically embedded instances [3]. Unweighted graphs are considered in [4]. A set of heuristics is evaluated in [11]. Results on the hardness in Euclidean space are provided by [1] and [9].

Other geometric questions related to the use of directional antennas have also been considered. Carmi et al. [8] studied the α-MST, which arose from finding orientations of directional antennas with α-cones, such that the connectivity graph yields an MST of minimum weight, based on bidirectional communication. They prove that for $\alpha < \pi/3$, a solution may not exist, while $\alpha \geq \pi/3$ always suffices. See Aschner and Katz [5] for more recent hardness proofs and constant-factor approximations for some α.

2 Hardness of Approximation

We show that the AFT is computationally hard, even to approximate.

▶ **Theorem 2.1.** *A c-approximation algorithm for the AFT with $c < 5/3$ implies $P = NP$.*

Proof. We give a reduction from Satisfiability; see Figure 2 for a sketch. Our construction has a solution with a makespan of 3ε if it is satisfiable and 5ε otherwise, where $\varepsilon > 0$ is a sufficiently small angle. Our construction uses five different types of agents, as follows.

- The *start agent* p_0 directly activates the *decision agents*, but does not have any other agents within 5ε of α_i.

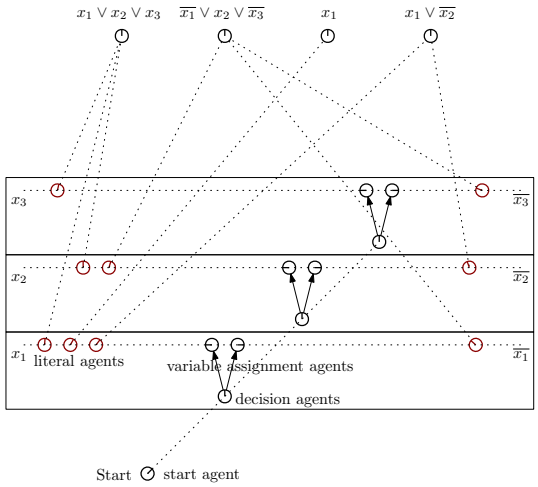


Figure 2 Sketch of the hardness construction. Red variable agents are 2ε from their designated heading, which they can target upon activation. The decision agent for each variable is a rotation ε from both of its corresponding variable assignment agents. A schedule of makespan 3ε exists if and only if there is a satisfying truth assignment; otherwise, the makespan is at least 5ε.

- For each variable we have a *decision agent* and two *variable assignment agents* (one each for `true` and `false`) in opposing angles of ε, but no further agents within a 5ε rotational range. It is directly activated from p_0.
- The *variable assignment agent* directly activates all corresponding *literal agents*, but has no further agents in a 4ε rotation range. The earliest possible activation time is ε. Only one of the two agents can be activated at time ε (by the *decision agent*), the other one has to wait an additional 2ε.
- For each literal there is a *literal agent* that has its clause agent a rotation of 2ε away, but no further agents within 4ε. The earliest possible activation time is ε.
- For each clause there is a *clause agent* that has no agent within its 2ε rotation range. Its earliest possible activation time is 3ε.

A clause agent can only be activated by its literal agents in less than 5ε and a literal agent is either activated at ε or 3ε, depending on which of the variable assignment agents got activated first. Thus, a clause is activated at 3ε if and only if a corresponding variable agent has been activated in time; otherwise, it takes 5ε. ◀

3 Approximation Algorithm

We can provide a simple constant factor approximation, based on a result by Beck [6] on the *linear search problem*. In that scenario, an agent has to locate a hidden object in a one-dimensional environment; from a given starting location, the best strategy for this online problem is to alternate between going left and right, while doubling the search depth in each iteration. This yields a total search distance that is within a factor of 9 of the optimum.

▶ **Theorem 3.1.** *There is a 9-approximation algorithm for the AFT in 2-dimensional space, even for unknown agent locations and headings, assuming a lower bound of $\varepsilon > 0$ for the rotational angle of any activating agent.*

Proof. As soon as an agent is activated, it follows the doubling strategy from linear search, carried out for rotation. It follows straightforward by induction that any agent p_i that gets

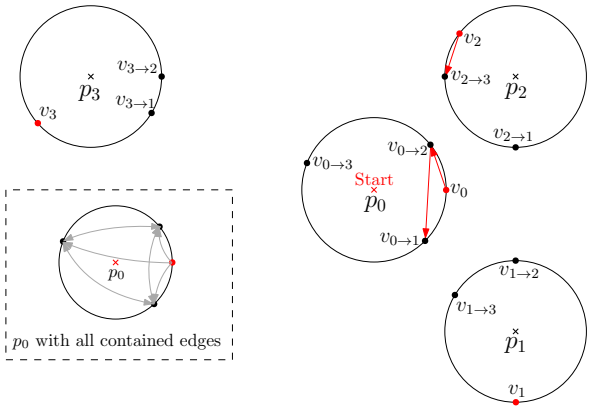


Figure 3 An example of the auxiliary graph. Every point (p_0, p_1, p_2, p_3) has a vertex for its initial heading (v_i) and a vertex for the heading to any other point different from p_0. Between the vertices of the same point, there are directed edges with the cost of the corresponding rotation; as shown in the lower left, there are no incoming edges for the start vertex. If points are collinear, there can be two vertices for the same heading. A possible solution would be for p_0 to first head to p_2 and then to p_1, while p_2 heads to p_3. The corresponding movements are visualized by red edges.

activated by T_i in an optimal schedule is activated within $9T_i$. ◀

Note that we cannot apply the refined technique by Bose et al. [7] for linear search, as it requires both an upper and a lower bound on the search distance.

4 Exact Solution

In the following, we describe the set of solutions by a Mixed Integer Program (MIP). This allows us to use an advanced solver such as CPLEX to obtain provably optimal solutions.

Each agent has only a finite set of relevant headings; between such two configurations there is an easily computable optimal rotation. The relevant configurations are the initial heading of an agent and the headings that activate other agents, for a total of $O(|P|)$ configurations and $O(|P|^2)$ transitions per agent. We can encode this into an auxiliary directed graph $G = (V, E)$ in which the configurations are the vertices and the vertices of each agent form a weakly connected component. For an agent $p_i \in P$ we denote the initial heading vertex by v_i, and the vertices that activate another agent $p_j \in P$ by $v_{i\to j}$. There is a directed edge between all vertices $v_{i\to j}, p_j \in P \setminus \{p_i, p_0\}$ as well as from v_i to all $v_{i\to j}, p_j \in P \setminus \{p_i, p_0\}$. There are no edges between the vertices of different agents. The movement (and agent activations) of an agent p_i can be represented by a directed path starting at v_i. Figure 3 visualizes such a graph and how to encode a solution.

We use *Boolean variables* $x_e, e \in E$ that represent the transition of an agent between two configurations, and *continuous variables* $y_v, v \in V$ that represent the time at which an agent reaches a specific configuration. If the configuration is not used, it may be zero. The value needs only to be tight for configurations that are critical for the makespan.

The general idea of the Mixed Integer Program is simple: the usage of an edge implies that the target's time has to be the source's time plus the transition time; we want to minimize the maximum value. It is also fairly simple to adapt this MIP to other problem variants. Let us start with the objective function that minimizes the latest activation time

$$\min \max_{p_i \in P} y_{v_i}. \tag{1}$$

Note that we need to implement the min-max via $\min t$ and $t \geq y_{v_i} \forall p_i \in P$, resulting in $O(|P|)$ additional constraints. For every agent we need to visit a vertex that activates it, i.e., we need to use an edge that visits such a vertex (exactly one to be precise).

$$\sum_{e \in E_{\text{in}}(v_{j \to i}), p_j \in P} x_e = 1 \quad \forall p_i \in P \setminus \{p_0\}. \tag{2}$$

Next we enforce that there are only directed paths starting at initial heading vertices by enforcing that there can only be at most one outgoing edge per vertex and only if there is also an ingoing one (3) or it is a start vertex (4), and prohibiting subcycles (5).

$$\sum_{E_{\text{out}}(v_{i \to j})} x_e \leq \sum_{E_{\text{in}}(v_{i \to j})} x_e \leq 1 \qquad \forall v_{i \to j} \in V \tag{3}$$

$$\sum_{E_{\text{out}}(v_i)} x_e \leq 1 \qquad \forall p_i \in P \tag{4}$$

$$\sum_{v,w \in S} x_{vw} \leq |S| - 1 \qquad \forall S \subset V \tag{5}$$

If agent p_i is activated by agent p_j, then $y_{v_i} = y_{v_{j \to i}}$. Since $y_{v_{k \to i}} = 0$ for all other agents p_k, we can write

$$y_{v_i} = \sum_{p_j \in P} y_{v_{j \to i}} \quad \forall p_i \in P \setminus \{p_0\}. \tag{6}$$

If we use a directed edge, we know that the target has to have the time of the source plus the minimal transition time, i.e., for an edge $vw \in E : y_w \geq y_v + \text{cost}(vw)$. We can neutralize this constraint by adding a large negative value to the right side that lowers it below zero if the edge is not selected. This value only needs to be 3π, because no optimal solution is larger than 2π and an edge cost is at most π.

$$y_w \geq y_v + \text{cost}(vw) + (3\pi x_{vw} - 3\pi) \quad \forall vw \in E \tag{7}$$

This constraint also prevents cyclic activations or cycles as in constraint (5) as long as they are not based on zero-cost transitions (this works analogous to the Miller-Tucker-Zemlin subtour elimination constraints for TSP [10]). To also prevent zero-cost cyclic activations we can use the following constraint:

$$\sum_{p_i, p_j \in S} \sum_{e \in E_{\text{in}}(v_{i \to j})} x_e \leq |S| - 1 \quad \forall S \subset P \setminus \{p_0\}. \tag{8}$$

Because this only happens for degenerated cases with zero-cost edges, we add the constraints (5) and (8) iteratively only if necessary.

In the end we have $\Theta(|P|^2)$ continuous variables, $\Theta(|P|^3)$ Boolean variables (of which only $|P|-1$ variables will be `true`), and $\Theta(|P|^3)$ constraints (excluding (5) and (8)), resulting in a relatively large problem that also becomes very quickly hard to solve, as can be seen in Fig. 4. Interestingly, this is not because CPLEX does not find a solution, but because it does not find an effective lower bound. Code on `https://github.com/d-krupke/eurocg18-angularft`.

5 Conclusion

We provided first results for a basic version of Angular Freeze Tag. Even in 2D with static transmitters, we need better lower bounds to improve approximation and the size of optimally solvable instances. There is also a wide spectrum of practically important generalizations.

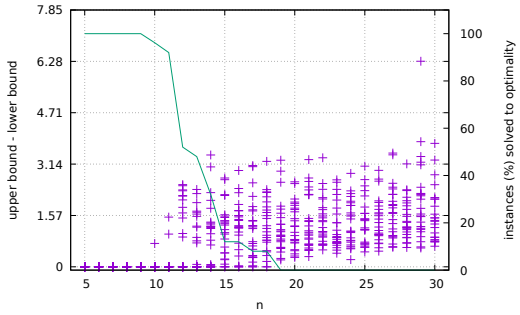


Figure 4 Results for random instances with CPLEX and a 15 min time limit on a PC (i7, 64GB). For 12 points only 50% can be solved to optimality. For unsolved instances, the lower bound is often close to zero, so providing better lower bounds will drastically improve performance.

These include approximation for the three-dimensional version and scenarios with moving satellites. Allowing inactive receivers to adjust their heading ahead of time may greatly speed up schedules. On the other hand, advanced missions may require *both* partners in a data exchange to have their directional antennas pointing at each other, making the scheduling process considerably more involved. All these issues are left for future work.

References

1 Z. Abel, H. A. Akitaya, and J. Yu. Freeze tag awakening in 2D is NP-hard. In *Proc. 27th Fall Worksh. Comp. Geom. (FWCG)*, 2017. Paper 28.

2 E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. Mitchell, and M. Skutella. The freeze-tag problem: How to wake up a swarm of robots. In *Proc. 13th Symp. Disc. Alg. (SODA)*, pages 568–577, 2002.

3 E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. Mitchell, and M. Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46(2):193–221, 2006.

4 E. M. Arkin, M. A. Bender, and D. Ge. Improved approximation algorithms for the freeze-tag problem. In *Proc. 15th Symp. Par. Alg. Arch. (SPAA)*, pages 295–303, 2003.

5 R. Aschner and M. J. Katz. Bounded-angle spanning tree: modeling networks with angular constraints. *Algorithmica*, 77(2):349–373, 2017.

6 A. Beck and D. J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8(4):419–429, 1970.

7 P. Bose, J. D. Carufel, and S. Durocher. Searching on a line: A complete characterization of the optimal solution. *Theoretical Computer Science*, 569:24–42, 2015.

8 P. Carmi, M. J. Katz, Z. Lotker, and A. Rosén. Connectivity guarantees for wireless networks with directional antennas. *Computational Geometry*, 44(9):477–485, 2011.

9 M. P. Johnson. Easier hardness for 3D freeze-tag. In *Proc. 27th Fall Worksh. Comp. Geom. (FWCG)*, 2017. Paper 29.

10 C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, 1960.

11 M. O. Sztainberg, E. M. Arkin, M. A. Bender, and J. S. Mitchell. Analysis of heuristics for the freeze-tag problem. In *Proc. 8th Scand. Worksh. Alg. Theo. (SWAT)*, pages 270–279, 2002.

Computing Crossing-Free Configurations with Minimum Bottleneck*

Sándor P. Fekete[1] and Phillip Keldenich[1]

1 **Department of Computer Science, TU Braunschweig, Germany**
{s.fekete,p.keldenich}@tu-bs.de

Abstract

We consider problems of finding non-crossing *bottleneck* structures for a given planar point set: For a given a set of vertices V, the problem Minimum Bottleneck Polygon (MBP) is to find a simple polygon P with vertex set V whose longest edge is as short as possible; the problem Minimum Bottleneck Simple Matching (MBSM) is to find a crossing-free matching of V whose longest edge is as short as possible. Both problems are known to be NP-complete and neither admits a PTAS. We develop exact methods that can solve benchmark instances (newly generated and from the classic TSPLIB library) with up to 1,500 points for MBP and up to 20,000 points for MBSM to provable optimality.

1 Introduction

Finding a simple polygon with a given set V of vertices in the plane is one of the basic problems of computational geometry. If we want to minimize the overall length, this is is equivalent to the classic Traveling Salesman Problem (TSP), as a shortest tour is always non-crossing. However, if the objective is to minimize the length of the longest edge, this is no longer the case, see Fig. 1. This problem Minimum Bottleneck Polygon (MBP) is NP-complete and unless P=NP, it cannot be approximated within a factor better than $\sqrt{3}$, as it is NP-complete to decide whether a hexagonal grid graph has a Hamiltonian cycle (HC) of unit edges (see Arkin et al. [5]). We are not aware of any constant-factor approximation algorithms for the MBP.

A similarly basic geometric optimization problem is to find a matching for a given vertex set. When minimizing the total length of all edges, an optimal solution must also be non-crossing; this allows it to use standard matching techniques, subject to the (purely theoretical) issue of computing the sum of a set of square roots. Matching techniques can also be used to compute a Minimum Bottleneck Matching (MBM) in polynomial time. However, a solution to MBM does not have to be non-crossing, as shown in Fig. 1. In fact, it was shown by Abu-Affash et al. [2] that this problem Minimum Bottleneck Simple Matching (MBSM) is NP-complete and does not allow a PTAS. They also provide a $2\sqrt{10} \approx 6.325$-approximation algorithm and state without proof that they can reduce this factor to $(1+\sqrt{2})\sqrt{5} \approx 5.398$.

In this paper, we develop methods for computing provably optimal solutions for benchmark instances up to 2,000 points. Beyond illustrating the practical solvability of both problems, this will provide ground truth for testing potential (improved) approximation methods.

Related work. There is a huge body of related work; due to limited space, we only mention a small subset.

* This work was partially supported by the DFG Research Unit "Controlling Concurrent Change", funding number FOR 1800, project FE407/17-2, "Conflict Resolution and Optimization".

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

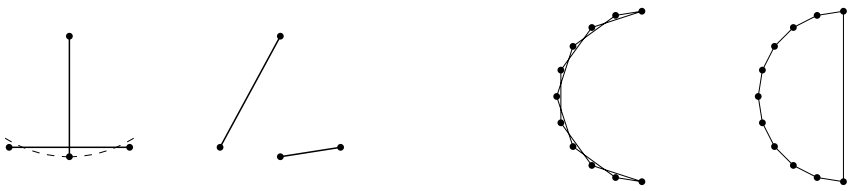


Figure 1 Left: A minimum bottleneck matching and a crossing-free minimum bottleneck matching. Right: A minimum bottleneck tour and a minimum bottleneck polygon.

The minimum bottleneck TSP was first introduced by Gilmore and Gomory [9]. For metric instances, there is a 2-approximation algorithm implied by Fleischner's theorem [8, 12] which states that the square of every two-connected graph is Hamiltonian; for general metric instances, this factor is best possible. Hochbaum and Shmoys [10] also prove a factor of 2 within a framework providing approximation algorithms for several bottleneck problems. The problem of finding a *longest* simple polygon for a given vertex set was considered by Alon et al. [3]; they conjecture this problem to be NP-hard, but this is still open. See Dumitrescu and Tóth [6] for improved approximation factors.

2 Minimum Bottleneck Polygonalization

2.1 Modeling

We start with a basic formulation NMBP(V) of Minimum Bottleneck Polygon as a Mixed Integer Program, where x_{pq} is a Boolean variable encoding whether pq is an edge of the polygon and B encodes the bottleneck of the solution. For two points $p, q \in V$, let $\chi(pq)$ be the set of line segments crossing pq.

$$\min B \text{ s. t.}$$

$$\forall p \in V : \sum_{q \neq p} x_{pq} = 2 \tag{1}$$

$$\forall p, q \in V, rs \in \chi(pq) : x_{pq} + x_{rs} \leq 1 \tag{2}$$

$$\forall \emptyset \subsetneq S \subsetneq V : \sum_{p \in S, q \in V \setminus S} x_{pq} \geq 2 \tag{3}$$

$$\forall p, q \in V : \|pq\|_2 \cdot x_{pq} \leq B \tag{4}$$

$$x_{pq} \in \{0, 1\}$$

Degree constraints (1) ensure two incident edges for each point, while *crossing constraints* (2) exclude crossing edges. The *subtour constraints* (3) enforce a connected solution. Finally, the *bottleneck constraints* (4) enforce the maximum edge length B.

This naïve formulation is only practical for small point sets. Firstly, it is well known that using an auxiliary variable B to encode a min max-type objective often induces weak LP relaxations and thus leads to a suboptimally large search tree. Moreover, the total number of crossing constraints corresponds to the number of convex quadruples in V, which is known to be $\Omega(n^4)$ (see [11, 14]), so using all these constraints at once becomes prohibitively expensive.

In the following, we present a formulation of the MBP as a sequence BMBP$_\tau(V)$ of IPs addressing these issues; this resembles the approach used in [7] for determining the threshold value for a triangulation whose shortest edge is as long as possible. For a threshold value τ, BMBP$_\tau(V)$ is integer feasible iff V has a polygon with bottleneck at most τ; we exclude all edges longer than τ. As before, we use degree constraints (5), subtour constraints (7) and

crossing constraints (6). Thus, a minimum bottleneck polygon can be found with binary search over possible values of τ. The optimal bottleneck is the length of an edge; therefore, this is a discrete set of possible values.

$$\min \sum_{p,q \in V, \|pq\|_2 \leq \tau} \|pq\|_2 \cdot x_{pq} \text{ s. t.}$$

$$\forall p \in V : \sum_{q \in V, \|pq\|_2 \leq \tau} x_{pq} = 2 \tag{5}$$

$$\forall p, q \in V, \|pq\|_2 \leq \tau, rs \in \chi(pq) : x_{pq} + x_{rs} \leq 1 \tag{6}$$

$$\forall \emptyset \subsetneq S \subsetneq V : \sum_{p \in S, q \notin S, \|pq\|_2 \leq \tau} x_{pq} \geq 2 \tag{7}$$

$$x_{pq} \in \{0, 1\}$$

In our implementation of this formulation, only violated crossing and subtour constraints are added iteratively by generating appropriate cutting planes. This results in only small subsets of these large families actually being used. This is aided by our use of the objective function. Instead of directly minimizing the bottleneck, we minimize the sum of all edge lengths. Due to the triangle inequality, most avoidable edge crossings never occur in intermediate (fractional and integral) solutions. Moreover, we do not have to solve $\text{BMBP}_\tau(V)$ to optimality; we can abort the search as soon as the first integer feasible solution is found.

2.2 Computational Results

We implemented $\text{NMBP}(V)$ and $\text{BMBP}_\tau(V)$ in C++, using IBM ILOG CPLEX 12.6.2 as our IP solver. All our experiments ran on a workstation running Linux 4.4 on an Intel Core i7-6700K CPU at 4 GHz clock frequency with 64 GiB of RAM. In order to efficiently construct $\text{BMBP}_\tau(V)$, we used an implementation of *kd*-trees provided by CGAL [1] to enumerate all points within distance τ of a query point. Moreover, violated crossing constraints are detected using CGAL's sweep line implementation. To compare the performance of NMBP and BMBP, we ran both NMBP and BMBP on a set of small instances. Using NMBP, most instances with more than 50 points cannot be solved within 500 seconds, while BMBP solves these instances in less than half a second. Thus, we only evaluate BMBP in the remainder of the section. Fig. 2 shows the results of running BMBP on randomly generated point sets with a modest time limit of 10 minutes. We also ran BMBP on all geometric instances from the classic TSPLIB [13] with fewer than 2,500 points. Within a time limit of one hour, we were able to solve most of them to optimality; see Fig. 3.

3 Minimum Bottleneck Matching

3.1 Modeling

By modifying the right-hand side of the degree constraints (1) and (5) to 1 and removing the subtour constraints (3) and (7), we obtain a formulation of the crossing-free minimum bottleneck matching problem as naïve MILP $\text{NMBM}(V)$ and as sequence of IPs $\text{BMBM}_\tau(V)$. In order to improve its performance, we generate blossom constraints

$$\forall S \subsetneq V, |S| \text{ odd}: \sum_{p \in S, q \notin S, \|pq\|_2 \leq \tau} x_{pq} \geq 1, \tag{8}$$

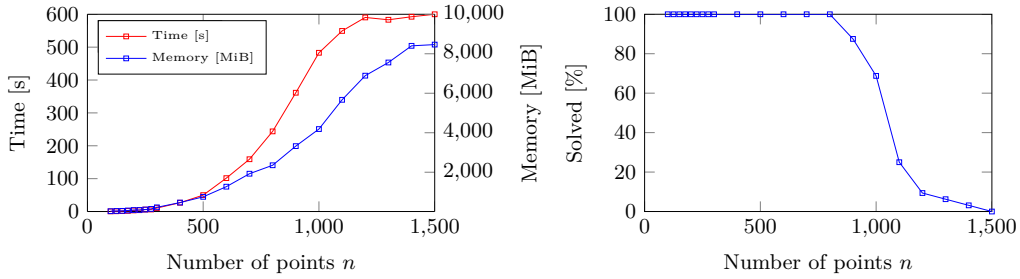


Figure 2 Left: Average running time and peak memory usage for BMBP, run with a time limit of 600 s on point sets generated uniformly at random. Right: Percentage of instances solved within the time limit.

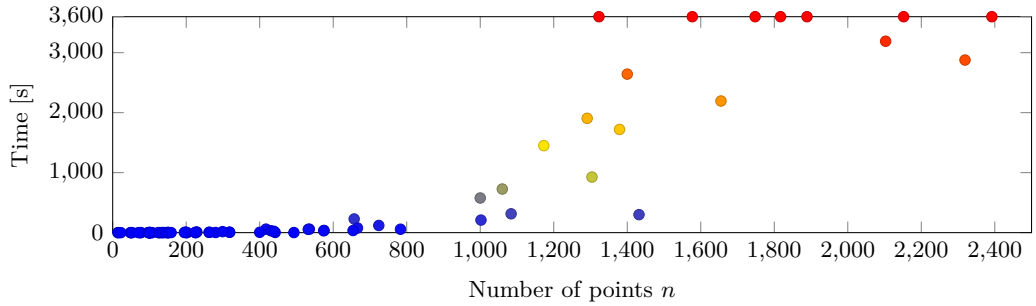


Figure 3 Time required by BMBP to solve the TSPLIB instances; computation was aborted after one hour.

as cutting planes. We identify violated blossom inequalities by searching for odd components in the support graph of a fractional solution. In order to further restrict the search space for the binary search, we implemented a minimum bottleneck matching algorithm to serve as a lower bound and a crossing removal heuristic to produce an initial crossing-free solution as an upper bound; see Section 3.2.

3.2 Crossing Repair Heuristic

A straightforward way to heuristically turn a crossing matching into a non-crossing one is to use a sequence of local 2-OPT exchanges, replacing a crossing pair of edges pq, rs by pr, sq or ps, qr. Any 2-OPT step decreases the sum of edge lengths, so this process must terminate with a non-crossing matching. However, this heuristic does not seem to perform well with respect to the bottleneck.

An alternative is a simple but effective heuristic for converting a crossing matching M_C with bottleneck B into a non-crossing matching M_{NC}, while trying to keep the bottleneck edge as short as possible. We use a standard sweep line algorithm to detect crossings. If there is no more crossing, we are done. Otherwise, we pick an arbitrary crossing pq, rs. Using a *kd*-tree, we perform a simultaneous incremental nearest-neighbor search, starting from p, q, r and s, constructing a set of close points N that contains up to K points, for some constant K; we use $K = 50$ in our experiments. Whenever a new point is discovered, it is added to N, together with its matching partner in M_C. Once N contains K points, we compute the bounding box of N and extend it by B in every direction. We use a range query

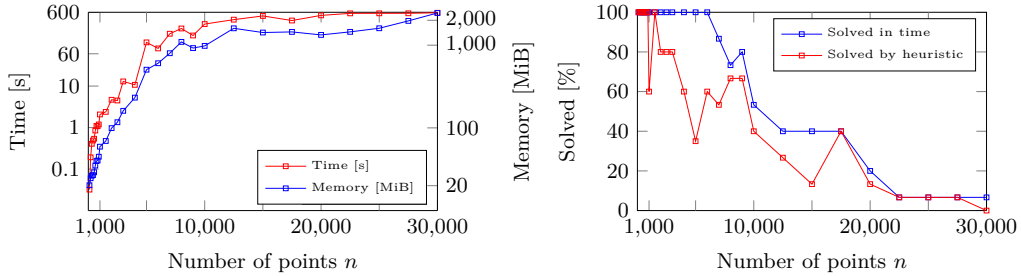


Figure 4 Left: Average running time and peak memory usage for BMBM, run with a time limit of 600 s on point sets generated uniformly at random. Right: Percentage of instances solved within the time limit, and percentage of instances solved to optimality using the crossing-removal heuristic.

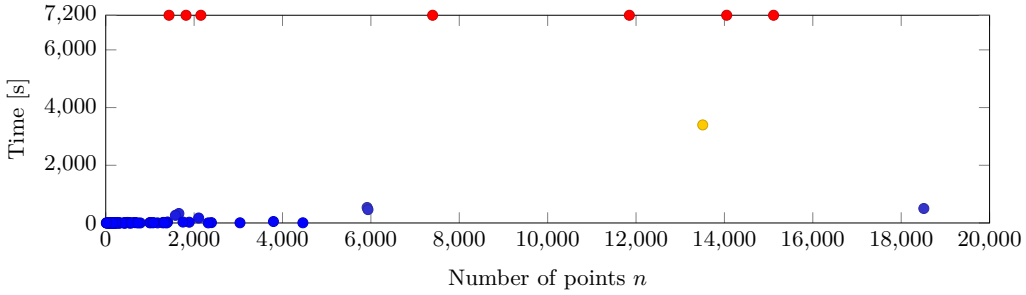


Figure 5 Time required by BMBM to solve the TSBLIB instances; the time limit was two hours.

to find all points inside the extended bounding box; this set of points consists of our *internal* points N and *external* points T. For all external points, we find the corresponding matching edge in M_C; this gives us a set of edges E_T. We use $\text{BMBM}_\tau(N)$ with binary search on τ to find a minimum bottleneck crossing-free matching on N; however, in order to avoid introducing new crossings, we prohibit using any edge that crosses an edge of E_T, unless this edge is part of M_C. In M_C, we replace the matching edges corresponding to points in N with the edges from the resulting matching. In this way, the crossing pq, rs disappears and no new crossings can appear. We iterate this procedure until there are no more crossings; the number of crossings is reduced by at least one in each iteration, thus the heuristic terminates with a crossing-free matching M_{NC}. In certain situations, this heuristic can fail, because a crossing-free matching cannot be found due to the forbidden edges. In this case, we resort to performing 2-OPT steps to remove some crossings before continuing to use the original heuristic.

3.3 Computational Results

We implemented both NMBM and BMBM and evaluated them under the same circumstances as outlined in Section 2.2. Similar to the situation for polygons, the naïve NMBM cannot compete with BMBM, so we give only give computational results BMBM. We were able to solve almost all TSPLIB instances with up to 20,000 points within a time limit of two hours (see Fig. 5). For point sets chosen uniformly at random from the unit square, we were able to solve all generated instances with up to 6,000 points and most instances with up to 10,000 points within ten minutes (see Fig. 4).

In many instances, applying our crossing removal heuristic to a minimum bottleneck matching yields a crossing-free solution with the same bottleneck (see Fig. 4), thus resulting in a provably optimal solution. For all randomly generated instances, the minimum bottleneck

was achievable in a crossing-free manner; on these instances, our crossing repair heuristic was off by a factor of at most 2.215 (this factor was 1.11 on average with median 1.024).

4 Future Work

We presented exact approaches for both the MBP and the MBSM. Many interesting theoretical and practical problems remain that are left for future work.

The most interesting theoretical problem is to develop a constant-factor approximation algorithm for MBP. In our experiments, we found that for large point sets generated uniformly at random, a minimum bottleneck matching can always be achieved with a non-crossing solution. Is there an analytic basis for this observation? For general point sets, it may be interesting to explore the properties of the matching polytope with added crossing constraints.

On the practical side, the quadratic number of edge variables is the biggest impediment for solving larger instances. For the classic TSP, this has been dealt with by using column generation and related methods [4]. For MBP, there are potentially many additional constraints that could be used for cutting plane generation. Doing this in an efficient manner requires rewriting large parts of the integer programming solver. For MBSM, more sophisticated algorithms may be able to identify more (helpful) blossom constraints.

References

1 The Computational Geometry Algorithms Library. `http://www.cgal.org`.

2 A. K. Abu-Affash, P. Carmi, M. J. Katz, and Y. Trabelsi. Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47(3):447–457, 2014.

3 N. Alon, S. Rajagopalan, and S. Suri. Long non-crossing configurations in the plane. *Fundam. Inform.*, 22(4):385–394, 1995.

4 D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A computational study*. Princeton university press, 2011.

5 E. M. Arkin, S. P. Fekete, K. Islam, H. Meijer, J. S. Mitchell, Y. Núñez-Rodríguez, V. Polishchuk, D. Rappaport, and H. Xiao. Not being (super)thin or solid is hard: A study of grid Hamiltonicity. *Computational Geometry*, 42(6–7):582–605, 2009.

6 A. Dumitrescu and C. D. Tóth. Long non-crossing configurations in the plane. *Discrete & Computational Geometry*, 44(4):727–752, 2010.

7 S. P. Fekete, W. Hellmann, M. Hemmer, A. Schmidt, and J. Troegel. Computing MaxMin edge length triangulations. In *Proc. 17th Worksh. Alg. Eng. Exp. (ALENEX)*, pages 55–69, 2015. Full version to appear in *Journal of Computational Geometry*.

8 H. Fleischner. The square of every two-connected graph is Hamiltonian. *Journal of Combinatorial Theory, Series B*, 16(1):29–34, 1974.

9 P. C. Gilmore and R. E. Gomory. Sequencing a one state-variable machine: A solvable case of the Traveling Salesman Problem. *Operations Research*, 12(5):655–679, 1964.

10 D. S. Hochbaum and D. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33:533–550, 07 1986.

11 L. Lovász, K. Vesztergombi, U. Wagner, and E. Welzl. Convex quadrilaterals and k-sets. In *Contemporary Mathematics Series, 342, AMS 2004*, pages 139–148, 2004.

12 R. G. Parker and R. L. Rardin. Guaranteed performance heuristics for the bottleneck Travelling Salesman Problem. *Operations Research Letters*, 2(6):269–272, 1984.

13 G. Reinelt. TSPlib — A Traveling Salesman Problem library. *ORSA J. Computing*, 3(4), 1991.

14 E. R. Scheinerman and H. S. Wilf. The rectilinear crossing number of a complete graph and Sylvester's "four point problem" of geometric probability. *The American Mathematical Monthly*, 101(10):939–943, 1994.

Properties of Minimal-Perimeter Polyominoes*

Gill Barequet[1] and Gil Ben-Shachar[1]

1 **The Technion—Israel Inst. of Technology**
{barequet,gilbe}@cs.technion.ac.il

Abstract

A polyomino is a set of connected squares on a grid. In this work we address the class of polyominoes with minimal perimeter for their area, and show a bijection between minimal-perimeter polyominoes of certain areas.

1 Introduction

A polyomino is an edge-connected set of cells on the square lattice. The area of a polyomino is the number of cells it contains. The problem of counting polyominoes dates back to the 1950s when it was studied in parallel in the fields of combinatorics [8] and statistical physics [6]. Let $A(n)$ denote the number of polyominoes of area n. A general formula for $A(n)$ is still unknown. Klarner [10] showed the existence of the *growth rate* of $A(n)$, denoting it by $\lambda := \lim_{n\to\infty} \sqrt[n]{A(n)}$. The exact value of λ is also unknown yet, and its best estimate, 4.06, is by Jensen [9]. The current best lower and upper bounds on λ are 4.0025 [3] and 4.6496 [11], respectively. Several works provide enumeration by area of special classes of polyominoes, such as column-convex [7], convex [5], and directed [4] polyominoes.

The perimeter of a polyomino P consists of the empty cells adjacent to P. Asinowski et al. [2] showed that a polyomino of area n has a perimeter of size at most $2n+2$, and provided formulae for the numbers of polyominoes with area n and perimeter $2n+2-k$, for some small values of k. In this paper, we shed some light on polyominoes with the *minimum*-size perimeter for their area. Related works are by Altshuler et al. [1] and by Sieben [12], providing a formula for the maximum area of a polyomino with a certain perimeter size. Sieben [12] also gave a formula for the minimum perimeter size of a polyomino of area n. Both works also characterized all polyominoes that have the maximum area for a given perimeter size. In this paper, we study the number of polyominoes which have the minimum perimeter size for their area, and show a bijection between some sets of minimal-perimeter polyominoes.

2 The Problem

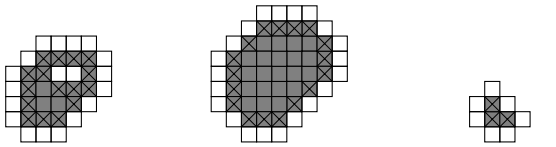
2.1 Definitions

Let Q be a polyomino, and let $\mathcal{P}(Q)$ be the perimeter of Q. Define $\mathcal{B}(Q)$, the *border* of Q, to be the set of cells of Q which have at least one empty neighboring cell. Given a polyomino Q, its *inflated* polyomino, $I(Q)$, is defined as $I(Q) = Q \cup \mathcal{P}(Q)$. Notice that the border of $I(Q)$ is a subset of the perimeter of Q. Analogously, the *deflated* polyomino, $D(Q)$, is defined as $D(Q) = Q \setminus \mathcal{B}(Q)$, which is obtained by "shaving" the outer layer, i.e., the border cells from the polyomino. Notice that the perimeter of $D(Q)$ is a subset of the border of Q. Also note that $D(Q)$ is not necessarily a valid polyomino since the removal of the border of Q may break it into disconnected pieces. Figure 1 demonstrates all the above definitions.

Following the notation of Sieben [12], we denote by $\epsilon(n)$ the minimum size of the perimeter of all polyominoes of area n. Sieben showed that $\epsilon(n) = \lceil 2 + \sqrt{8n-4} \rceil$. A polyomino Q of area n will be called a minimal-perimeter polyomino if $|\mathcal{P}(Q)| = \epsilon(n)$.

* Work on this paper by both authors has been supported in part by ISF Grant 575/15.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.



(a) Polyomino Q **(b)** $I(Q)$ **(c)** $D(Q)$

Figure 1 A polyomino Q, its inflated polyomino, and its deflated polyomino. The gray cells are the polyomino cells, while the white cells are the perimeter. Border cells are marked with crosses.

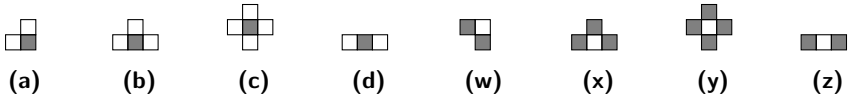


Figure 2 All possible patterns of excess cells. The gray cells are polyomino cells, while the white cells are perimeter cells. Patterns (a–d) exhibit excess border cells and their surrounding perimeter cells, while Patterns (w–z) exhibit excess perimeter cells and their surrounding polyomino cells.

2.2 The Relation between Border, Perimeter, and Excess

In this section we express the size of the perimeter of a polyomino, $|\mathcal{P}(Q)|$, as a function of the border size, $|\mathcal{B}(Q)|$, and the number of excess cells as defined below. The excess of a perimeter cell [2] is defined as the number of polyomino cells that are adjacent to it minus one, and the total excess of a polyomino Q, e_P, is defined as the sum of excess over all the cells of the perimeter of Q. Similarly, the excess of a border cell is defined as the number of perimeter cells adjacent to it minus one, and the border excess, denoted by e_B, is defined as the sum of excess over all the border cells. Let $\pi = |\mathcal{P}(Q)|$ and $\beta = |\mathcal{B}(Q)|$.

▶ **Observation 2.1.** The following holds for any polyomino: $\pi + e_P = \beta + e_B$. Equivalently,

$$\pi = \beta + e_B - e_P. \tag{1}$$

Equation (1) holds since both $\pi + e_P$ and $\beta + e_B$ are equal to the total length of the polygons forming the boundary of the polyomino. This quantity can be calculated either by summing up over the perimeter cells, where each cell contributes 1 plus its excess for a total of $\pi + e_P$, or by summing up over the border cells for a total of $\beta + e_B$. Figure 2 shows all possible patterns of border and perimeter excess cells, while Figure 3 shows a sample polyomino with some cells tagged with the corresponding patterns.

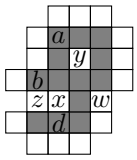


Figure 3 A sample polyomino with marked patterns.

Let $\#\square$ be the number of excess cells of a certain type in a polyomino as classified in the figure, where '$\square$' is one of the symbols a–d or w–z, as in Figure 2. Counting e_P and e_B as functions of the different patterns of excess cells, we see that $e_B = \#a+2\#b+3\#c+\#d$ and $e_P = \#w+2\#x+3\#y+\#z$. Substituting e_B and e_P in Equation (1), we obtain

$$\pi = \beta + \#a + 2\#b + 3\#c + \#d - \#w - 2\#x - 3\#y - \#z.$$

Since Pattern (c) is a singleton cell, we can ignore it in the general formula. Thus, we have

$$\pi = \beta + \#a + 2\#b + \#d - \#w - 2\#x - 3\#y - \#z.$$

2.3 Properties of Minimal-Perimeter Polyominoes

▶ **Lemma 2.2.** *Any minimal-perimeter polyomino is simply connected (that is, it does not contain holes).*

(a) Q **(b)** Q' **(c)** Q **(d)** Q'

Figure 4 Examples for the first and second parts of the proof of Theorem 2.4.

Proof. The sequence $\epsilon(n)$ is monotone increasing in the wide sense[1] [12]. Assume that there exists a minimal-perimeter polyomino Q with a hole. Consider the polyomino Q' that is obtained by filling this hole. The area of Q' is clearly larger than the area of Q, and its perimeter size is smaller since we eliminated the perimeter cells inside the hole and did not introduce new perimeter cells. This is a contradiction to $\epsilon(n)$ being monotone increasing. ◀

▶ **Lemma 2.3.** *For a simply connected polyomino, we have* $\#a + 2\#b - \#w - 2\#x = 4$.

Proof. The boundary of a polyomino without holes is a simple polygon, thus, the sum of its internal angles is $(180(v-2))°$, where v is the complexity of the polygon. Notice that Pattern (a) (resp., (b)) adds one (resp., two) 90°-vertex to the polygon. Similarly, Pattern (w) (resp. (x)) adds one (resp., two) 270°-vertex. All other patterns do not involve vertices. Let $L = \#a + 2\#b$ and $R = \#w + 2\#x$. Then, the sum of angles of the boundary polygon implies that $L \cdot 90° + R \cdot 270° = (L + R - 2) \cdot 180°$, that is, $L - R = 4$. The claim follows. ◀

▶ **Theorem 2.4.** *(Stepping Theorem) For a minimal-perimeter polyomino (except the singleton cell), we have that* $\pi = \beta + 4$.

Proof. Lemma 2.3 tells us that $\pi = \beta + 4 + \#d - \#z$. We will show that any minimal-perimeter polyomino contains neither Pattern (d) nor Pattern (z).

Let Q be a minimal-perimeter polyomino. For the sake of contradiction, assume first that there is a cell $f \in \mathcal{P}(Q)$ as part of Pattern (z). Assume w.l.o.g. that the two adjacent polyomino cells are to the left and to the right of f. These two cells must be connected, thus, the area below (or above) f must be bounded by polyomino cells. Let, then, Q' be the polyomino with the area below f, and the cell f itself, filled with polyomino cells. The cell directly above f becomes a perimeter cell, the cell f ceases to be a perimeter cell, and at least one perimeter cell in the area filled below f is eliminated, thus, $|\mathcal{P}(Q')| < |\mathcal{P}(Q)|$ and $|Q'| > |Q|$, which is a contradiction to the sequence $\epsilon(n)$ being increasing. Thus, Q does not contain perimeter cells that fit Pattern (z). Figures 4(a,b) demonstrate this argument.

Now assume for contradiction that Q contains a cell f, forming Pattern (d). Let Q' be the polyomino obtained from Q by removing f and then "pushing" together the two cells adjacent to f. This is always possible since Q is of minimal perimeter, hence, by Lemma 2.2, it is simply connected, and thus, removing f breaks Q into two separate polyominoes. Any two separated polyominoes can be shifted by one cell without colliding, thus, the transformation described above is valid. The area of Q' is one less than the area of Q, and the perimeter of Q' is smaller by at least two than the perimeter of Q, since the perimeter cells below and above f cease to be part of the perimeter, and connecting the two parts does not create new perimeter cells. From the formula of $\epsilon(n)$ we know that $\epsilon(n+1) - \epsilon(n) \le 1$ for $n \ge 2$, but $|Q| - |Q'| = 1$ and $|\mathcal{P}(Q)| - |\mathcal{P}(Q')| = 2$, hence, Q is not a minimal-perimeter polyomino, which contradicts our assumption. Thus, there are no cells in Q that fit Pattern (d). Figures 4(c,d) demonstrate this argument. This completes the proof. ◀

[1] In the sequel we simple say "monotone increasing."

2.4 Inflating a Minimal-Perimeter Polyomino

In this section we reach our main results.

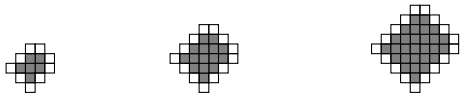
▶ **Lemma 2.5.** *If Q is a minimal-perimeter polyomino, then $|\mathcal{P}(I(Q))| \leq |\mathcal{P}(Q)| + 4$.*

Proof. Since Q is a minimal-perimeter polyomino, we know by Lemma 2.2 that $I(Q)$ is simply connected. For a hole to be formed in $I(Q)$, the original polyomino Q must have either Pattern (z) (two cells separated by a single perimeter cell), or two cells separated by *two* perimeter cells, as in ■□□■. The former case (Pattern (z)) is not possible, as is shown in the proof of Theorem 2.4. We show, using the same technique, that the latter case is also impossible.

Since $I(Q)$ is simply connected, we have, by Lemma 2.3, that $|\mathcal{P}(I(Q))| = |\mathcal{B}(I(Q))| + 4 + \#d - \#z$. Since $|\mathcal{B}(I(Q))| \leq |\mathcal{P}(Q)|$, all that remains to show is that Pattern (d) does not occur in $I(Q)$. Assume to the contrary that there is a cell f forming Pattern (d) in $I(Q)$. Since $I(Q)$ is simply connected, removing f will break it into exactly two pieces, denoted by Q_1 and Q_2. Both Q_1 and Q_2 must contain cells of the original Q since any cell in $I(Q)$ either belongs to Q or is adjacent to a cell of Q. However, this implies that Q is not connected, which is a contradiction. Hence, Q cannot contain a pattern of type (d), as required. ◀

▶ **Theorem 2.6.** *(Inheritance Theorem) If Q is a minimal-perimeter polyomino, then $I(Q)$ is a minimal-perimeter polyomino as well.*

Proof. Let Q be a minimal-perimeter polyomino. Assume to the contrary that $I(Q)$ is not a minimal-perimeter polyomino, i.e., there exists a polyomino Q' with the same area as $I(Q)$, such that $|\mathcal{P}(Q')| < |\mathcal{P}(I(Q))|$. From Lemma 2.5 we know that $|\mathcal{P}(I(Q))| \leq |\mathcal{P}(Q)| + 4$, thus, the perimeter of Q' is at most $|\mathcal{P}(Q)| + 3$, and since Q' is a minimal-perimeter polyomino, we know by Theorem 2.4 that the size of its border is at most $|\mathcal{P}(Q)| - 1$. Consider now $D(Q')$. The area of Q' is $|Q| + |\mathcal{P}(Q)|$, thus, the size of $D(Q')$ is at least $|Q|+1$, and its perimeter size is at most $\epsilon(n) - 1$ (since the perimeter of $D(Q')$ is a subset of the border of Q'). This is a contradiction to the sequence $\epsilon(n)$ being monotone increasing. Hence, Q' cannot exist, and $I(Q)$ is a minimal-perimeter polyomino. Figure 5 demonstrates this theorem. It shows a minimal-perimeter polyomino Q of area 6 and the two minimal-perimeter polyominoes of areas 15 and 28 obtained by inflating Q twice. ◀



(a) $|Q| = 6$ **(b)** $|I(Q)| = 15$ **(c)** $|I(I(Q))| = 28$

Figure 5 A demonstration of Theorem 2.6.

▶ **Corollary 2.7.** *The minimum perimeter size of a polyomino of area $n + k\epsilon(n) + 2k(k-1)$ (for $n \neq 1$ and any $k \in \mathbb{N}$) is $\epsilon(n) + 4k$.*

Proof. Inflating a minimal-perimeter polyomino of size n increases its area by $\epsilon(n)$. The border size of the inflated polyomino is $\epsilon(n)$, thus, by Theorem 2.4, the new perimeter size is $\epsilon(n) + 4$. By induction, after the kth inflation, the perimeter size is $\epsilon(n) + 4k$ and the increase in the area is $\epsilon(n) + 4(k-1)$. Summing up the increase in area, we obtain $\sum_{i=1}^{k}(\epsilon(n) + 4(i-1)) = k\epsilon(n) + 2k(k-1)$, implying the claim. ◀

▶ **Lemma 2.8.** *Let Q be a minimal-perimeter polyomino of area $n + \epsilon(n)$ (for $n \geq 3$). Then, $D(Q)$ is a valid (connected) polyomino.*

Proof. Assume to the contrary that $D(Q)$ is not connected and that it is composed of at least two parts. Assume first that $D(Q)$ is composed of exactly two parts, Q_1 and Q_2.

Define the *joint perimeter* of the two parts, $\mathcal{P}(Q_1, Q_2)$, to be $\mathcal{P}(Q_1) \cup \mathcal{P}(Q_2)$. Since Q is a minimal-perimeter polyomino of area $n + \epsilon(n)$, we know that its perimeter size is $\epsilon(n) + 4$ and its border size is $\epsilon(n)$, by Corollary 2.7 and Theorem 2.4, respectively. Thus, the size of $D(Q)$ is exactly n regardless of whether or not $D(Q)$ is connected. Since Q_1 and Q_2 are the result of deflating Q, the polyomino Q must have an (either horizontal, vertical, or diagonal) "bridge" of border cells which disappeared in the deflation. The width of the bridge is at most 2, thus, $|\mathcal{P}(Q_1) \cap \mathcal{P}(Q_2)| \leq 2$. Hence, $|\mathcal{P}(Q_1)| + |\mathcal{P}(Q_2)| - 2 \leq |\mathcal{P}(Q_1, Q_2)|$. Since $\mathcal{P}(Q_1, Q_2)$ is a subset of $\mathcal{B}(Q)$, we have that $|\mathcal{P}(Q_1, Q_2)| \leq \epsilon(n)$. Therefore,

$$\epsilon(|Q_1|) + \epsilon(|Q_2|) - 2 \leq \epsilon(n). \tag{2}$$

Recall that $|Q_1| + |Q_2| = n$. It is easy to observe that $\epsilon(|Q_1|) + \epsilon(|Q_2|)$ is minimized when $|Q_1| = 1$ and $|Q_2| = n - 1$ (or vice versa). Had the function $\epsilon(n)$ (shown in Figure 6) been $2 + \sqrt{8n - 4}$ (without rounding up), this would be obvious. But since $\epsilon(n) = \lceil 2 + \sqrt{8n - 4} \rceil$, it is a step function (with an infinite number of intervals), where the gap between all successive steps is exactly 1, except the gap between the two leftmost steps which is 2. This guarantees that despite the rounding, the minimum of $\epsilon(|Q_1|) + \epsilon(|Q_2|)$ occurs as claimed. Substituting this into Equation (2), and using the fact that $\epsilon(1) = 4$, we see that $\epsilon(n-1) + 2 \leq \epsilon(n)$. However, we know [12] that $\epsilon(n) - \epsilon(n-1) \leq 1$ for $n \geq 3$, which is a contradiction. Thus, $D(Q)$ cannot split into two parts unless it splits into two singleton cells, which is indeed the case for a minimal-perimeter polyomino of size 8.

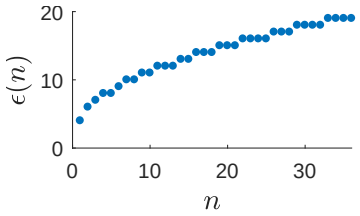


Figure 6 Values of $\epsilon(n)$.

The same method can be used to show that $D(Q)$ cannot be composed of more then two parts. Note that this proof does not hold for polyominoes of area which is not of the form $n + \epsilon(n)$, but it suffices for the proof of Theorem 2.10 below. ◀

▶ **Lemma 2.9.** *Let Q_1, Q_2 be two different minimal-perimeter polyominoes. Then, regardless of whether or not Q_1, Q_2 have the same area, $I(Q_1)$ and $I(Q_2)$ are different as well.*

Proof. Assume to the contrary that $Q = I(Q_1) = I(Q_2)$. By definition, this means that $Q = Q_1 \cup \mathcal{P}(Q_1) = Q_2 \cup \mathcal{P}(Q_2)$. Furthermore, since $Q_1 \neq Q_2$, and since a cell can belong to either a polyomino or to its perimeter, but not to both, it must be that $\mathcal{P}(Q_1) \neq \mathcal{P}(Q_2)$. The border of Q is a subset of both $\mathcal{P}(Q_1)$ and $\mathcal{P}(Q_2)$, that is, $\mathcal{B}(Q) \subset \mathcal{P}(Q_1) \cap \mathcal{P}(Q_2)$. Since $\mathcal{P}(Q_1) \neq \mathcal{P}(Q_2)$, we have that either $|\mathcal{B}(Q)| < |\mathcal{P}(Q_1)|$ or $|\mathcal{B}(Q)| < |\mathcal{P}(Q_2)|$; assume w.l.o.g. the former case. Now consider the polyomino $D(Q)$. Its area is $|Q| - |\mathcal{B}(Q)|$. The area of Q is $|Q_1| + |\mathcal{P}(Q_1)|$, thus, $|D(Q)| > |Q_1|$, and since the perimeter of $D(Q)$ is a subset of the border of Q, we conclude that $|\mathcal{P}(D(Q))| < |\mathcal{P}(Q_1)|$. However, Q_1 is a minimal-perimeter polyomino, which is a contradiction to $\epsilon(n)$ being monotone increasing. ◀

▶ **Theorem 2.10.** *(Chain Theorem) Let M_n be the set of minimal-perimeter polyominoes of area n. Then, for $n \geq 3$, we have that $|M_n| = \left|M_{n+\epsilon(n)}\right|$.*

Proof. By Theorem 2.6, if $Q \in M_n$, then $I(Q) \in M_{n+\epsilon(n)}$, and hence, by Lemma 2.9, we have that $|M_n| \leq \left|M_{n+\epsilon(n)}\right|$. Let us now show the opposite relation, namely, that $|M_n| \geq \left|M_{n+\epsilon(n)}\right|$. The combination of the two relations will imply the claim.

Let $I(M_n) = \{I(Q) \mid Q \in M_n\}$. For $Q \in M_{n+\epsilon(n)}$, our goal is to show that $Q \in I(M_n)$. Since $Q \in M_{n+\epsilon(n)}$, we have by Corollary 2.7 that $|\mathcal{P}(Q)| = \epsilon(n) + 4$. Moreover, by Theorem 2.4, we have that $|\mathcal{B}(Q)| = \epsilon(n)$, thus, $|D(Q)| = n$ and $|\mathcal{P}(D(Q))| \geq \epsilon(n)$. Since the perimeter of $D(Q)$ is a subset of the border of Q, and $|\mathcal{B}(Q)| = \epsilon(n)$, we conclude that the perimeter of $D(Q)$ and the border of Q are the same set of cells. Thus, $I(D(Q)) = Q$. Since

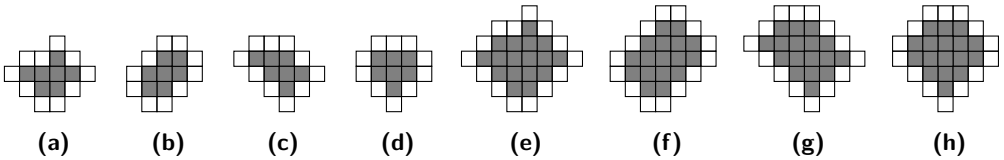


Figure 7 A demonstration of Theorem 2.10.

$|\mathcal{P}(D(Q))| = \epsilon(n)$, we have that $D(Q)$ is a minimal-perimeter polyomino, thus, $Q \in I(M_n)$ as required. Hence, $M_{n+\epsilon(n)} \subseteq I(M_n)$, implying that $\left|M_{n+\epsilon(n)}\right| \leq |I(M_n)| = |M_n|$.

Figure 7 shows, for example, all minimal-perimeter polyominoes of area 7. When they are inflated, they become the entire set of minimal-perimeter polyominoes of area 17. ◀

▶ **Corollary 2.11.** *For $n \geq 3$ and any $k \in \mathbb{N}$, we have that $|M_n| = \left|M_{n+k\epsilon(n)+2k(k-1)}\right|$.*

Proof. The claim follows from applying Theorem 2.10 repeatedly on M_n. ◀

3 Future work

We have shown that inflating a set of minimal-perimeter polyominoes of a certain area creates a new set, of the same cardinality, of minimal-perimeter polyominoes of some other area. This creates chains of sets of minimal-perimeter polyominoes of the same area. In the future we would like to characterize the roots of these chains and to determine how many minimal-perimeter polyominoes the sets of each chain contains.

References

1 Y. Altshuler, V. Yanovsky, D. Vainsencher, I.A. Wagner, and A.M. Bruckstein. On minimal perimeter polyminoes. In *DGCI*, pages 17–28. Springer, 2006.
2 A. Asinowski, G. Barequet, and Y. Zheng. Enumerating polyominoes with fixed perimeter defect. In *Proc. 9th European Conf. on Combinatorics, Graph Theory, and Applications*, volume 61, pages 61–67, Vienna, Austria, August 2017. Elsevier.
3 G. Barequet, G. Rote, and M. Shalah. $\lambda > 4$: An improved lower bound on the growth constant of polyominoes. *Comm. of the ACM*, 59(7):88–95, 2016.
4 M. Bousquet-Mélou. New enumerative results on two-dimensional directed animals. *Discrete Mathematics*, 180(1-3):73–106, 1998.
5 M. Bousquet-Mélou and J.-M. Fédou. The generating function of convex polyominoes: The resolution of a q-differential system. *Discrete Mathematics*, 137(1-3):53–75, 1995.
6 S.R. Broadbent and J.M. Hammersley. Percolation processes: I. Crystals and Mazes. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 53, pages 629–641. Cambridge University Press, 1957.
7 M.-P. Delest. Generating functions for column-convex polyominoes. *J. of Combinatorial Theory, Series A*, 48(1):12–31, 1988.
8 S.W. Golomb. Checker boards and polyominoes. *The American Mathematical Monthly*, 61(10):675–682, 1954.
9 I. Jensen and A.J. Guttmann. Statistics of lattice animals (polyominoes) and polygons. *J. of Physics A: Mathematical and General*, 33(29):L257, 2000.
10 D.A. Klarner. Cell growth problems. *Canadian J. of Mathematics*, 19:851–863, 1967.
11 D.A. Klarner and R.L. Rivest. A procedure for improving the upper bound for the number of n-ominoes. *Canadian J. of Mathematics*, 25(3):585–602, 1973.
12 N. Sieben. Polyominoes with minimum site-perimeter and full set achievement games. *European J. of Combinatorics*, 29(1):108–117, 2008.

On Optimal Polyline Simplification using the Hausdorff and Fréchet Distance

Marc van Kreveld[1], Maarten Löffler[1], and Lionov Wiratma[1,2]

1 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands**
[m.j.vankreveld|m.loffler|l.wiratma]@uu.nl
2 **Dept. of Informatics, Parahyangan Catholic University, Indonesia**
lionov@unpar.ac.id

Abstract

We revisit the classical polygonal line simplification problem and study it using the Hausdorff distance and Fréchet distance. We use these measures in its pure form, namely: for a given $\varepsilon > 0$, choose a minimum size subsequence of the vertices of the input such that the Hausdorff or Fréchet distance between the input and output polylines is at most ε.

We analyze how the Douglas-Peucker and Imai-Iri simplification algorithms perform compared to the optimum possible. We prove that it is NP-hard to compute the optimal simplification under (undirected) Hausdorff distance. Under the Fréchet distance, the optimal simplification of a polygonal line consisting of n vertices can be computed in $O(kn^5)$ time and $O(kn^2)$ space, where k is the output complexity of the simplification.

1 Introduction

Line simplification (a.k.a. polygonal approximation) is one of the oldest and best studied applied topics in computational geometry. A simplification should have a similar shape as the input, and hence we need a similarity or distance measure to specify when a simplification is acceptable. The *Hausdorff distance* and the *Fréchet distance* are probably the best known distance measures used for shape similarity in computational geometry.

Among the well-known simplification algorithms, the ones by Douglas and Peucker [4] and by Imai and Iri [7] are frequently implemented and cited. For a given constant $\varepsilon > 0$, both algorithms start with a polygonal line (henceforth *polyline*) as an input, specified by a sequence of points $\langle p_1, \ldots, p_n \rangle$, and compute a subsequence starting with p_1 and ending with p_n, representing a simplified polyline which is within a distance of ε from the input.

The Douglas-Peucker algorithm [4] is a simple procedure that starts with a simplification $\overline{p_1 p_n}$, determines the furthest vertex p_k, and if it is further than ε, adds p_k to the simplification. This gives two subproblems with $\overline{p_1 p_k}$ and $\overline{p_k p_n}$ that are solved recursively in the same way and then merged. Hershberger and Snoeyink [6] provide an $O(n \log n)$ time implementation of this algorithm. The Imai-Iri algorithm [7] takes a different approach. It determines for every *link* $\overline{p_i p_j}$ $(i < j)$ if it lies within distance ε from the vertices $p_{i+1}, \ldots, p_{j-1}$ and if so, deems it *valid*. The graph G with all vertices $p_1, \ldots, p_n$ as nodes and all valid links as edges can then be constructed, and a minimum link path from p_1 to p_n represents an optimal simplification. By the implementation of Chan and Chin [3], this algorithm runs in $O(n^2)$ time.

The Imai-Iri algorithm is considered an optimal line simplification algorithm, because it minimizes the number of vertices in the output. It also guarantees the Hausdorff distance between the input and the simplification of at most ε. However, the simplification is not optimal for the Hausdorff distance, because there are simple examples where a simplification with fewer vertices have the Hausdorff distance at most ε to the input. This comes from the fact that the algorithm uses the Hausdorff distance *between a link* $\overline{p_i p_j}$ *and the sub-polyline* $\langle p_i, \ldots, p_j \rangle$, and not an overall Hausdorff distance.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

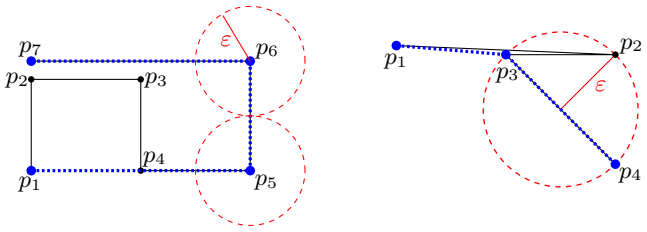


Figure 1 The Douglas-Peucker and Imai-Iri algorithms do not simplify the inputs for the Hausdorff distance (left) or the Fréchet distance (right). The optimal simplifications are shown dotted in blue.

Note that we can easily adapt the Imai-Iri algorithm to guarantee the Fréchet distance of at most ε: we deem a link $\overline{p_i p_j}$ valid if its Fréchet distance to the sub-polyline $\langle p_i, \ldots, p_j \rangle$ is at most ε [1]. This simple variation of the Imai-Iri algorithm does not yield the optimal simplification within the Fréchet distance of ε, because *it requires us to match a vertex p_i in the input to the vertex p_i in the output in the parametrizations, if p_i is used in the output.* This restriction on the parametrizations limits the simplification in undesirable ways.

The examples in Figure 1 show that under the Hausdorff distance (left) and Fréchet distance (right) the Douglas-Peucker and Imai-Iri simplifications are both equal to P itself and may use more vertices than an optimal simplification using these measures.

The discussion begs the following questions: How much worse do the known algorithms and their variations perform in theory, when compared to the optimal Hausdorff and Fréchet simplifications? What if the optimal Hausdorff and Fréchet simplifications use a smaller value than ε? How efficiently can the optimal Hausdorff simplification and the optimal Fréchet simplification be computed (when using the input vertices)?

Organization and results. In Section 2 we show that the optimal simplification has fewer vertices than the Imai-Iri output, both under the Hausdorff and the Fréchet distance (we ignore the Douglas-Peucker method from now on because it never yields fewer vertices than the Imai-Iri method). In particular, we analyze how much worse the output of the Imai-Iri algorithm can be for the two measures. In Section 3 we show that the optimal simplification under the undirected Hausdorff distance is NP-hard to compute. In Section 4 we show that simplification can be done optimally in polynomial time for the Fréchet distance.

2 Approximation Quality of Imai-Iri Simplification

We denote the simplification by the Imai-Iri algorithm under the Hausdorff distance as $II_H(P, \varepsilon)$, and will leave out the arguments P and/or ε if they are understood. We refer to the simplification from the adapted Imai-Iri algorithm using the Fréchet distance as $II_F(P, \varepsilon)$. We denote the optimal simplification using the Hausdorff distance by $OPT_H(P, \varepsilon)$, and using the Fréchet distance by $OPT_F(P, \varepsilon)$. The example in Figure 1 shows that to let II_H use as few vertices as OPT_H, we must use 2ε instead of ε when the example is stretched horizontally. For the Fréchet distance, the enlargement factor needed in the example approaches $\sqrt{2}$ if we put p_1 far to the left. In this section we analyze how the approximation enlargement factor relates to the number of vertices in the Imai-Iri simplification and the optimal ones.

Hausdorff Distance To show that II_H may use many more vertices than OPT_H, even if we enlarge ε, we give a construction where this occurs in Figure 2 that applies for both the directed and undirected Hausdorff distance.

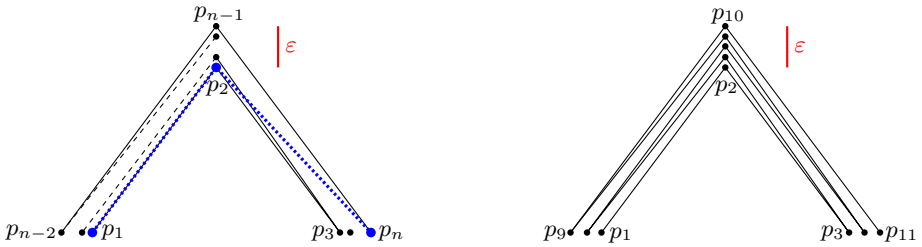


Figure 2 The Imai-Iri algorithm may not be able to simplify $\langle p_1, \ldots, p_n \rangle$ at all. The optimal Hausdorff simplification (dotted, blue) has three vertices. Right, an example input with 11 vertices.

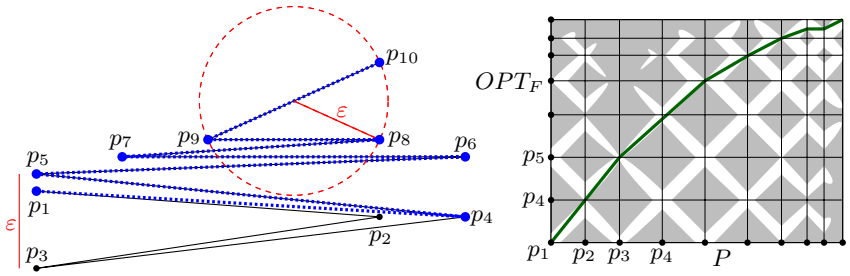


Figure 3 The optimal simplification can skip p_2 and p_3; in the parametrizations witnessing the Fréchet distance, OPT_F "stays two vertices behind" on the input until the end. Right, the free space diagram of P and OPT_F.

An optimal simplification is $\langle p_1, p_i, p_n \rangle$ where i is any even number between 1 and n. Since the only valid links are the ones connecting two consecutive vertices of P, II_H is P itself. If the triangle is large enough with respect to ε, this remains true even if we give the Imai-Iri algorithm a much larger error threshold than ε.

▶ **Theorem 2.1.** *For any $c > 1$, there exists a polyline P with n vertices and an $\varepsilon > 0$ such that $II_H(P, c\varepsilon)$ has n vertices and $OPT_H(P, \varepsilon)$ has 3 vertices.*

Fréchet Distance We give another input polyline P in Figure 3 to show that II_F does not approximate OPT_F even if II_F is allowed to use ε that is larger by a constant factor. Our main construction has ten vertices placed in such a way that II_F has all ten vertices, while OPT_F has only eight of them, see Figures 3. We can append multiple copies of this construction together with a suitable connection in between. We obtain:

▶ **Theorem 2.2.** *There exist constants $c_1 > 1$, $c_2 > 1$, a polyline P with n vertices, and an $\varepsilon > 0$ such that $|II_F(P, c_1\varepsilon)| > c_2|OPT_F(P, \varepsilon)|$.*

By a result of Agarwal et al. [1], we know that the theorem is not true for $c_1 \geq 4$.

3 Algorithmic Complexity of Optimal Simplification using the Hausdorff Distance

The results in the previous section lead us to the following question: *Is it possible to compute the optimal Hausdorff or Fréchet simplification in polynomial time?*

We first consider the undirected (or bidirectional) Hausdorff distance; that is, we require both the maximum distance from the initial polyline P to the simplified polyline Q and the maximum distance from Q to P to be at most ε.

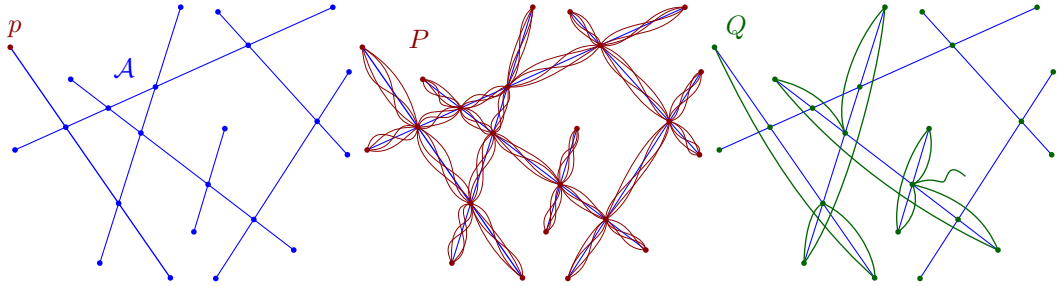


Figure 4 The construction: $\mathcal{A}$ is the arrangement of a set of segments S. We build an input path P that "paints" over S completely, and we are looking for an output path Q that corresponds to a Hamiltonian cycle. In this case, there is no Hamiltonian cycle, and the path gets stuck.

▶ **Theorem 3.1.** *Given a polyline $P = \langle p_1, p_2, \ldots, p_n \rangle$ and a value ε, the problem of computing a minimum length polyline Q defined by a subsequence of the vertices of P such that the undirected Hausdorff distance between P and Q is at most ε is NP-hard.*

Our proof uses a reduction from Hamiltonian cycle in segment intersection graphs. Since deciding if a Hamiltonian cycle exists is NP-complete in planar graphs [5], and planar graphs are included in segment intersections graphs [2], it follows that Hamiltonian cycle in segment intersections graphs is NP-complete. Let S be a set of n line segments in $\mathbb{R}^2$, and assume all intersections are proper (if not, extend the segments slightly). Let G be its intersection graph. Assume that G is connected; otherwise, there is no Hamiltonian cycle in G.

We first construct an initial polyline P as follows (see Figure 4). Let $\mathcal{A}$ be the arrangement of S, let p be some endpoint of a segment in S, and let π be any path on $\mathcal{A}$ that starts and finishes at p and visits all vertices and edges of $\mathcal{A}$. Then P is simply $3n + 1$ copies of π appended to each other. We now set ε to a sufficiently small value. Then, an output polyline Q with Hausdorff distance at most ε to P must also visit all vertices and edges of $\mathcal{A}$, and stay close to $\mathcal{A}$. If ε is sufficiently small, there will be no benefit for Q to ever leave $\mathcal{A}$.

▶ **Lemma 3.2.** *A solution Q of length $3n + 1$ exists iff G admits a Hamiltonian cycle.*

Proof. Clearly, any simplification Q will need to visit the $2n$ endpoints of the segments in S, and—since it starts and ends at the same point p—will need to have length at least $2n + 1$. Furthermore, Q will need to have at least two internal vertices on every segment $s \in S$: once to enter and once to leave the segment (we cannot enter or leave a segment at an endpoint since all intersections are proper intersections). This means the theoretical minimum number of vertices possible for Q is $3n + 1$.

Now, if G admits a Hamiltonian cycle, it is easy to construct a simplification with $3n + 1$ vertices. We start at p, an endpoint of the segment s_1, and collect the other endpoint. Then we follow the Hamiltonian cycle to segment s_2; by definition $s_1 s_2$ is an edge in G so their corresponding segments intersect, and we use the intersection point to leave s_1 and enter s_2. We proceed in this fashion until we reach s_n, which intersects s_1, and finally return to p.

On the other hand, any solution with $3n + 1$ vertices must necessarily be of this form and therefore imply a Hamiltonian cycle: in order to have only 3 vertices per segment the vertex at which we leave s_1 must coincide with the vertex at which we enter some other segment, which we call s_2, and we must continue until we visited all segments and return to p. ◀

For completeness, we also state the results for simplification using the *directed* Hausdorff distance, in both directions. If we require the distance from the input to the simplification

to be at most ε, then an optimal simplification using the (directed) Hausdorff distance is NP-hard to compute. However, if we require the distance from the simplification to the input to be at most ε, an optimal simplification can be computed in polynomial time. We give the proofs in the full paper.

4 Algorithmic Complexity of Optimal Simplification using the Fréchet Distance

In this section, we show that for a given polyline $P = \langle p_1, p_2, ..., p_n \rangle$ and an error ε, the optimal simplification $Q = OPT_F(P, \varepsilon)$ can be computed in polynomial time using a dynamic programming approach. First, we define π, a parameterization of P as a continuous mapping: $\pi : [0, 1] \to \mathbb{R}^2$ where $\pi(0) = p_1$ and $\pi(1) = p_n$. We also write $P[s, t]$ for $0 \leq s \leq t \leq 1$ to be the subcurve of P starting at $\pi(s)$ and ending at $\pi(t)$, also writing $P[t] = P[0, t]$ for short.

For the dynamic programming approach to work, we might imagine to store, for each vertex p_i and value k, the point $\pi(\alpha)$ which is the farthest along P such that a simplification of $\langle p_1, ..., p_i \rangle$ using k links has Fréchet distance at most ε to $P[\alpha]$. However, this is not sufficient to ensure that we find an optimal solution (see the full paper for details). Instead, we argue that if we maintain the set of *all* points at P that can be "reached" by a simplification up to each vertex, then we can make dynamic programming work. We now make this precise and argue that the complexity of these sets of reachable points is never worse than linear.

We say that a point $\pi(t)$ can be *reached* by a (k, i)-simplification for $0 \leq k < i \leq n$ if there exists a simplification of $\langle p_1, \ldots, p_i \rangle$ using k links which has Fréchet distance at most ε to $P[t]$. We let $\rho(k, i, t) = \texttt{true}$ in this case, and $\texttt{false}$ otherwise. With slight abuse of notation we also say that t itself is reachable, and that an interval I is reachable if all $t \in I$ are reachable (by a (k, i)-simplification).

▶ **Observation 4.1.** *A point $\pi(t)$ can be reached by a (k, i)-simplification if and only if there exist a $0 < h < i$ and a $0 \leq s \leq t$ such that $\pi(s)$ can be reached by a $(k - 1, h)$-simplification and the segment $\overline{p_h p_i}$ has Fréchet distance at most ε to $P[s, t]$.*

Proof. Follows directly from the definition of the Fréchet distance. ◀

Observation 4.1 immediately suggests a dynamic programming algorithm: for every k and i we store a subdivision of $[0, 1]$ into intervals where ρ is true and intervals where ρ is false, and we calculate them for increasing values of k. We simply iterate over all possible values of h, calculate which intervals can be reached using a simplification via h, and then take the union over all those intervals. For this, the only unclear part is how to calculate these intervals. We argue that, for any given k and i, there are at most $n - 1$ reachable intervals on $[0, 1]$, each contained in an edge of P. Indeed, every (k, i)-reachable point $\pi(t)$ must have distance at most ε to p_i, and since the edge e of P that $\pi(t)$ lies on intersects the disk of radius ε centered at p_i in a line segment, every point on this segment is also (k, i)-reachable. We denote the farthest point on e which is (k, i)-reachable by $\hat{t}$.

Furthermore, we argue that for each edge of P, we only need to take the farthest reachable point into account during our dynamic programming algorithm.

▶ **Lemma 4.2.** *If k, h, i, s, and t exist such that $\rho(k - 1, h, s) = \rho(k, i, t) = \texttt{true}$, and $\overline{p_h p_i}$ has Fréchet distance $\leq \varepsilon$ to $P[s, t]$, then $\overline{p_h p_i}$ also has Fréchet distance $\leq \varepsilon$ to $P[\hat{s}, \hat{t}]$.*

Proof. By the above argument, $P[s, \hat{s}]$ is a line segment that lies completely within distance ε from p_h, and $P[t, \hat{t}]$ is a line segment that lies completely within distance ε from p_i.

We are given that the Fréchet distance between $\overline{p_h p_i}$ and $P[s,t]$ is at most ε; this means a mapping $f : [s,t] \to \overline{p_h p_i}$ exists such that $|\pi(x) - f(x)| \leq \varepsilon$. Let $q = f(s')$. Then $|p_h - \pi(\hat{s})| \leq \varepsilon$ and $|q - \pi(\hat{s})| \leq \varepsilon$, so the line segment $\overline{p_h q}$ lies fully within distance ε from $\hat{s}$.

Therefore, we can define a new ε-Fréchet mapping between $P[\hat{s},\hat{t}]$ and $\overline{p_h p_i}$ which maps $\hat{s}$ to the segment $\overline{p_h q}$, the curve $P[\hat{s},t]$ to the segment $\overline{q p_i}$ (following the mapping given by f), and the segment $\overline{\pi(t)\pi(\hat{t})}$ to the point p_i. ◀

Now, we can compute the optimal simplification by maintaining a $k \times n \times n$ table storing $\rho(k,i,\hat{t})$, and calculate each value by looking up n^2 values for the previous value of k, and testing in linear time for each combination whether the Fréchet distance between the new link and $P[\hat{s},\hat{t}]$ is within ε or not.

▶ **Theorem 4.3.** *Given a polyline $P = \langle p_1, ..., p_n \rangle$ and a value ε, we can compute the optimal polyline simplification of P that has Fréchet distance at most ε to P in $O(kn^5)$ time and $O(kn^2)$ space, where k is the output complexity of the optimal simplification.*

5 Future Work

A number of challenging open problems remain. First, we would like to know whether the problem of computing an optimal simplification using the Hausdorff distance remains NP-hard when the simplification may not have self-intersections. Second, we are interested in the computational status of the optimal simplification when the simplification need not use the vertices of the input. Finally, we may consider optimal polyline simplifications using the weak Fréchet distance.

Acknowledgements. M.v.K is supported by the Netherlands Organisation for Scientific Research (NWO). M.L. is supported by the Netherlands Organisation for Scientific Research (NWO). L.W. is supported by the Ministry of Research, Technology and Higher Education of Indonesia (138.41/E4.4/2015).

References

1 Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3):203–219, 2005.

2 Jérémie Chalopin and Daniel Gonçalves. Every planar graph is the intersection graph of segments in the plane: Extended abstract. In *Proceedings 41st Annual ACM Symposium on Theory of Computing*, STOC '09, pages 631–638, New York, NY, USA, 2009. ACM.

3 W.S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 06(01):59–77, 1996.

4 David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.

5 M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976.

6 John Hershberger and Jack Snoeyink. An $O(n \log n)$ implementation of the Douglas-Peucker algorithm for line simplification. In *Proceedings 10th Annual Symposium on Computational Geometry*, SCG '94, pages 383–384, New York, NY, USA, 1994. ACM.

7 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve - formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology: A Computational Geometric Approach to the Analysis of Form*. North-Holland, Amsterdam, 1988.

Probabilistic embeddings of the Fréchet distance*

Anne Driemel[1] and Amer Krivošija[2]

1 **Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands**
a.driemel@tue.nl

2 **Department of Computer Science, TU Dortmund, Germany**
amer.krivosija@tu-dortmund.de

Abstract

The Fréchet distance is a popular distance measure for curves which naturally lends itself to fundamental computational tasks, such as clustering, nearest-neighbor searching, and spherical range searching in the corresponding metric space. However, its inherent complexity poses considerable computational challenges in practice. To address this problem we study distortion of the probabilistic embedding that results from projecting the curves to a randomly chosen line. Such an embedding could be used in combination with, e.g. locality-sensitive hashing. We show that in the worst case and under reasonable assumptions, the discrete Fréchet distance between two polygonal curves in $\mathbb{R}^2$ or $\mathbb{R}^3$ of complexity t degrades by a factor linear in t with constant probability. We show upper and lower bounds on the distortion.

1 Introduction

The Fréchet distance is a distance measure for curves which naturally lends itself to fundamental computational tasks, such as clustering, nearest-neighbor searching, and spherical range searching in the corresponding metric space. However, their inherent complexity poses considerable computational challenges in practice. Indeed, spherical range searching under the Fréchet distance was recently the topic of the yearly ACM SIGSPATIAL GISCUP competition[1], highlighting the relevance and the difficulty of designing efficient data structures for this problem. At the same time, Afshani and Driemel show lower bounds on the space-query-tradeoff in the pointer model [1] that demonstrate that this problem is even harder than simplex-range searching.

The computational complexity of computing a single Fréchet distance between two given curves is a well-studied topic [2, 6–9, 12, 15]. It is believed that it takes time that is quadratic in the length of the curves and this running time can be achieved by applying dynamic programming. In this body of literature, the case of 1-dimensional curves under the continuous Fréchet distance stands out. In particular, no lower bounds are known on computing the continuous Fréchet distance between 1-dimensional curves. It has been observed that the problem has a special structure in this case [10]. Clustering under the Fréchet distance can be done efficiently for 1-dimensional curves [13], but seems to be harder for curves in the plane or higher dimensions. Bringmann and Künnemann use projections to lines to speed up their approximation algorithm for the Fréchet distance [8]. They show that the distance computation can be done in linear time, if the convex hulls of the two curves are disjoint. It is tempting to believe that the curves being restricted to 1-dimensional space makes the

* Driemel has been supported by NWO Veni project "Clustering time series and trajectories (10019853)". Krivošija has been partly supported by DFG within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A2.

[1] 6th ACM SIGSPATIAL GISCUP 2017, see also http://sigspatial2017.sigspatial.org/giscup2017/

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

problem significantly easier. However, in the general case, there are no algorithms known which are faster for 1-dimensional curves than for curves in higher dimensions. In practice, it is very common to separate x and y components of trajectories to simplify computational tasks. It seems that in practice the inherent character of a trajectory is often largely preserved when restricted to one of the coordinates of the ambient space. Mathematically, this amounts to projecting the trajectory to a line.

This motivates our study of probabilistic embeddings of the Fréchet distance into the space of 1-dimensional curves. Concretely, we study distortion of the probabilistic embedding that results from projecting the curves to a randomly chosen line. Such a random projection could be used in combination with probabilistic data structures, e.g. locality-sensitive hashing [14], but also with the multi-level data structures for Fréchet range searching given by Afshani and Driemel [1]. See below for a more in-depth discussion of these data structures.

We show that in the worst case and under certain assumptions, the discrete Fréchet distance between two polygonal curves in $\mathbb{R}^2$ or $\mathbb{R}^3$ of complexity t degrades by a factor linear in t with constant probability. In particular, we show upper and lower bounds on the change in distance for the class of c-packed curves. The notion of the c-packed curves was introduced by Driemel, Har-Peled and Wenk in [12] and has proved useful as a realistic input assumption [3, 6, 11]. A curve is called c-packed for a value $c > 0$ if the length of the intersection of the curve with any ball of any radius r is a most cr. While our study is mostly restricted to the discrete Fréchet distance, we expect that our techniques can be extended to the case of the continuous Fréchet distance.

A closely related distance measure, which is popular in the field of data-mining, is dynamic time warping (DTW). The computational complexity of DTW has also been extensively studied, both empirically and in theory [3, 16]. Some of our lower bounds extend to DTW.

1.1 Related work

The work that is perhaps closest to ours is a recent result by Backurs and Sidiropoulos [4]. They gave an embedding of the Hausdorff distance into constant-dimensional ℓ_∞ space with constant distortion. More precisely, for any $s, d \geq 1$, they obtain an embedding for the Hausdorff distance over point sets of size s in d-dimensional space, into $\ell_\infty^{s^{O(s+d)}}$ with distortion $s^{O(s+d)}$. No such metric embeddings are known for the discrete or continuous Fréchet distance. It has been shown that the doubling dimension of the Fréchet distance is unbounded, even in the case when the metric spaces is restricted to curves of constant complexity [13]. A result of Bartal *et al.* [5] for doubling spaces implies that a metric embedding of the Fréchet distance into an ℓ_p space would have at least super-constant distortion, but it is not known how to find such an embedding.

The complexity of classic data structuring problems for the Fréchet distance is still not very well-understood, despite several papers on the topic. We review what is known for nearest-neighbor searching and range searching. Indyk [17] gave a deterministic and approximate near-neighbor data structure for the discrete Fréchet distance. Given n curves which have at most t vertices, this data structure achieves approximation factor $O(\log t + \log\log n)$ and has query time $O(\text{poly}(t)\log n)$. This data structure requires large space, as it precomputes all queries with curves with $\sqrt{t}$ vertices. For short curves (with $t \in O(\log n)$) Driemel and Silvestri [14] described an approximate near-neighbor structure based on locality-sensitive hashing with approximation factor $O(t)$, query time $O(t \log n)$, using space $O(n \log n + tn)$. LSH is a technique that uses families of hash functions with the property that near points are more likely to be hashed to the same index than far points. Driemel and Silvestri were the first to define locality-sensitive hash functions for the discrete Fréchet distance. No such hash

functions are known for the continuous case. It is conceivable that the concept of signatures which was introduced by Driemel, Krivosija and Sohler [13] in the context of clustering of 1-dimensional curves could be used to define an LSH for the continuous case and that this technique could be used in combination with projections to random lines.

Afshani and Driemel recently showed how to leverage semi-algebraic range searching for this problem [1]. Their data structure also supports polygonal curves of low complexity and answers queries exactly. In particular, for the discrete Fréchet distance they describe a data structure which uses space in $O\big(n(\log\log n)^{t_s-1}\big)$ and achieves query time in $O\Big(n^{1-1/d}\cdot\log^{O(t_s)} n\cdot t_q^{O(d)}\Big)$, where t_s denotes the complexity of an input curve and it is assumed that the complexity of the query curves is upper-bounded by a polynomial of $\log n$. For the continuous Fréchet distance they describe a data structure for polygonal curves in the plane which uses space in $O\Big(n(\log\log n)^{O(t_s^2)}\Big)$ and achieves query time in $O\Big(\sqrt{n}\log^{O(t_s^2)} n\Big)$. For the case where the curves lie in dimension higher than 2 and the distance measure is the continuous Fréchet distance, no data structures for range searching or range counting are known.

1.2 Our results

Given two polygonal curves P and Q with t vertices each from $\mathbb{R}^2$ or $\mathbb{R}^3$. Consider sampling a unit vector $\mathbf{u}$ in $\mathbb{R}^2$ (resp. $\mathbb{R}^3$ if the curves lie in $\mathbb{R}^3$) uniformly at random and let P' and Q' be the projections of the two curves to the line supporting $\mathbf{u}$. We show that if the curves P and Q are c-packed for constant c, then, with constant probability, the discrete Fréchet distance between the curves P and Q degrades by at most a linear factor in t.

▶ **Theorem 1.1.** *For any two polygonal curves P and Q and for any $\gamma\in(0,1)$*

$$Pr\left[\frac{d_F(P,Q)}{d_F(P',Q')}\leq\frac{12c+16}{\gamma}\cdot t\right]\geq 1-\gamma.$$

We also present a lower bound on the ratio of the two distances. The construction of the lower bound uses c-packed curves with $c<3$.

▶ **Theorem 1.2.** *There exist polygonal curves P and Q, such that for any $\gamma\in(0,1/\pi)$*

$$Pr\left[\frac{d_F(P,Q)}{d_F(P',Q')}\geq\frac{5\pi\gamma}{6}\cdot t\right]\geq 1-\gamma.$$

Theorem 1.2 holds for the continuous Fréchet distance and for dynamic time warping distance as well. We also show that there exist polygonal curves P and Q that are not c-packed for sublinear c and their (continuous or discrete) Fréchet distance degrades by a linear factor for any projection line (i.e. with probability 1).

2 Preliminaries

Throughout the paper we use the following notational conventions. Consider two polygonal curves $P=\{p_1,p_2,\ldots,p_t\}$ and $Q=\{q_1,q_2,\ldots,q_t\}$ in $\mathbb{R}^d$ given by their sequences of vertices. We choose a unit vector $\mathbf{u}$ in $\mathbb{R}^d$ by choosing a point on the $(d-1)$-dimensional unit hypersphere uniformly at random. We denote with L the line through the origin that supports the vector $\mathbf{u}$. Let $P'=\{p'_1,p'_2,\ldots,p'_t\}$ and $Q'=\{q'_1,q'_2,\ldots,q'_t\}$ be the projections of P and Q to L, defined by $p'_i=\langle p_i,\mathbf{u}\rangle$ and $q'_j=\langle q_j,\mathbf{u}\rangle$, for all $1\leq i\leq t$ and $1\leq j\leq t$. We denote $\delta_{i,j}=\|q_j-p_i\|$ and $\delta'_{i,j}=\|q'_j-p'_i\|$, for all $1\leq i\leq t$ and $1\leq j\leq t$, i.e. $\delta_{i,j}$ and

$\delta'_{i,j}$ are the pairwise distances of the vertices for the input curves P and Q and for their respective projections P' and Q'.

We define the discrete Fréchet distance of P and Q as follows: we call the *traversal* T of P and Q the sequence of pairs of indices (i, j) of vertices $(p_i, q_j) \in P \times Q$ such that

i) the traversal T starts with $(1, 1)$ and ends with (t, t), and

ii) the pair (i, j) of T can be followed only by one of $(i + 1, j)$, $(i, j + 1)$ or $(i + 1, j + 1)$.

We notice that every traversal is monotone. If $\mathcal{T}$ is the set of all traversals T of P and Q, then the discrete *Fréchet distance* between P and Q is defined as

$$d_F(P, Q) = \min_{T \in \mathcal{T}} \max_{(i,j) \in T} \|p_i - q_j\|. \tag{1}$$

Furthermore, we define a directed, vertex-weighted graph $G = (V, E)$ on the node set $V = \{(i, j) : 1 \leq i, j \leq t\}$. A node (i, j) corresponds to a pair of vertices p_i of P and q_j of Q and we assign it the weight $\delta_{i,j}$. The set of edges is defined as $E = \{((i, j), (i', j')) : i' \in \{i, i + 1\}, j' = \{j, j + 1\}, 1 \leq i, i', j, j' \leq t\}$. The set of paths in the graph G between $(1, 1)$ and (n, n) corresponds to the set of traversals $\mathcal{T}$. We call a path in G which does not start in $(1, 1)$ or end in (t, t) a *partial traversal* of P and Q.

It is useful to picture the nodes of the graph G as a matrix, where rows correspond to the vertices of P and columns correspond to the vertices of Q. For any fixed value $\Delta > 0$, we define the free-space matrix[2] $F_\Delta = (\phi_{i,j})_{1 \leq i,j \leq t}$ with

$$\phi_{i,j} = \begin{cases} 1 & \text{if } \|q_j - p_i\| < \Delta \\ 0 & \text{if } \|q_j - p_i\| \geq \Delta. \end{cases}$$

Overlaying the graph with the free-space matrix for $\Delta > d_F(P, Q)$, we can observe that there exists a path in the graph from $(1, 1)$ to (t, t) that visits only the matrix entries with value 1. Moreover, the existence of such a path in the free-space matrix for some value of Δ implies that $\Delta > d_F(P, Q)$.

We define c-packedness of curves as follows.

▶ **Definition 2.1** (c-packed curve)**.** Given $c > 0$, a curve $P \in \mathbb{R}^d$ is c-packed if for any point $p \in \mathbb{R}^d$ and any radius $r > 0$, the total length of the curve P inside the hypersphere $\texttt{ball}(p, r)$ is at most $c \cdot r$.

We prove the following basic fact about random projections to a line. For a general problem in $\mathbb{R}^d$ the probability bound degrades due to the measure concentration around $\pi/2$.

▶ **Lemma 2.2.** *If the line segment $\overline{pq}$ is projected to the straight line L, supported by the unit vector chosen uniformly at random on the unit hypersphere in $\mathbb{R}^2$ or $\mathbb{R}^3$, the probability that its length will be reduced by a factor greater than φ is at most φ.*

3 Upper bound

The discrete Fréchet distance between curves P and Q is realized by some pair (p_i, q_j) of vertices $p_i \in P$ and $q_j \in Q$, being at the distance $\|p_i - q_j\| = \delta$. We would like to apply Lemma 2.2 to this pair of vertices to show that the distance is preserved up to some constant factor. However, it is possible that the pairwise distances in the projection are such that a

[2] Note that the conventional definition of the free-space matrix for parameter Δ is slightly different, since usually there is an 1-entry iff $\|q_j - p_i\| \leq \Delta$. We are using this definition since it better suits our needs.

cheaper traversal is possible that avoids the pair (p_i, q_j) altogether. Therefore, we apply the lemma to a subset of pairs of vertices of P and Q whose distance is large (e.g. larger than $\Delta = \delta/\theta$ for some small value of $\theta \geq 1$) and such that the chosen set forms a hitting set for the set of traversals $\mathcal{T}$. To this end we introduce the notion of the *guarding set*:

▶ **Definition 3.1** (Guarding set)**.** For any two polygonal curves $P = \{p_1, \ldots, p_t\}$ and $Q = \{q_1, \ldots, q_t\}$ and a given parameter $\theta \geq 1$, a θ-guarding set $B \subseteq V$ for P and Q is a subset of the set of vertices of G that satisfies the following conditions:
a) (distance property) for all $(i, j) \in B$, it holds that $\delta_{i,j} \geq d_F(P, Q)/\theta$, and
b) (guarding property) for any traversal T of P and Q, it is $T \cap B \neq \emptyset$.

Note that the set B "guards" every traversal of P and Q in the sense that any path in G from $(1, 1)$ to (t, t) has non-empty intersection with B. In other words, B is a hitting set for the set of traversals $\mathcal{T}$. We can prove the following lemma using Lemma 2.2 for all elements of B in a union bound.

▶ **Lemma 3.2.** *Given parameter $\theta \geq 1$, if B is a θ-guarding set for the given curves $P = \{p_1, \ldots, p_t\}$ and $Q = \{q_1, \ldots, q_t\}$ from $\mathbb{R}^2$ or $\mathbb{R}^3$, and if P' and Q' are their projections to the straight line L, whose support unit vector $\boldsymbol{u}$ is chosen uniformly at random on the unit hypersphere, then for any $\beta > 1$ it holds that*

$$\frac{d_F(P', Q')}{d_F(P, Q)} \geq \frac{1}{\beta \cdot \theta \cdot |B|}$$

with positive constant probability at least $1 - 1/\beta$.

To show existence of a θ-guarding set B for any $\theta \geq 1$ we can construct such a set using breadth-first-search over graph G. Unfortunately, such built set B can have a quadratic number of elements in terms of the input size in the general case.

If the input curves P and Q are c-packed for some constant c, $c \geq 2$, then we construct the 1-guarding set B and modify it using the trimming operation based on Lemma 3.3. The idea is to trim the part of the graph G reachable by a partial traversal from $(1, 1)$ that does not pass through any of the vertices of B. See Figure 1 for an illustration.

▶ **Lemma 3.3.** *Given point p and a c-packed curve $Q = \{q_1, \ldots, q_t\}$ from $\mathbb{R}^d$. Then for any value $b > 0$ there exists a value $r \in [b/2, b]$, such that the hypersphere centered at p with radius r intersects or is tangent to at most $2c$ edges of Q.*

$$F_b^{\text{before}} = \begin{pmatrix} \ldots & & & & & \\ 0 & 0 & \boxed{0} & \boxed{0} & \boxed{0} & \ldots \\ 0 & \boxed{0} & 1 & 1 & \boxed{0} & \ldots \\ 0 & 1 & 1 & \boxed{0} & 0 & \ldots \\ \boxed{0} & 1 & 1 & \boxed{0} & \boxed{0} & \ldots \\ 1 & 1 & 1 & 1 & 1 & \ldots \end{pmatrix} \qquad F_{b/2}^{\text{after}} = \begin{pmatrix} \ldots & & & & & \\ 0 & 0 & \textcircled{0} & \textcircled{0} & \textcircled{0} & \ldots \\ 0 & \textcircled{0} & 1 & 1 & \textcircled{0} & \ldots \\ 0 & 1 & 1 & \textcircled{0} & 0 & \ldots \\ \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \ldots \\ 1 & 1 & 1 & 1 & 1 & \ldots \end{pmatrix}$$

Figure 1 The elements of a guarding set (marked with boxes) before (left) and after (right) applying the trimming operation to the second row. The removed pairs are marked by circles

We call a pair $(i, j) \in B$ avoidable if there are two traversals of P and Q which guarantee that the pair (i, j) can be removed from the guarding set. Lemma 3.4 describes the algorithm to obtain a 4-guarding set whose size will be at most $(3c + 4) \cdot t$.

We omit discussion of our lower bounds due to space constraints.

▶ **Lemma 3.4.** *Let B be a 1-guarding set.*

(i) After the first phase of the algorithm, which removes all avoidable pairs, the modified set B is a 1-guarding set.

(ii) After the second phase of the algorithm, which applies the trimming operation to each row with $b = d_F(P, Q)$, the modified set B is a 2-guarding set.

(iii) After the third phase of the algorithm, which applies the trimming operation to each column with $b = d_F(P, Q)/2$, the modified set B is a 4-guarding set.

Acknowledgements. We want to thank Kevin Buchin for useful discussions on the topic of this paper.

References

1 P. Afshani and A. Driemel. On the complexity of range searching among curves. In *SODA*, pages 898–917, 2018.

2 P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal of Computing*, 43(2):429–449, 2014.

3 P. K. Agarwal, K. Fox, J. Pan, and R. Ying. Approximating Dynamic Time Warping and Edit Distance for a Pair of Point Sequences. In *SoCG*, volume 51, pages 6:1–6:16, 2016.

4 A. Backurs and A. Sidiropoulos. Constant-distortion embeddings of Hausdorff metrics into constant-dimensional l_p spaces. In *APPROX/RANDOM*, pages 1:1–1:15, 2016.

5 Y. Bartal, L. Gottlieb, and O. Neiman. On the impossibility of dimension reduction for doubling subsets of lp. In *SoCG*, pages 60–66, 2014.

6 K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *FOCS*, pages 661–670, 2014.

7 K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS*, pages 79–97, 2015.

8 K. Bringmann and M. Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *IJCGA*, 27(1-2):85–120, 2017.

9 K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog-with an application to Alt's conjecture. In *SODA*, pages 1399–1413, 2014.

10 K. Buchin, J. Chun, M. Löffler, A. Markovic, W. Meulemans, Y. Okamoto, and T. Shiitada. Folding free-space diagrams: Computing the Fréchet distance between 1-dimensional curves (multimedia contribution). In *SoCG*, pages 64:1–64:5, 2017.

11 A. Driemel and S. Har-Peled. Jaywalking your dog – computing the Fréchet distance with shortcuts. *SIAM Journal of Computing*, 42(5):1830–1866, 2013.

12 A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near-linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

13 A. Driemel, A. Krivošija, and C. Sohler. Clustering time series under the Fréchet distance. In *SODA*, pages 766–785, 2016.

14 A. Driemel and F. Silvestri. Locally-sensitive hashing of curves. In *SoCG*, pages 37:1–37:16, 2017.

15 T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory, 1994.

16 O. Gold and M. Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. In *ICALP*, pages 25:1–25:14, 2017.

17 P. Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *SoCG*, pages 102–106, 2002.

Augmenting a tree to a k-arbor-connected graph with pagenumber k

Toru Hasunuma[1]

1 **Tokushima University**
hasunuma@tokushima-u.ac.jp

Abstract

A tree is one of the most fundamental structures of graphs and has good properties on layouts, while it is weak from a fault-tolerant point of view. Motivated by these points of view, we consider an augmentation problem for a tree to increase fault-tolerance while preserving its good property on book-embeddings. A k-arbor-connected graph is a graph which has k spanning trees such that for any two vertices, the k paths between them in the k spanning trees are pairwise edge-disjoint and internally vertex-disjoint. We show that any tree with n vertices can be augmented in $O(nk)$ time to a minimum k-arbor-connected graph with pagenumber k for any k at most the radius of the tree. Our result is optimal for both the number of added edges and the number of pages for a book-embedding of a resultant graph. Besides, we extend our augmentation for trees to cacti.

1 Introduction

Throughout the paper, a graph means a simple undirected graph. Let $G = (V, E)$ be a graph. An augmentation problem for a graph is to find a set E' of pairs of non-adjacent vertices in G such that the augmented graph $G' = (V, E \cup E')$ satisfies a given condition.

A *book* is a structure consisting of a line called the *spine* and half planes called *pages* sharing the spine as a common boundary. A *k-page book-embedding* of G is defined by an assignment of the vertices of G to distinct points on the spine, i.e., a vertex-ordering σ of $V(G)$, and an assignment of the edges of G to pages such that no two edges assigned to the same page cross, where two edges uv and xy cross under σ if $\sigma(u) < \sigma(x) < \sigma(v) < \sigma(y)$. The *pagenumber* $\mathrm{pn}(G)$ of G is the minimum number of pages for a book-embedding of G. Book-embeddings have applications to VLSI layouts and there are many results on the subject until now (e.g., see [1, 2, 4]). In particular, one of the most famous results on book-embeddings is that every planar graph can be embedded in 4 pages [15].

Let $T_1, T_2, \ldots, T_k$ be spanning trees in G. If for any two vertices of G, the k paths between them in $T_1, T_2, \ldots, T_k$ are pairwise edge-disjoint and internally vertex disjoint, then $T_1, T_2, \ldots, T_k$ are *completely independent spanning trees* in G. Completely independent spanning trees can be applied to fault-tolerant communication problems, e.g., fault-tolerant broadcasting problems, since by deleting any $k-1$ vertices, at least one of the k completely independent spanning trees keeps its connectedness. We define the *arbor-connectivity* of G as the maximum number $\tau(G)$ of completely independent spanning trees in G, and G is *k-arbor-connected* if $\tau(G) \geq k$. So far, arbor-connectedness of graphs has been studied for graph classes related to interconnection networks (e.g., see [3, 5, 12]). It has also been shown that every maximal 4-connected planar graph is 2-arbor-connected [7], and G is $\lfloor \frac{n}{k} \rfloor$-arbor-connected if the minimum degree of G is at least $n - k$ where $3 \leq k \leq \frac{n}{2}$ [8]. Although any k-arbor-connected graph is k-vertex-connected, it has been proved [13] that there is no direct relationship between the vertex-connectivity and the arbor-connectivity; for any $k \geq 2$, there exists a k-vertex-connected graph G with $\tau(G) = 1$. From an algorithmic point of view, it has been shown that the problem of deciding whether a given graph is 2-arbor-connected is NP-complete [7].

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

A tree is one of the most fundamental structures of graphs and has good properties on layouts, e.g., the pagenumber of a tree is one. On the other hand, a tree is weak from a fault-tolerant point of view since it can be disconnected by deleting only one vertex. Motivated by these points of view, we consider an augmentation problem for a tree to increase fault-tolerance while preserving its good property on book-embeddings. On connectivity augmentation of graphs with geometric constraints, there are many results until now (see [9]). In particular, Kant and Bodlaender [10] have shown that the planarity-preserving minimum 2-vertex-connectivity augmentation problem is NP-hard, and Rutter and Wolff [14] have proved that the corresponding 2-edge-connectivity version is also NP-hard.

In this paper, we show that any tree T can be augmented in $O(nk)$ time to a minimum k-arbor-connected graph T^* which can be embedded in k pages for any k at most the radius of T. Every graph with n vertices and m edges needs at least $\lceil \frac{m-n}{n-3} \rceil$ pages for its book-embedding, which follows from the fact that a graph with pagenumber one is outerplanar [2]. This means that any k-arbor-connected graph cannot be embedded in $k-1$ pages, i.e., the pagenumber of T^* is determined to be k. Thus, our augmentation result is optimal for both the number of added edges and the number of pages for a book-embedding of a resultant augmented graph. Our augmented graph T^* also has a property that T^* is decomposed into completely independent spanning trees $T_1, T_2, \ldots, T_k$ such that each T_i can be embedded in one page under the same vertex-ordering. Besides, we extend our augmentation for trees to cacti and present an augmentation result for cycles.

2 Preliminaries

Given a set F of edges, the graph induced by F is denoted by $\langle F \rangle$, i.e., $V(\langle F \rangle) = \{u \mid uv \in F\}$ and $E(\langle F \rangle) = F$. The *distance* $d_G(u, v)$ of vertices u and v in a connected graph G is the length of a shortest path between u and v. The *eccentricity* $e_G(w)$ of a vertex w in G is $\max_{v \in V(G)} d_G(w, v)$. The *diameter* $\mathrm{diam}(G)$ of G is $\max_{w \in V(G)} e_G(w)$ and the *radius* $\mathrm{rad}(G)$ of G is $\min_{w \in V(G)} e_G(w)$. A *central vertex* of G is a vertex v with $e_G(v) = \mathrm{rad}(G)$. The *center* of G is the set of central vertices of G. Let T be a tree rooted at a vertex r. The *ℓ-ancestor* $p_\ell(v)$ of a vertex v in T is a vertex w which is on the path from r to v such that $d_T(v, w) = \ell$. If w is the ℓ-ancestor of v, then v is an *ℓ-descendant* of w. The set of ℓ-descendants of w is denoted by $D_\ell(w)$. The *lowest common ancestor* $\mathrm{lca}_T(u, v)$ of u and v in T is a common ancestor w of u and v in T such that there is no descendant of w which is a common ancestor of u and v. The *height* $h(T)$ of T is $\max_{v \in V(T)} d_T(r, v)$. A *leaf* of T is a vertex with degree one, while an *internal vertex* of T is a vertex with degree greater than one. The set of internal vertices in T is denoted by $V_I(T)$. A *star* is a tree in which there exists at most one internal vertex. A *cut-vertex* of G is a vertex v such that the graph obtained from G by deleting v is disconnected. A *block* of G is a maximal subgraph of G without a cut-vertex. A *cactus* is a graph whose every block is either a cycle or the complete graph with two vertices. A *cycle edge* of a cactus is an edge on a cycle. A *unicyclic graph* G is a graph with exactly one cycle and the cycle is denoted by $C(G)$.

Let σ be a vertex-ordering of G, i.e., a bijection from $V(G)$ to $\{1, 2, \ldots, |V(G)|\}$. When $\sigma(u) < \sigma(v)$, we simply write $u <_\sigma v$. For $u, v \in S \subseteq V(G)$, if $u <_\sigma v$ such that there is no vertex $w \in S$ with $u <_\sigma w <_\sigma v$, then u and v are *consecutive* in S under σ and we write $u \lessdot_{\sigma,S} v$. When $S = V(G)$, we may write $u \lessdot_\sigma v$. In order to construct completely independent spanning trees, we use a characterization shown in [6]; spanning trees $T_1, T_2, \ldots, T_k$ in G are completely independent if and only if $E(T_i) \cap E(T_j) = \emptyset$ and $V_I(T_i) \cap V_I(T_j) = \emptyset$ for any $1 \leq i < j \leq k$.

3 Results

▶ **Theorem 3.1.** *Any tree T with n vertices can be augmented to a minimum k-arbor-connected graph with pagenumber k for any $2 \leq k \leq \mathrm{rad}(T)$ in $O(nk)$ time.*

Proof. If T has two central vertices, then let x and y be the central vertices of T. Note that $xy \in E(T)$. Otherwise, let x be the central vertex of T and let y a vertex adjacent to x such that y is on a path between x and a vertex v with $d_T(x,v) = \mathrm{rad}(T)$. Let T^+ be the tree obtained from T by adding a new vertex z, joining it to x and y, and deleting the edge xy. In what follows, ancestors and descendants of a vertex are defined based on T^+ rooted at z unless otherwise stated. For any vertex u in T^+, T_u denotes the subtree rooted at u in T^+. By the definitions of x and y, it holds that $h(T_x) = \mathrm{rad}(T) \geq h(T_y) \geq \mathrm{rad}(T) - 1$.

Regarding the vertex z as the root of T^+, compute a depth-first-search ordering $\sigma^+ : V(T^+) \mapsto \{1, 2, \ldots, n+1\}$, where $\sigma^+(z) = 1$. Then, let $\sigma : V(T) \mapsto \{1, 2, \ldots, n\}$ be the vertex-ordering of T defined to be $\sigma(v) = \sigma^+(v) - 1$. Now let $V_i = D_{i+1}(z)$ for $0 \leq i < \mathrm{rad}(T)$. Also, let $W_\ell = \bigcup_{i \bmod k = \ell} V_i$ for each $0 \leq \ell < k$. We first divide $E(T) - \{xy\}$ into k subsets $E_1, E_2, \ldots, E_k$ defined as follows: for each $1 \leq i \leq k$,

- $E_i = \{vw \mid v \in W_{i-1},\ w \in D_1(v)\}$.

The set of added edges in our augmentation is divided into three types defined as follows: for each $1 \leq i \leq k$,

- $A_i = \{vw \mid v \in W_{i-1},\ w \in D_j(v),\ 2 \leq j \leq k\}$,
- $B_i = \{uw \mid u, v \in V_{i-1},\ u \lessdot_{\sigma, V_{i-1}} v,\ \sigma^{-1}(\max_{u' \in V(T_u)} \sigma(u')) <_\sigma w \leq_\sigma v\}$,
- $B'_i = \{wv \mid v = \sigma^{-1}(\max_{u' \in V_{i-1}} \sigma(u')),$
  $w <_\sigma \sigma^{-1}(\min_{u' \in V_{i-1}} \sigma(u'))$ or $\sigma^{-1}(\max_{u' \in V(T_v)} \sigma(u')) <_\sigma w\}$.

Note that $B_1 = \{xy\}$. Based on these sets, we define $T_1, T_2, \ldots, T_k$ as $T_i = \langle E_i \cup A_i \cup B_i \cup B'_i \rangle$ for $1 \leq i \leq k$. We then show that $T_1, T_2, \ldots, T_k$ are completely independent spanning trees in $T^* = T_1 \cup T_2 \cup \cdots \cup T_k$ such that each T_i can be embedded in one page under σ, which implies that the augmented graph $T^* \supseteq T$ is a minimum k-arbor-connected graph with pagenumber k.

The graph $\langle E_i \rangle$ is a disjoint union of stars whose central vertices are in W_{i-1}. The augmented graph $\langle E_i \cup A_i \rangle$ is a disjoint union of $|V_{i-1}|$ trees, each of which is obtained from the stars in $\langle E_i \rangle$ by joining each vertex in W_{i-1} and all its ℓ-descendants for $2 \leq \ell \leq k$. Thus, $V(\langle E_i \cup A_i \rangle) = V(T) - \cup_{0 \leq j < i-1} V_j$. The $|V_{i-1}|$ subtrees are connected by the edges uv for $u \lessdot_{\sigma, V_{i-1}} v$ in B_i, and moreover all the vertices in $\cup_{0 \leq j < i-1} V_j$ are joined to a vertex in V_{i-1} by other edges in $B_i \cup B'_i$. Therefore, $\langle E_i \cup A_i \cup B_i \cup B'_i \rangle$ is a tree with vertex set $V(T)$. Note that any edge in $B_i \cup B'_i$ joins a vertex w in $\cup_{0 \leq j < i-1} V_j$ and a vertex in V_{i-1} which is not a descendant of w. In each T_i, every vertex in $V(T) - W_{i-1}$ is directly joined to a vertex in W_{i-1} which means that every vertex in $V(T) - W_{i-1}$ is a leaf of T_i and $V_I(T_i) \subseteq W_{i-1}$. Since $W_i \cap W_j = \emptyset$ for any $0 \leq i < j < k$, $V_I(T_i) \cap V_I(T_j) = \emptyset$ for any $1 \leq i < j \leq k$. Now assume that $e = uv \in E(T_i) \cap E(T_j)$ for some $i < j$. Then, e is incident to a vertex in W_{i-1} and a vertex in W_{j-1}. If $uv \in B_i \cup B'_i$, then $u \in V_{i-1}$ and v must be in V_ℓ where $0 \leq \ell < i$ which is a contradiction. Thus, $uv \in A_i$ such that $u \in V_{kt+i-1}$, $v \in V_{kt+j-1}$ for some $t \geq 0$. This means that u is an ancestor of v. However, no ancestor of v is joined to v as a leaf of T_j. Therefore, $E(T_i) \cap E(T_j) = \emptyset$ for any $1 \leq i < j \leq k$. Consequently, $T_1, T_2, \ldots, T_k$ are completely independent spanning trees.

The graph $\langle E_i \cup A_i \rangle$ is a disjoint union of $|V_{i-1}|$ trees $S_1, S_2, \ldots, S_{|V_{i-1}|}$ such that the vertex set of each S_i corresponds to the vertex set of a subtree rooted at a vertex in V_{i-1}. From a property of a depth-first-search, for any subtree, the vertices in the subtree are consecutive in $V(T)$ under σ. Thus, it can be inductively shown (on the height) that S_i can

be embedded in one page under σ. All the vertices in $\cup_{0\leq j<i-1}V_j$ are isolated in $\langle E_i \cup A_i\rangle$ such that no vertex in $\cup_{0\leq j<i-1}V_j$ is placed between vertices of any subtree S_i. Hence, no crossing of edges is produced by adding the edges in $B_i \cup B'_i$ to $\langle E_i \cup A_i\rangle$. Therefore, each T_i can be embedded in one page under the same vertex-ordering σ.

The vertices x and y can be computed in linear time by applying a breadth-first-search twice. The vertex ordering σ follows from σ^+ which is obtained by applying a depth-first-search to T^+ from z. In the depth-first-search, $p_1(v)$ and $\sigma^{-1}(\max_{u'\in V(T_v)}\sigma(u'))$ can also be found for each vertex v. By a depth-first-search, V_{i-1}, W_{i-1}, the ordering relation $\preccurlyeq_{\sigma,V_{i-1}}$, $\sigma^{-1}(\min_{u'\in V_{i-1}}\sigma(u'))$, and $\sigma^{-1}(\max_{u'\in V_{i-1}}\sigma(u'))$ can be computed in $O(n)$ time. Here, E_i and A_i can be rewritten by

- $E_i = \{p_1(v)v \mid v \notin \{x, y\}, v \in W_{i \bmod k}\}$,
- $A_i = \{p_j(v)v \mid v \in V(T),\ p_j(v) \in W_{i-1},\ 2 \leq j \leq k\}$.

For each vertex v and each $1 < j \leq k$, $p_j(v)$ can be computed in $O(k)$ time. Therefore, the sets E_i, A_i, B_i, and B'_i for $1 \leq i \leq k$ can be computed in $O(nk)$ time. ◀

When $k = 2$, a star is the only tree to which Theorem 3.1 cannot be applied. However, a star can be easily augmented to a minimum 2-arbor-connected graph with pagenumber 2 as follows. Let S_n be a star with vertex set $\{v_1, v_2, \ldots, v_n\}$ such that v_1 is the central vertex of S_n. We augment S_n to a wheel graph W_n by adding the edges in $\{v_iv_{i+1} \mid 2 \leq i < n\} \cup \{v_2v_n\}$. Let $E_1 = \{v_1v_i \mid 2 \leq i < n\} \cup \{v_2v_n\}$ and $E_2 = \{v_iv_{i+1} \mid 2 \leq i < n\} \cup \{v_1v_n\}$. Then, $\langle E_1\rangle$ and $\langle E_2\rangle$ are edge-disjoint spanning trees such that $V_I(\langle E_1\rangle) = \{v_1, v_2\}$ and $V_I(\langle E_2\rangle) = \{v_3, \ldots, v_n\}$. Thus, $\langle E_1\rangle$ and $\langle E_2\rangle$ are completely independent spanning trees, and therefore W_n is a minimum 2-arbor-connected graph. Employing the vertex-ordering σ defined as $v_1 <_\sigma v_n <_\sigma v_2 <_\sigma \ldots <_\sigma v_{n-1}$, each $\langle E_i\rangle$ can be embedded in one page under σ. Thus, we have the following corollary.

▶ **Corollary 3.2.** *Any tree can be augmented to a minimum 2-arbor-connected graph with pagenumber 2 in linear time.*

It follows from Corollary 3.2 that any tree can be augmented to a minimum planar 2-arbor-connected graph in linear time, since a graph with pagenumber 2 is planar.

We here remark that in the proof of Theorem 3.1 other constructions can be employed if we do not insist on the upper bound on k. Select a path P with $|V(P)| \geq 3$ and consider the $|V(P)|$ subtrees each of which is rooted at a vertex in P (instead of two subtrees T_x and T_y). Then, we can construct a minimum k-arbor-connected graph where k is at most the maximum j such that there exist two vertices in P both of which have a $(j-1)$-descendant. In fact, we employ such a construction to prove Theorem 3.5.

Given a graph G with a vertex-ordering σ of a t-page book-embedding of G, let P be the path with $V(P) = V(G)$ and $E(P) = \{\sigma^{-1}(i)\sigma^{-1}(i+1) \mid 1 \leq i < n, i \neq \lfloor\frac{n}{2}\rfloor\} \cup \{\sigma^{-1}(1)\sigma^{-1}(\lfloor\frac{n}{2}\rfloor+1)\}$. According to the proof of Theorem 3.1, we augment P to P^*. Then, $G \cup P^*$ is a k-arbor-connected graph with pagenumber at most $t+k$. From this observation, we have the following existential result.

▶ **Corollary 3.3.** *For any graph G with n vertices and for any $2 \leq k \leq \frac{n}{2}$, there exists a k-arbor-connected graph $G^* \supseteq G$ with $\mathrm{pn}(G^*) \leq \mathrm{pn}(G) + k$.*

Next, we extend Theorem 3.1 to the class of cacti.

▶ **Theorem 3.4.** *Any cactus G can be augmented to a minimum k-arbor-connected graph with pagenumber k for any $\lfloor\frac{\ell_G}{2}\rfloor + 1 \leq k \leq \mathrm{rad}(G)$ in $O(nk)$ time, where ℓ_G is the maximum length of a cycle in G.*

Proof. Let x be a central vertex of G and y a vertex adjacent to x such that y is on a path between x and a vertex v with $d_G(x,v) = \mathrm{rad}(G)$. Let G^+ be the graph obtained from $G - xy$ by adding a new vertex z with new edges xz and yz. Besides, let T^+ be a breadth-first-search tree of G^+ from z. For each cycle C in G, there is exactly one cycle edge in $E(C) \cap (E(G^+) - E(T^+))$ and we denote by $f(C)$ the cycle edge. Now let $V_i = D_{i+1}(z)$ for $0 \le i < \mathrm{rad}(G)$ and $W_\ell = \bigcup_{i \bmod k = \ell} V_i$ for each $0 \le \ell < k$. Consider a depth-first-search ordering σ^+ of the tree T^+ from z such that for any $f(C) = a_C b_C$ with $a_C <_{\sigma^+} b_C$,

- if $a_C \in V_i$, then $b_C \in V_i \cup V_{i-1}$,
- if $a_C <_{\sigma^+} v <_{\sigma^+} b_C$, then v is either a descendant of a_C or an ancestor of b_C.

Define σ as $\sigma(v) = \sigma^+(v) - 1$ for any $v \in V(G)$. Similarly to the proof of Theorem 3.1, we define E_i, A_i, B_i, B'_i and then let $T_i = \langle E_i \cup A_i \cup B_i \cup B'_i \rangle$ for $1 \le i \le k$. Consider a cycle edge $f(C) = a_C b_C$ with $a_C <_\sigma b_C$. If $a_C \in V_i$ where $i < k$, then the cycle edge is used in T_i since $f(C) \in \cup_{1 \le i \le k} B_i$ from the properties of σ^+. Note that if xy is on a cycle C', then $f(C') \in \cup_{1 \le i \le k} B_i$. Suppose that $a_C \in V_{kt+i}$ where $t \ge 1$ and $0 \le i < k$. Let r_C be the k-ancestor of a_C. Since $k \ge \lfloor \frac{\ell_C}{2} \rfloor + 1$, the subtree rooted at r_C contains the vertex $\mathrm{lca}(a_C, b_C)$. Let $M(C) = \{w \mid \sigma^{-1}(\max_{v \in V(T_{a_C})} \sigma(v)) <_\sigma w \le_\sigma b_C\}$. Note that $\{r_C w \mid w \in M(C)\} \subseteq A_i$. Replace $E(T_i)$ with $(E(T_i) - \{r_C w \mid w \in M(C)\}) \cup \{a_C w \mid w \in M(C)\}$. Let $T'_1, T'_2, \ldots, T'_k$ be the spanning trees obtained by doing the modification for each cycle edge $f(C)$. Since any edge in $\{a_C w \mid w \in M(C)\}$ is not used in $T_1 \cup T_2 \cup \cdots \cup T_k$, the resultant spanning trees are completely independent spanning trees such that their union contains G. Besides, from the second property of σ^+, each T'_i can be embedded in one page under σ.

It has been shown in [11] that the center of a cactus can be found in linear time. Thus, x (and also y) can be found in linear time. By applying a breadth-first-search to G^+ from z, we can find $f(C)$ for each cycle C and label the end-vertices so that $d_{G^+}(z, a_C) \ge d_{G^+}(z, b_C)$. Besides, we can find $\mathrm{lca}(a_C, b_C)$ and recognize all the edges of C in $O(k)$ time. Let a'_C (resp., b'_C) be the vertex adjacent to $\mathrm{lca}(a_C, b_C)$ on the path from $\mathrm{lca}(a_C, b_C)$ to a_C (resp., b_C). We then apply a depth-first-search in which for each cycle C, each edge $p_1(v)v$ on the path from a'_C to a_C is traversed as the last edge in $\{p_1(v)w \mid w \in D_1(p_1(v))\}$ for the search of $T_{a'_C}$ and just after the search of $T_{a'_C}$, the traversal proceeds through the path from b'_C to b_C. This depth-first-search generates σ^+ satisfying the above two properties in $O(n)$ time. For each cycle edge $f(C)$, the corresponding modification can be done in $O(k)$ time and the number of cycle edges is at most $\lfloor \frac{n-1}{2} \rfloor$. Therefore, we can obtain a minimum k-arbor-connected graph containing G with pagenumber k in $O(nk)$ time. ◀

Although Theorem 3.4 cannot be applied to cycles, we can show the following result.

▶ **Theorem 3.5.** *Any cycle C with n vertices can be augmented to a minimum k-arbor-connected graph with pagenumber k for any $2 \le k \le \frac{n}{2}$.*

Proof. Let T be the path obtained from C by deleting one edge ab of C. Suppose that n is even. Let x and y be the central vertices of T. Construct a minimum k-arbor-connected graph T^* according to the construction in the proof of Theorem 3.1. Let $q = \frac{n-2}{2} \bmod k$. Let $V_q = \{a', b'\}$ such that $d_T(a, a') = d_T(b, b') < \frac{n}{2}$. If $q \ne 0$, then by replacing the edges in $B_{q+1} \cup B'_{q+1}$ with the edges in $\{p_j(b')a \mid 1 \le j \le q\} \cup \{p_j(a')b \mid 1 \le j \le q\} \cup \{ab\}$ in T_{q+1}, we obtain a desired graph. Suppose that $q = 0$ and $x'x, xy, yy' \in E(T)$. Define T_4^+ as the tree obtained from T by deleting the edges $x'x, xy, yy'$ and adding the new vertex z with the edges zx', zx, zy, zy'. Based on T_4^+ instead of T^+ in the proof of Theorem 3.1, we construct $T_1, T_2, \ldots, T_k$ under the condition $x' <_\sigma x <_\sigma y <_\sigma y'$. Note that in this construction, $V_0 = \{x', x, y, y'\}$, $B_1 = \{x'x, xy, yy'\}$ and $a, b \in W_{k-1}$. By modifying T_k in a similar fashion for T_{q+1} in the case that $q \ne 0$, we have a desired graph. Suppose that n is odd and x is the center

of T such that $x_1x_2, x_2x, xx_3, x_3x_4 \in E(T)$. Let $r = \frac{n-3}{2} \bmod k$. Define T_3^+ (resp., T_5^+) as the tree obtained from T by deleting the edges x_2x, xx_3 (resp., $x_1x_2, x_2x, xx_3, x_3x_4$) and adding the new vertex z with the edges zx_2, zx, zx_3 (resp., $zx_1, zx_2, zx, zx_3, zx_4$). Similarly to the case that n is even and $q = 0$, we have the desired result by considering T_3^+ (resp., T_5^+) and modifying T_{r+1} (resp., T_k) if $r \neq 0$ (resp., $r = 0$). ◀

We finally remark that the constructions shown in Theorem 3.5 can be generalized to a unicyclic graph G for $2 \leq k \leq \frac{|V(C(G))|}{2}$ by additional discussions.

4 Conclusion

We have shown that any tree with n vertices can be augmented in $O(nk)$ time to a minimum k-arbor-connected graph with pagenumber k for any k at most the radius of the tree. Besides, we have extended the result to cacti and presented an augmentation result for cycles.

It would be interesting to consider augmentation problems for a tree to a minimum k-arbor-connected graph while preserving other good geometric properties of a tree.

Acknowledgments. The author is grateful to the anonymous reviewers for their helpful comments.

References

1 J. Balogh and G. Salazar, Book embeddings of regular graphs, *SIAM J. Discrete Math.* 29 (2015) 811–822.
2 F. Bernhart and P.C. Kainen, The book thickness of a graph, *J. Combin. Theory* Ser. B 27 (1979) 320–331.
3 B. Cheng, D. Wang, and J. Fan, Constructing completely independent spanning trees in crossed cubes, *Discrete Appl. Math.* 219 (2017) 100–109.
4 F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg, Embedding graphs in books: a layout problem with application to VLSI design, *SIAM J. Alg. Discr. Meth.* 8 (1987) 33–58.
5 B. Darties, N. Gastineau, and O. Togni, Completely independent spanning trees in some regular graphs, *Discrete Appl. Math.* 217 (2017) 163–174.
6 T. Hasunuma, Completely independent spanning trees in the underlying graph of a line digraph, *Discrete Math.* 234 (2001) 149–157.
7 T. Hasunuma, Completely independent spanning trees in maximal planar graphs, Lecture Notes in Comput. Sci. 2573, pp. 235–245, 2002, Springer.
8 T. Hasunuma, Minimum degree conditions and optimal graphs for completely independent spanning trees, Lecture Notes in Comput. Sci. 9538, pp. 260–273, 2016, Springer.
9 F. Hurtado and C. Tóth, Plane geometric graph augmentation: a generic perspective, In: Pach J. (eds) Thirty Essays on Geometric Graph Theory, Springer, 2013, pp. 327–354.
10 G. Kant and H. L. Bodlaender, Planar graph augmentation problems, Lecture Notes in Comput. Sci., 519, pp. 286-298, 1991, Springer.
11 Y.-F. Lan, Y.-L. Wang, and H. Suzuki, A linear-time algorithm for solving the center problem on weighted cactus graphs, *Inform. Process. Lett.* 71 (1999) 205–212.
12 K-J. Pai and J-M. Chang, Constructing two completely independent spanning trees in hypercube-variant networks, *Theor. Comput. Sci.* 652 (2016) 28–37.
13 F. Péterfalvi, Two counterexamples on completely independent spanning trees, *Discrete Math.* 312 (2012) 808–810.
14 I. Rutter and A. Wolff, Augmenting the connectivity of planar and geometric graphs, *Electronic Notes in Discrete Math.* 31 (2008) 53–56.
15 M. Yannakakis, Embedding planar graphs in four pages, *J. Comput. System Sci.* 38 (1989) 36–67.

1-Bend RAC Drawings of NIC-Planar Graphs in Quadratic Area

Steven Chaplick, Fabian Lipp, Alexander Wolff, and Johannes Zink

Lehrstuhl für Informatik I, Universität Würzburg.
`http://www1.informatik.uni-wuerzburg.de/en/staff/`

Abstract

A drawing of a graph is called *1-planar* if every edge is crossed at most once. A 1-planar drawing is called *independent-crossing planar* (*IC-planar*) if no two pairs of crossing edges share a vertex. A 1-planar drawing is called *near-independent-crossing planar* (*NIC-planar*) if any two pairs of crossing edges share at most one vertex. The 1-planar, NIC-planar, and IC-planar graphs are the graphs that admit a 1-planar, NIC-planar, and IC-planar drawing, respectively. The NIC-planar graphs are a subset of the 1-planar graphs and a superset of the IC-planar graphs, which are important *beyond-planar* graph classes. We constructively show that every n-vertex NIC-plane graph admits a NIC-planar drawing with only right-angle crossings (RAC) and at most one bend per edge on a grid of size $O(n) \times O(n)$. Our construction takes linear time. We also give an overview of the relationships between several classes of 1-planar and RAC graphs.

1 Introduction

In graph theory and graph drawing, *beyond-planar* graph classes have experienced increasing interest in recent years. A prominent example is the class of *1-planar graphs*, that is, graphs that admit a drawing where each edge is crossed at most once. 1-planar graphs were introduced by Ringel [13] in 1965; Kobourov et al. [11] surveyed them recently. Another example that has received considerable attention are *RAC_k graphs*, that is, graphs that admit a poly-line drawing where all crossings are at right angles and each edge has at most k bends. RAC_k graphs were introduced by Didimo et al. [7]. We investigate the relationships between (certain subclasses of) 1-planar graphs and RAC_k graphs that admit drawings on a polynomial-size grid. Known results and our contributions are summarized in Fig. 1.

Basic Terminology. A mapping Γ is called a *drawing* of the graph $G = (V, E)$ if each vertex $v \in V$ is mapped to a point in $\mathbb{R}^2$ and each edge $\{u, v\}$ is mapped to a simple open Jordan curve in $\mathbb{R}^2$ such that the endpoints of this curve are $\Gamma(u)$ and $\Gamma(v)$. For convenience, we will refer to the points and simple open Jordan curves of a drawing as vertices and edges. The topologically connected regions of $\mathbb{R}^2 \setminus \Gamma$ are called *faces* of Γ. The unbounded face of Γ is called *outer* face; the other faces are *inner* faces. An equivalence class of drawings of a graph G having the same set of faces and the same outer face is called an *embedding* of G.

A *k-bend (poly-line) drawing* is a drawing in which every edge is drawn as a connected sequence of at most $k + 1$ line segments. The up to k inner vertices of an edge connecting these line segments are called *bend points* or *bends*. A 0-bend drawing is called *straight-line*. A *drawing on the grid of size $w \times h$* is a drawing where every vertex, bend point, and crossing point has integer coordinates in the range $[0, w] \times [0, h]$. Recall that a drawing is *1-planar* if every edge is crossed at most once. A 1-planar drawing is called *independent-crossing planar* (*IC-planar*) if no two pairs of crossing edges share a vertex. A 1-planar drawing is called *near-independent-crossing planar* (*NIC-planar*) if any two pairs of crossing edges share at most one vertex. A drawing is called *right-angle-crossing* (*RAC*) if it is a poly-line drawing and for each crossing point c, there are at most two edges that cross in c, there is no bend point in c, and the line segments of the edges that cross in c intersect in a right angle. As

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

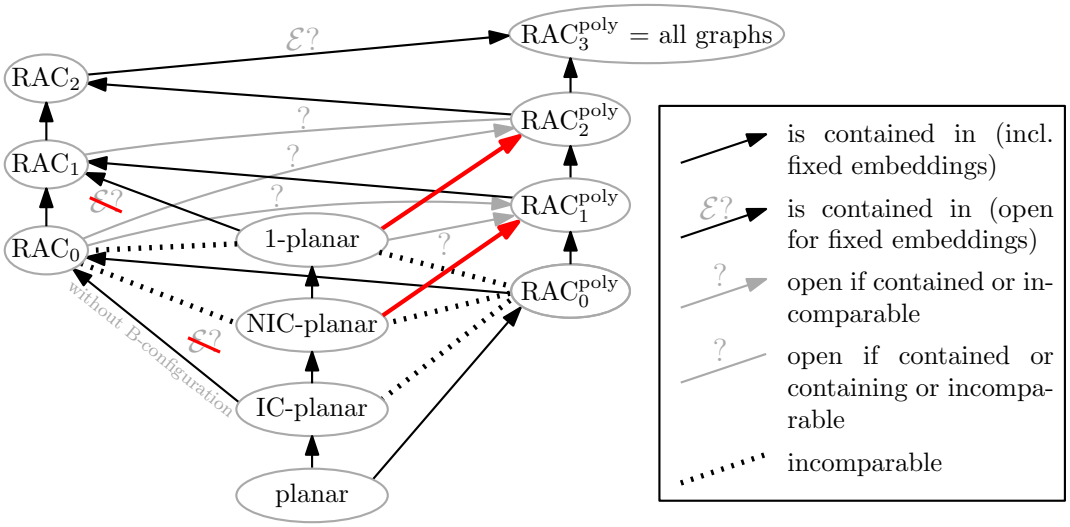


Figure 1 Relationships between (beyond-)planar graphs and RAC graphs. Our main results are the containment relationships indicated by the thick red arrows.

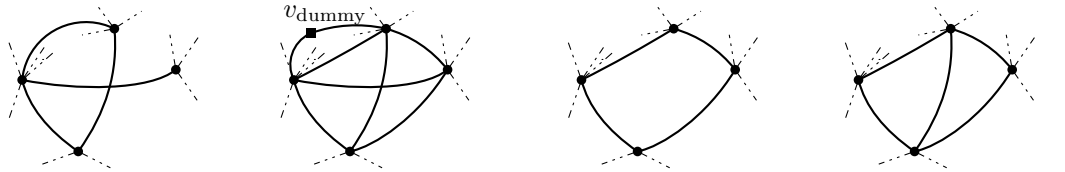
mentioned above, a drawing is called RAC$_k$ if it is RAC and k-bend. The *planar*, *1-planar*, *NIC-planar*, *IC-planar*, and *RAC$_k$* graphs are the graphs that admit a crossing-free, 1-planar, NIC-planar, IC-planar, and RAC$_k$ drawing, respectively. More specifically, RAC$_k^{poly}$ is the set of graphs that admit a RAC$_k$ drawing on a grid of size polynomial in the number of vertices. A *plane*, *1-plane*, *NIC-plane*, and *IC-plane* graph is a graph given with a specific planar, 1-planar, NIC-planar, and IC-planar embedding, respectively.

Previous Work. In the diagram in Fig. 1, we give an overview of the relationships between classes of 1-planar graphs and RAC$_k$ graphs. Clearly, the planar graphs are a subset of the IC-planar graphs, which are a subset of the NIC-planar graphs, which are a subset of the 1-planar graphs. It is well known that every plane graph can be drawn with straight-line edges on a grid of quadratic size [6, 14]. Every IC-planar graph admits an IC-planar RAC$_0$ drawing but not always in polynomial area [4]. Moreover, there are graphs in RAC$_0^{poly}$ that are not 1-planar [8] and, therefore, also not IC-planar. The class of RAC$_0$ graphs is incomparable with the classes of NIC-planar graphs [1] and 1-planar graphs [8]. Bekos et al. [2] showed that every 1-planar graph admits a 1-planar 1-bend RAC drawing, but their recursive drawings may need exponential area. Brandenburg [3] claimed that every 1-planar graph admits a 1-planar 1-bend RAC drawing where the vertices lie on a polynomial-size grid. In a private communication, he later retracted his claim—therefore, this question remains open. Every graph admits a RAC$_3$ drawing in polynomial area [7]. This does not hold if a given embedding of a planarization of the graph must be preserved [7].

Our Contributions. Our main result is as follows.

▶ **Theorem 1.** *Every n-vertex NIC-planar graph G admits a NIC-planar 1-bend RAC drawing on a grid of size $(16n-32) \times (8n-16)$. Given a NIC-planar embedding $\mathcal{E}$ of G, a drawing that has these characteristics and respects $\mathcal{E}$ can be computed in $O(n)$ time.*

For IC-plane graphs, this reduces the number of bends compared to a recent result of Liotta and Montecchiani [12] who showed that every IC-planar graph admits an IC-planar RAC$_2$ drawing on a grid of quadratic size. We have also shown (see Zink's master's thesis [15]) that every 1-plane graph admits a 1-planar RAC$_2$ drawing in polynomial area and, by a small



(a) crossing as it initially appears **(b)** empty kite and subdivided original edge **(c)** empty quadrangle **(d)** divided quadrangle

Figure 2 Modifications of crossings and computation of the biconnected canonical ordering.

modification of the algorithm by Bekos et al. [2], that not only every 1-planar, but even every 1-plane graph admits a 1-planar RAC_1 drawing. We can also show, by a small modification of the algorithm by Brandenburg et al. [4], that not only every IC-planar, but even every IC-plane graph without so called *B-configurations* admits a IC-planar RAC_0 drawing. Due to space considerations, we omit these results here.

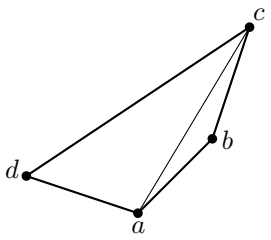
2 1-Bend RAC Drawings of NIC-Plane Graphs in Quadratic Area

Our algorithm takes an n-vertex NIC-plane graph $(G, \mathcal{E})$ as input and returns a NIC-planar RAC_1 drawing of G on a grid of size $O(n) \times O(n)$ while maintaining $\mathcal{E}$. We now describe the algorithm. We omit a formal correctness proof due to lack of space.

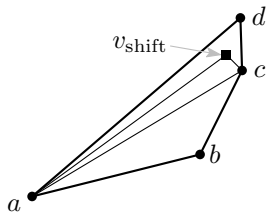
Preprocessing. We first aim to make the NIC-plane input graph biconnected and planar so that we can draw it using the algorithm by Harel and Sardas [9]. Around each crossing in $\mathcal{E}$, we insert up to four dummy edges to obtain *empty kites*. A *kite* is a K_4 that is embedded such that (i) every vertex lies on the boundary of the outer face, (ii) there is exactly one crossing, and (iii) this crossing doesn't lie on the boundary of the outer face. A kite K as a subgraph of a graph H is said to be *empty* if there is no edge of $H \backslash K$ that is on an inner face of K or crosses edges of K. Whenever we insert a dummy edge, we may create a pair of parallel edges. Then, we subdivide the original edge participating in this pair by a dummy vertex (see the transition from Fig. 2a to 2b). Note that we never create parallel dummy edges since G is NIC-planar. After this, we remove both crossing edges from each empty kite and obtain *empty quadrangles* (see Fig. 2c). We store each such empty quadrangle in a list Q. At the end of the preprocessing, we make the resulting plane graph biconnected via, e.g., the algorithm of Hopcroft and Tarjan [10]. Let $(G', \mathcal{E}')$ be the resulting plane biconnected graph.

Drawing Step. Now, we draw a graph that we obtain from $(G', \mathcal{E}')$. We use the algorithm by Harel and Sardas [9], which is a generalization of the algorithm of Chrobak and Payne [5], which in turn is based on the shift algorithm of de Fraysseix et al. [6]. The algorithm of Harel and Sardas consists of two phases. Given a plane biconnected graph H, in the first phase a biconnected canonical ordering Π of the vertices in the plane input graph is computed. In the second phase, H is drawn according to Π on a grid of size $(2|V(H)| - 4) \times (|V(H)| - 2)$. Biconnected canonical orderings are a generalization of canonical orderings that assume only biconnectivity (instead of triconnectivity). Unlike the classical shift algorithm, the algorithm of Harel and Sardas computes the (biconnected) canonical ordering *bottom-up*, which we will exploit here. Let $\Pi_k = (v_1, \ldots, v_k)$ be a partial biconnected canonical ordering of H after step k, and let H_k be the plane subgraph of H induced by Π_k. We say that a vertex u is *covered* by v_k if u is on the boundary of the outer face of H_{k-1}, but not on that of H_k.

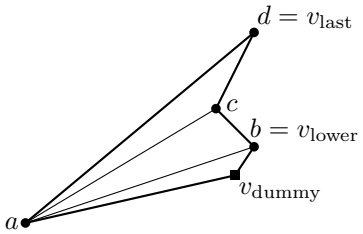
We perform the following additional operations when we compute the biconnected canonical ordering. Whenever we reach an empty quadrangle $q = (a, b, c, d)$ in the list Q for



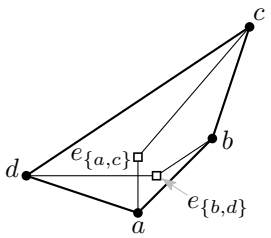
(a) divided quadrangle in Case 1; $v_{\text{last}} = c$



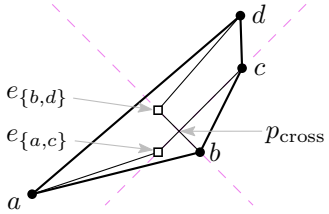
(b) divided quadrangle in Case 2; $v_{\text{last}} = d$ and b is covered by c



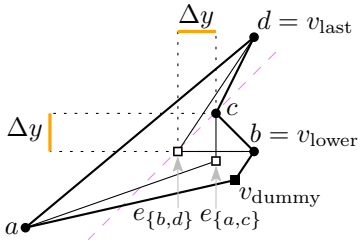
(c) divided quadrangle in Case 3; $v_{\text{last}} = d$ and b is not covered by c



(d) Case 1 after inserting the crossing edges



(e) Case 2 after inserting the crossing edges



(f) Case 3 after inserting the crossing edges

Figure 3 The three cases of the drawing step (a)–(c) and the reinsertion step (d)–(f) in our algorithm. For orientation, lines with slope 1 or -1 are dashed violet.

the first time, i.e., when the first vertex of q—say a—is added to the biconnected canonical ordering, we insert an edge inside q from a to the vertex opposite a in q, that is, to c. We call the resulting structure a *divided quadrangle* (see Fig. 2d). In two special cases, we perform further modifications of the graph. They will help us to guarantee a correct reinsertion of the crossing edges in the next step of the algorithm. Namely, when we encounter the last vertex $v_{\text{last}} \in \{b, c, d\}$ of q, we distinguish three cases.

Case 1: $v_{\text{last}} = c$ (see Fig. 3a).

In this case, we perform no extra operation.

Case 2: $v_{\text{last}} \in \{b, d\}$, and the other of these two vertices is covered by c (see Fig. 3b).

We insert a dummy vertex v_{shift}, which we call *shift vertex*, into the current biconnected canonical ordering directly before v_{last} and make it adjacent to a and c. Later, we will remove v_{shift}, but for now it forces the algorithm of Harel and Sardas to shift a and c away from each other before v_{last} is added.

Case 3: $v_{\text{last}} \in \{b, d\}$, and neither b nor d is covered by c (see Fig. 3c).

Let $\{v_{\text{lower}}\} = \{b, d\} \setminus v_{\text{last}}$. We subdivide the edge $\{a, v_{\text{lower}}\}$ via a dummy vertex v_{dummy}. If $\{a, v_{\text{lower}}\}$ is an original edge of the input graph, this edge will be bent at v_{dummy} in the final drawing. We insert v_{dummy} into the current biconnected canonical ordering directly before v_{lower}. To obtain a divided quadrangle again, we insert the dummy edge $\{a, v_{\text{lower}}\}$, which we will remove before we reinsert the crossing edges. This will give us some extra space inside the triangle $(a, v_{\text{dummy}}, v_{\text{lower}})$ for a bend point.

We draw the resulting plane biconnected $\hat{n}$-vertex graph $(\hat{G}, \hat{\mathcal{E}})$ according to its biconnected canonical ordering $\hat{\Pi}$ via the algorithm by Harel and Sardas and obtain a crossing-free drawing $\hat{\Gamma}$. We do not modify the actual drawing phase.

Postprocessing (Reinserting the Crossing Edges). We refine the underlying grid of $\hat{\Gamma}$ by a factor of 2 in both dimensions. Let $q = (a, b, c, d)$ be a quadrangle in Q, where a is the

first and v_{last} the last vertex in $\hat{\Pi}$ among the vertices in q. From q, we first remove the chord edge $\{a, c\}$ and obtain an empty quadrangle. Then, we distinguish three cases for reinserting the crossing edges that we removed in the preprocessing. These are the same cases as in the description of the modified computation of the biconnected canonical ordering above.

Case 1: $v_{\text{last}} = c$ (see Fig. 3a).

Since c is adjacent to a, b, and d in $\hat{G}$, it has the largest y-coordinate among the vertices in q. Assume that $y(d)$ is smaller or equal to $y(b)$ since the other case is symmetric. An example of a quadrangle in this case before and after the reinsertion of the crossing edges is given in Figs. 3a and 3d, respectively. We will have a crossing point at $(x(a), y(d))$. To this end, we insert the edge $\{a, c\}$ with a bend at $e_{\{a,c\}} = (x(a), y(d) + 1)$ and we insert the edge $\{b, d\}$ with a bend at $e_{\{b,d\}} = (x(a) + 1, y(d))$.

Case 2: $v_{\text{last}} \in \{b, d\}$, and the other of these two vertices is covered by c (see Fig. 3b).

Assume that $y(d) > y(b)$; the other case is symmetric. An example of a quadrangle in this case before and after the reinsertion of the crossing edges is given in Figs. 3b and 3e, respectively. Here, we remove v_{shift} in addition to removing the edge $\{a, c\}$. We define the crossing point $p_{\text{cross}} = (x_{\text{cross}}, y_{\text{cross}})$ as the intersection point of the lines with slope 1 and -1 through c and b, respectively. The coordinates of this crossing point are $x_{\text{cross}} = (x(c) - y(c) + x(b) + y(b))/2$ and $y_{\text{cross}} = (-x(c) + y(c) + x(b) + y(b))/2$. Despite the fact that both coordinates are the result of a division by 2, both are integers—recall that we refined the grid by a factor of 2 in each dimension. We place the two bend points onto the same lines at the closest grid points that are next to p_{cross}. In other words, we draw the edge $\{a, c\}$ with a bend point at $e_{\{a,c\}} = (x_{\text{cross}} - 1, y_{\text{cross}} - 1)$ and we insert the edge $\{b, d\}$ with a bend point at $e_{\{b,d\}} = (x_{\text{cross}} - 1, y_{\text{cross}} + 1)$. We do not intersect or touch the edge $\{a, d\}$ because we shifted a far enough away from c by the extra shift due to v_{shift}. Moreover, the points $e_{\{a,c\}}$ and p_{cross} on the line with slope 1 through c are inside the empty quadrangle q since b is covered by c (then b is below the line with slope 1 through c) and $y(b)$ is at most equal to $y(e_{\{a,c\}})$.

Case 3: $v_{\text{last}} \in \{b, d\}$, and neither b nor d is covered by c (see Fig. 3c).

Assume that $y(d) > y(b)$; again, the other case is symmetric. An example of a quadrangle in this case before and after the reinsertion of the crossing edges is given in Figs. 3c and 3f, respectively. Note that the edge $\{a, b\}$ is the dummy edge which we inserted during the computation of $\hat{\Pi}$ and next to this edge, there is the path $(a - v_{\text{dummy}} - b)$. This path is the former edge $\{a, b\}$. We will reinsert the edges $\{a, c\}$ and $\{b, d\}$ such that they cross in $(x(c), y(b))$. We will bend the edge $\{b, d\}$ on the line with slope 1 through c at $y = y(b)$ because from this point we always "see" d inside q. So, we define $x_{\text{bend}} := x(c) - \Delta y$ with $\Delta y := y(c) - y(b)$. First, we remove the dummy edge $\{a, b\}$. Second, we insert the edge $\{a, c\}$ with a bend point at $e_{\{a,c\}} = (x(c), y(b) - 1)$. Third, we insert the edge $\{b, d\}$ with a bend point at $e_{\{b,d\}} = (x_{\text{bend}}, y(b))$. Note that $e_{\{a,c\}}$ might be below the straight line segment $\overline{ab}$ since a could have been shifted away from c several times. However, $e_{\{a,c\}}$ cannot be on or below the path $(a - v_{\text{dummy}} - b)$ because $y(v_{\text{dummy}}) < y(e_{\{a,c\}})$ and the slope of the line segment $\overline{v_{\text{dummy}}b}$ is either greater than 1 or negative. Therefore, the crossing edges $\{a, c\}$ and $\{b, d\}$ lie completely inside the pentagon $(a, v_{\text{dummy}}, b, c, d)$.

After we have reinserted the crossing edges into each quadrangle of Q, we remove all dummy edges and transform the remaining dummy vertices to bend points. The resulting drawing Γ is a RAC_1 drawing that preserves the embedding of the NIC-plane input graph $(G, \mathcal{E})$. Our algorithm runs in linear time. Since the shift algorithm draws $\hat{\Gamma}$ on a grid of size $(2\hat{n} - 4) \times (\hat{n} - 2)$, which we refined by a factor of 2, and $\hat{n} \leq 4n - 6$, Γ lies on a grid of size at most $(16n - 32) \times (8n - 16)$.

3 Conclusion and Open Questions

We have presented an algorithm for drawing any NIC-plane graph on a small grid with right-angle crossings and at most one bend per edge. Our algorithm is based on the shift algorithm for 2-connected graphs by Harel and Sardas [9]. Before and while we execute their algorithm, we modify the graph (incl. removing the crossing edges) to obtain faces with nice properties into which we reinsert the crossing edges afterwards.

The diagram in Fig. 1 leaves some open questions. Does every 1-planar graph admit a 1-planar 1-bend RAC drawing in polynomial area? Can every graph in RAC_0 be drawn in polynomial area if we allow one or two bends per edge? What is the relationship between RAC_1 and $\mathrm{RAC}_2^{\mathrm{poly}}$?

References

1 Christian Bachmaier, Franz J. Brandenburg, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber. NIC-planar graphs. *Discrete Appl. Math.*, 232:23–40, 2017. doi:10.1016/j.dam.2017.08.015.

2 Michael A. Bekos, Walter Didimo, Giuseppe Liotta, Saeed Mehrabi, and Fabrizio Montecchiani. On RAC drawings of 1-planar graphs. *Theor. Comput. Sci.*, 689:48–57, 2017. doi:10.1016/j.tcs.2017.05.039.

3 Franz J. Brandenburg. Compact 1-bend RAC drawings of 1-planar graphs. In *Proc. 33rd Europ. Workshop Comput. Geom. (EuroCG'17)*, pages 129–132, 2017. URL: http://csconferences.mah.se/eurocg2017/proceedings.pdf.

4 Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani. Recognizing and drawing IC-planar graphs. *Theor. Comput. Sci.*, 636:1–16, 2016. doi:10.1016/j.tcs.2016.04.026.

5 Marek Chrobak and Thomas H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Inf. Process. Lett.*, 54(4):241–246, 1995. doi:10.1016/0020-0190(95)00020-D.

6 Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. doi:10.1007/BF02122694.

7 Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. *Theor. Comput. Sci.*, 412(39):5156–5166, 2011. doi:10.1016/j.tcs.2011.05.025.

8 Peter Eades and Giuseppe Liotta. Right angle crossing graphs and 1-planarity. *Discrete Appl. Math.*, 161(7-8):961–969, 2013. doi:10.1016/j.dam.2012.11.019.

9 David Harel and Meir Sardas. An algorithm for straight-line drawing of planar graphs. *Algorithmica*, 20(2):119–135, 1998. doi:10.1007/PL00009189.

10 John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.

11 Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. arXiv report, 2017. URL: http://arxiv.org/abs/1703.02261.

12 Giuseppe Liotta and Fabrizio Montecchiani. L-visibility drawings of IC-planar graphs. *Inf. Process. Lett.*, 116(3):217–222, 2016. doi:10.1016/j.ipl.2015.11.011.

13 Gerhard Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 29(1–2):107–117, 1965.

14 Walter Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Symp. Discrete Algorithms (SODA'90)*, pages 138–148, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.

15 Johannes Zink. 1-planar RAC drawings with bends. Master's thesis, Institut für Informatik, Universität Würzburg, 2017. Under Review. URL: http://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2017-zink-master.pdf.

Stabbing Pairwise Intersecting Disks by Five Points*

Sariel Har-Peled[1], Haim Kaplan[2], Wolfgang Mulzer[3], Liam Roditty[4], Paul Seiferth[3], Micha Sharir[2], and Max Willert[3]

1 **Department of Computer Science, University of Illinois, Urbana, USA**
sariel@illinois.edu
2 **School of Computer Science, Tel Aviv University, Israel**
{haimk,michas}@post.tau.ac.il
3 **Department of Computer Science, Freie Universität Berlin, Germany**
{mulzer,pseiferth,willerma}@inf.fu-berlin.de
4 **Department of Computer Science, Bar Ilan University, Israel**
liamr@macs.biu.ac.il

Abstract

We present a deterministic linear time algorithm to find a set of five points that stab a set of n pairwise intersecting disks in the plane. We also give a simple construction with 13 pairwise intersecting disks that cannot be stabbed by three points.

1 Introduction

Let $\mathcal{D}$ be a set of n pairwise intersecting disks in the plane. If every three disks in $\mathcal{D}$ have a nonempty intersection, then, by Helly's theorem, the whole intersection $\cap\mathcal{D}$ is nonempty [6–8]. Thus, $\mathcal{D}$ can be *stabbed* by one point. More generally, when there are three disks with empty intersection, $\mathcal{D}$ can still always be stabbed by four points. In July 1956, Danzer presented a proof at Oberwolfach (see [3]). Since Danzer was not satisfied with his proof, he never published it, but he gave a new proof in 1986 [3]. Previously, in 1981, Stachó published a proof for the existence of four stabbing points [11], using similar arguments as in his previous construction of five stabbing points [10]. Hadwiger and Debrunner showed that three points suffice for unit disks [5]. Danzer's upper bound proof is fairly involved, and there seems to be no obvious way to turn it into an efficient algorithm. The two constructions of Stachó are simpler, but not enough for an easy subquadratic algorithm. We present a new argument that yields five stabbing points. Our proof is constructive and allows us to find the stabbing points deterministically in linear time.

As for lower bounds, Grünbaum gave an example of 21 pairwise intersecting disks that cannot be stabbed by three points [4]. Later, Danzer reduced the number of disks to ten [3]. This example is close to optimal, because every set of eight disks can be stabbed by three points [10]. It is hard to verify the lower bound by Danzer for ten disks—even with dynamic geometry software. We present a simple construction that uses 13 disks.

2 The geometry of pairwise intersecting disks

Let $\mathcal{D}$ be a set of n pairwise intersecting disks. A disk $D_i \in \mathcal{D}$ is given by its center c_i and its radius r_i. We assume without loss of generality that no disk is contained in another. The *lens* of two disks $D_i, D_j \in \mathcal{D}$ is the set $L_{i,j} = D_i \cap D_j$. Let u be any of the two intersection points of ∂D_i

* Supported in part by grant 1367/2016 from the German-Israeli Science Foundation (GIF). W.M. supported in part by ERC StG 757609. P.S. and W.M. supported in part by DFG grant MU-3501/1.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

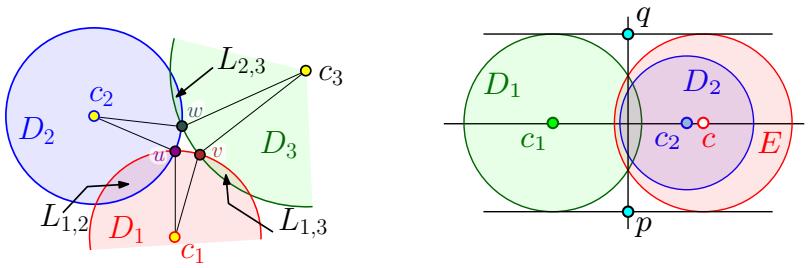


Figure 1 Left: At least one lens angle is large. **Right:** D_1 and E have the same radii and lens angle $2\pi/3$. By Lemma 2.2, D_2 is a subset of E. $\{c_1, c, p, q\}$ is the set P from Lemma 2.4.

and ∂D_j. The angle $\angle c_i u c_j$ is called the *lens angle* of D_i and D_j. It is at most π. Three disks D_i, D_j, and D_k are *non-Helly* if $D_i \cap D_j \cap D_k = \emptyset$. We present some useful geometric lemmas.

▶ **Lemma 2.1.** *Among any three non-Helly pairwise intersecting disks D_1, D_2, and D_3, there are two disks with lens angle larger than $2\pi/3$.*

Proof. By assumption, the lenses $L_{1,2}$, $L_{1,3}$ and $L_{2,3}$ are pairwise disjoint. Let u be the vertex of $L_{1,2}$ nearer to D_3, and let v, w be the analogous vertices of $L_{1,3}$ and $L_{2,3}$ (see Figure 1, Left). Consider the simple hexagon $c_1uc_2wc_3v$, and write $\angle u$, $\angle v$, and $\angle w$ for the interior angles at u, v, and w. The sum of all interior angles is 4π. Thus, $\angle u + \angle v + \angle w < 4\pi$, so at least one angle is less than $4\pi/3$. It follows that the exterior angle at u, v, or w must be larger than $2\pi/3$. ◀

▶ **Lemma 2.2.** *Let D_1 and D_2 be two intersecting disks with radii $r_1 \geq r_2$ and lens angle $\alpha \geq 2\pi/3$. Let E be the unique disk with radius r_1 and center c, such that (i) the centers c_1, c_2, and c are collinear and c lies on the same side of c_1 as c_2; and (ii) the lens angle of D_1 and E is exactly $2\pi/3$ (see Figure 1, Right). Then, if c_2 lies between c_1 and c, we have $D_2 \subseteq E$.*

Proof. Let $x \in D_2$. Then, since c_2 lies between c_1 and c, the triangle inequality gives

$$|xc| \leq |xc_2| + |c_2c| = |xc_2| + |c_1c| - |c_1c_2|. \tag{1}$$

Since $x \in D_2$, we get $|xc_2| \leq r_2$. Also, since D_1 and E each have radius r_1 and form the lens angle $2\pi/3$, it follows that $|c_1c| = \sqrt{3}r_1$. Finally, by the law of cosines, $|c_1c_2| = \sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos\alpha}$. As $\alpha \geq 2\pi/3$ and $r_1 \geq r_2$, we get $\cos\alpha \leq -0.5 \leq (\sqrt{3} - 1.5)\frac{r_1}{r_2} - \sqrt{3} + 1$, so

$$|c_1c_2|^2 = r_1^2 + r_2^2 - 2r_1r_2\cos\alpha \geq r_1^2 + r_2^2 - 2r_1r_2\left(\left(\sqrt{3} - 1.5\right)\frac{r_1}{r_2} - \sqrt{3} + 1\right) = \left(r_1\left(\sqrt{3} - 1\right) + r_2\right)^2.$$

Plugging this into Eq. (1), we obtain $|xc| \leq r_2 + \sqrt{3}r_1 - \left(r_1\left(\sqrt{3} - 1\right) + r_2\right) = r_1$, i.e., $x \in E$. ◀

▶ **Lemma 2.3.** *Let D_1 and D_2 be two intersecting disks of equal radius r with lens angle $2\pi/3$. There is a set P of four points so that any disk F of radius at least r that intersects both D_1 and D_2 contains a point of P.*

Proof. Consider the two tangent lines of D_1 and D_2, and let p and q be the midpoints on these lines between the respective two tangency points. We set $P = \{c_1, c_2, p, q\}$ (see Figure 2, Left).

Given F, we decrease its radius, keeping its center fixed, until either the radius becomes r or until F is tangent to D_1 or D_2. Suppose the latter case holds and F is tangent to D_1. We move the center of F continuously along the line spanned by the center of F and c_1 towards c_1, decreasing the radius of F to maintain the tangency. We stop when either the radius of F reaches

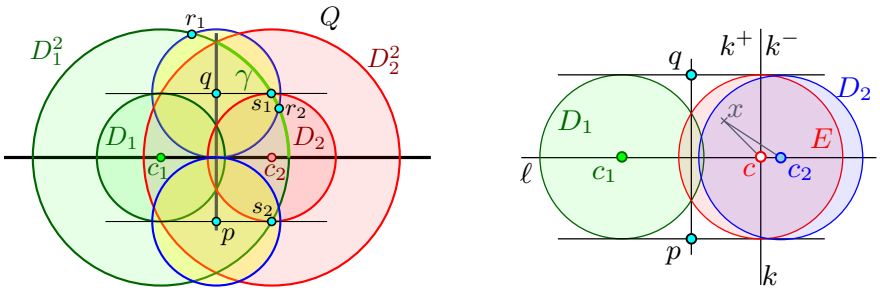


Figure 2 Left: $P = \{c_1, c_2, p, q\}$ is the stabbing set. The green arc $\gamma = \partial(D_1^2 \cap D_2^2) \cap Q$ is covered by $D_1^2 \cap D_q$. **Right:** Situation (ii) in the proof of Lemma 2.4: $D_2 \not\subseteq E$. x is an arbitrary point in $D_2 \cap F \cap k^+$. The angle at c in the triangle Δxcc_2 is $\geq \pi/2$.

r or F becomes tangent to D_2. We obtain a disk $G \subseteq F$ with center $c = (c_x, c_y)$ so that either: (i) radius$(G) = r$ and G intersects both D_1 and D_2, or (ii) radius$(G) \geq r$ and G is tangent to both D_1 and D_2. Since $G \subseteq F$, it suffices to show that $G \cap P \neq \emptyset$. We introduce a coordinate system, setting the origin o midway between c_1 and c_2, so that the y-axis passes through p and q. Then, in the manner depicted in Figure 2 (left), we have $c_1 = (-\sqrt{3}r/2, 0)$, $c_2 = (\sqrt{3}r/2, 0)$, $q = (0, r)$, and $p = (0, -r)$.

For case (i), let D_1^2 be the disk of radius $2r$ centered at c_1, and D_2^2 the disk of radius $2r$ centered at c_2. Since G has radius r and intersects both D_1 and D_2, its center c has distance at most $2r$ from both c_1 and c_2, i.e., $c \in D_1^2 \cap D_2^2$. Let D_p and D_q be the two disks of radius r centered at p and q. We will show that $D_1^2 \cap D_2^2 \subseteq D_1 \cup D_2 \cup D_p \cup D_q$. Then it is immediate that $G \cap P \neq \emptyset$. By symmetry, it is enough to focus on the upper-right quadrant $Q = \{(x, y) \mid x \geq 0, y \geq 0\}$. We show that all points in $D_1^2 \cap Q$ are covered by $D_2 \cup D_q$. Without loss of generality, we assume that $r = 1$. Then, the two intersection points of D_1^2 and D_q are $r_1 = (\frac{5\sqrt{3}-2\sqrt{87}}{28}, \frac{38+3\sqrt{29}}{28}) \approx (-0.36, 1.93)$ and $r_2 = (\frac{5\sqrt{3}+2\sqrt{87}}{28}, \frac{38-3\sqrt{29}}{28}) \approx (0.98, 0.78)$, and the two intersection points of D_1^2 and D_2 are $s_1 = (\frac{\sqrt{3}}{2}, 1) \approx (0.87, 1)$ and $s_2 = (\frac{\sqrt{3}}{2}, -1) \approx (0.87, -1)$. Let γ be the boundary curve of D_1^2 in Q. Since $r_1, s_2 \notin Q$ and since $r_2 \in D_2$ and $s_1 \in D_q$, it follows that γ does not intersect the boundary of $D_2 \cup D_q$ and hence $\gamma \subset D_2 \cup D_q$. Furthermore, the subsegment of the y-axis from o to the startpoint of γ is contained in D_q, and the subsegment of the x-axis from o to the endpoint of γ is contained in D_2. Hence, the boundary of $D_1^2 \cap Q$ lies completely in $D_2 \cup D_q$, and since $D_2 \cup D_q$ is simply connected, it follows that $D_1^2 \cap Q \subseteq D_2 \cup D_q$, as desired.

For case (ii), since G is tangent to D_1 and D_2, the center c of G is on the perpendicular bisector of c_1 and c_2, so the points p, o, q and c are collinear. Suppose without loss of generality that $c_y \geq 0$. Then, it is easily checked that c lies above q, and radius$(G) + r = |c_1 c| \geq |oc| = r + |qc|$, so $q \in G$. ◀

▶ **Lemma 2.4.** *Consider two intersecting disks D_1 and D_2 with radii $r_1 \geq r_2$, having lens angle at least $2\pi/3$. Then, there is a set P of four points such that any disk F of radius at least r_1 that intersects both D_1 and D_2 contains a point of P.*

Proof. Let ℓ be the line through c_1 and c_2. Let E be the disk of radius r_1 and center $c \in \ell$ that satisfies the conditions (i) and (ii) of Lemma 2.2. Let P be the point set $\{c_1, c, p, q\}$ specified in the proof of Lemma 2.3, with respect to D_1 and E (see Figure 1, Right). We claim that

$$D_1 \cap F \neq \emptyset \ \wedge \ D_2 \cap F \neq \emptyset \ \Rightarrow \ E \cap F \neq \emptyset. \qquad (*)$$

Once (*) is established, we are done by Lemma 2.3. If $D_2 \subseteq E$, then (*) is immediate, so assume that $D_2 \not\subseteq E$. By Lemma 2.2, c lies between c_1 and c_2. Let k be the line through c perpendicular to ℓ, and let k^+ be the open halfplane bounded by k with $c_1 \in k^+$ and k^- the open halfplane

bounded by k with $c_1 \notin k^-$. Since $|c_1c| = \sqrt{3}r_1 > r_1$, we have $D_1 \subset k^+$ (see Figure 2, Right). Recall that F has radius at least r_1 and intersects D_1 and D_2. We distinguish two cases: (i) there is no intersection of F and D_2 in k^+, and (ii) there is an intersection of F and D_2 in k^+.

For case (i), let x be any point in $D_1 \cap F$. Since we know that $D_1 \subset k^+$, we have $x \in k^+$. Moreover, let y be any point in $D_2 \cap F$. By assumption (i), y is not in k^+, but it must be in the infinite strip defined by the two tangents of D_1 and E. Thus, the line segment $\overline{xy}$ intersects the diameter segment $k \cap E$. Since F is convex, the intersection of $\overline{xy}$ and $k \cap E$ is in F, so $E \cap F \neq \emptyset$.

For case (ii), let x be any point in $D_2 \cap F \cap k^+$. Consider the triangle Δxcc_2. Since $x \in k^+$, the angle at c is at least $\pi/2$ (Figure 2, Right). Thus, $|xc| \leq |xc_2|$. Moreover, since $x \in D_2$, we know that $|xc_2| \leq r_2 \leq r_1$. Hence, we have $|xc| \leq r_1$ so $x \in E$ and (*) follows, as $x \in E \cap F$. ◀

3 Stabbing disks in linear time

Let $\mathcal{D}$ be a set of n pairwise intersecting disks. For $r > 0$, we define $\bigcap_{\leq r} \mathcal{D}$ to be the intersection of all disks in $\mathcal{D}$ with radius at most r. The set $\bigcap_{< r} \mathcal{D}$ is defined analogously. Moreover, let X be a non-empty intersection of finitely many disks. Then, $\mathcal{V}(X)$ is the set of vertices on the boundary of X.

▶ **Lemma 3.1.** *For a set $\mathcal{D}$ of n pairwise intersecting disks, we can decide in linear time if the intersection $\bigcap \mathcal{D}$ is empty. In the same time, we can compute a point in $\bigcap \mathcal{D}$, if it exists, or a non-Helly triple D_i, D_j, D_k with $r_i, r_j \leq r_k$, such that $\bigcap_{<r_k} \mathcal{D} \neq \emptyset$, otherwise.*

Proof. Consider a subset $\mathcal{D}'$ of $\mathcal{D}$ and assume first that $\bigcap \mathcal{D}' = \emptyset$. In this case, there exists a disk $D_k \in \mathcal{D}'$ with radius r_k such that $\bigcap_{<r_k} \mathcal{D}' \neq \emptyset$ and $\bigcap_{\leq r_k} \mathcal{D}' = \emptyset$. We set $\mathrm{ind}(\mathcal{D}') = k$ and $\mathrm{rad}(\mathcal{D}') = r_k$. Next, assume that $\bigcap \mathcal{D}' \neq \emptyset$. In this case, we set $\mathrm{ind}(\mathcal{D}') = \infty$ and $\mathrm{rad}(\mathcal{D}') = \infty$. Now, for $\mathcal{D}' \subseteq \mathcal{D}$, we define $w(\mathcal{D}') = \left(\mathrm{rad}(\mathcal{D}'), -\min\left\{d(v, D_{\mathrm{ind}(\mathcal{D}')}) \mid v \in \mathcal{V}(\bigcap_{<\mathrm{rad}(\mathcal{D}')} \mathcal{D}')\right\}\right)$. If $\mathrm{ind}(\mathcal{D}') = \infty$ we set $d(v, D_\infty) = v_y$, the y-coordinate of v. Chan has observed that the problem $(\mathcal{D}, w)$ is *LP-type* [1,9]. The combinatorial dimension of $(\mathcal{D}, w)$ is 3, and therefore, the violation test can be done in constant time. Furthermore, for a basis $\mathcal{B}$ of $(\mathcal{D}, w)$, let $\mathrm{vio}(\mathcal{B})$ be the set of disks in $\mathcal{D}$ that violate $\mathcal{B}$, i.e., for all $D \in \mathrm{vio}(\mathcal{B})$, we have $w(\mathcal{B} \cup \{D\}) < w(\mathcal{B})$. Then, $(\mathcal{D}, \mathcal{R} = \{\mathrm{vio}(\mathcal{B}) \mid \mathcal{B} \text{ is a basis in } \mathcal{D}\})$ is the underlying range space for the LP-type problem, and one can check that it has constant VC-dimension. Thus, we can use the deterministic algorithm by Chazelle and Matoušek [2] to compute $w(\mathcal{D})$ and a corresponding basis $\mathcal{B}$ in $O(n)$ time. One can show that $\mathcal{B}$ is either a non-Helly triple for $\mathcal{D}$ with the desired properties, or that $\mathcal{B}$ yields a stabbing point for $\mathcal{D}$. ◀

▶ **Theorem 3.2.** *Given a set $\mathcal{D}$ of n pairwise intersecting disks in the plane, we can find in linear time a set S of five points such that every disk of $\mathcal{D}$ contains at least one point of S.*

Proof. Using Lemma 3.1, we decide if $\cap\mathcal{D}$ is empty. If not, we return a point in the common intersection. Otherwise, the lemma gives us a non-Helly tripe with smallest maximum radius r_k. For the disks $D_\ell \in \mathcal{D}$ with $r_\ell < r_k$, we can obtain in linear time a stabbing point s by using Helly's theorem and Lemma 3.1. Next, by Lemma 2.1, there are two disks D' and D'' among D_i, D_j and D_k whose lens angle is at least $2\pi/3$. Let P be the set of four points, as described in the proof of Lemma 2.4, that stabs any disk of radius at least r_k that intersects both D' and D''. Then $S = \{s\} \cup P$ is a set of five points that stabs all disks of $\mathcal{D}$. ◀

4 13 pairwise intersecting disks that cannot be stabbed by 3 points

The construction begins with an inner disk A, say of radius 1, and three larger disks D_1, D_2, D_3 of equal size, so that A is tangent to all three disks, and each pair of the disks are tangent to each other. Denote the contact point of A and D_i by ξ_i, for $i = 1, 2, 3$.

We add six very large disks as follows. For $i = 1, 2, 3$, we draw the two common outer tangents to A and D_i, and denote by T_i^- and T_i^+ the halfplanes that are bounded by these tangents and are openly disjoint from A. For concreteness, the labels T_i^- and T_i^+ are such that the points of tangency between A and T_i^-, D_i, and T_i^+, appear along ∂A in this counterclockwise order. One can show that the nine points of tangency between A and the other disks and halfplanes are all distinct (see Figure 3). We regard the six halfplanes T_i^-, T_i^+, for $i = 1, 2, 3$, as disks; in the end, we can apply a suitable inversion to turn the disks and halfplanes into actual disks, if so desired.

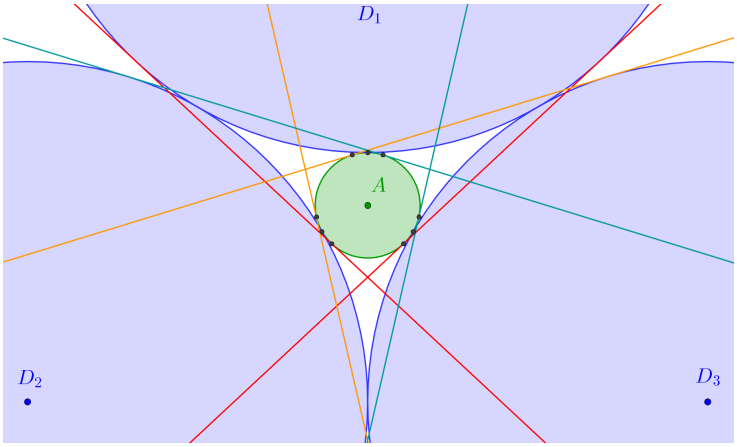


Figure 3 Each common tangent ℓ represents a very large disk tangent to the disks to which ℓ is tangent. The nine points of tangency are all distinct.

Finally, we construct three additional disks A_1, A_2, A_3. To construct A_i, we slightly expand A into a disk A_i' of radius $1 + \varepsilon_1$, while keeping it touching D_i at ξ_i. We then roll A_i' clockwise along D_i, by a tiny angle $\varepsilon_2 \ll \varepsilon_1$ to obtain A_i.

This completes the construction, giving 13 disks. For sufficiently small ε_1 and ε_2, we can ensure the following properties for each A_i: (i) A_i intersects all other 12 disks, (ii) the nine intersection regions $A_i \cap D_j$, $A_i \cap T_j^-$, $A_i \cap T_j^+$, for $j = 1, 2, 3$, are pairwise disjoint. (iii) $\xi_i \notin A_i$.

▶ **Lemma 4.1.** *The 13 disks in the construction cannot be stabbed by three points.*

Proof. Consider any set of three points and suppose they form a stabbing set. Let A^* be the union $A \cup A_1 \cup A_2 \cup A_3$. If p is a stabbing point in A^*, then typically p will stab all these four disks (unless p lies at certain peculiar locations), but, by construction, it stabs at most one of the nine remaining disks. It is thus impossible for all three stabbing points to lie in A^*, but at least one of them must lie there.

Assume first that A^* contains two stabbing points. As just argued, there are at most two of the remaining disks that are stabbed by these points. The following cases can then arise.

(a) The stabbed disks are both halfplanes. Then D_1, D_2, D_3 form a non-Helly triple, i.e. they do not have a common intersection, and none of them is stabbed. Since a non-Helly triple must be stabbed by at least two points, an unstabbed disk remains.

(b) The stabbed disks are both among D_1, D_2, D_3. Then the six unstabbed halfplanes form many non-Helly triples [1], e.g., T_1^-, T_2^-, and T_3^-, and again a disk remains unstabbed.

(c) One stabbed disk is D_1, D_2, or D_3, and the other is a halfplane. Then, there is (at least) one disk D_i such that it, and the two associated halfplanes T_i^-, T_i^+ are all unstabbed. (D_i is

[1] Note that it is easy to extend the definition of non-Helly triples to halfplanes.

a disk that is not stabbed by either of the two initial points, and neither of its two tangent halfplanes is stabbed.) Then D_i, T_i^-, and T_i^+ form an unstabbed non-Helly triple.

Assume then that A^* contains only one stabbing point p, so at most one of the nine remaining disks is stabbed by p. Since p is the only point that stabs all three disks A_1, A_2, A_3, it cannot be any of ξ_1, ξ_2, ξ_3, so the other disk that it stabs (if there is such a disk) must be a halfplane. That is, p does not stab any of D_1, D_2, D_3. Since D_1, D_2, D_3 form a non-Helly triple, they require two points to stab them all. Moreover, since we only have two points at our disposal, one of them must be the point of tangency of two of these disks, say of D_2 and D_3. This point stabs only two of the six halfplanes (concretely, they are T_1^- and T_1^+). But then D_1, T_2^+, and T_3^- form an unstabbed non-Helly triple. ◀

5 Conclusion

We presented a simple algorithm for the computation of five stabbing points for a set of pairwise intersecting disks by solving a corresponding LP-type problem. Nevertheless, the question remains open how to use the proofs of Danzer or Stachó (or any other technique) for an efficient construction of four stabbing points. Since eight disks can always be stabbed by three points [10], it remains open whether nine disks can be stabbed by three points or not. Furthermore, it would be interesting to find a simpler construction of ten pairwise intersecting disks that cannot be stabbed by three points.

Acknowledgments. We would like to thank Timothy Chan for pointing us to the direct LP-type algorithm described in Lemma 3.1.

References

1 T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proceedings of the fifteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–436, 2004.

2 B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996.

3 L. Danzer. Zur Lösung des Gallaischen Problems über Kreisscheiben in der Euklidischen Ebene. *Studia Sci. Math. Hungar.*, 21(1-2):111–134, 1986.

4 B. Grünbaum. On intersections of similar sets. *Portugal. Math.*, 18:155–164, 1959.

5 H. Hadwiger and H. Debrunner. Ausgewählte Einzelprobleme der kombinatorischen Geometrie in der Ebene. *Enseignement Math. (2)*, 1:56–89, 1955.

6 E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

7 E. Helly. Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten. *Monatshefte für Mathematik*, 37(1):281–302, 1930.

8 J. Radon. Mengen konvexer Körper, die einen gemeinsamen Punkt enthalten. *Mathematische Annalen*, 83(1):113–115, 1921.

9 M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. *STACS 92*, pages 567–579, 1992.

10 L. Stachó. Über ein Problem für Kreisscheibenfamilien. *Acta Sci. Math. (Szeged)*, 26:273–282, 1965.

11 L. Stachó. A solution of Gallai's problem on pinning down circles. *Mat. Lapok*, 32(1-3):19–47, 1981/84.

Combinatorial and Asymptotical Results on the Neighborhood Grid Data Structure

Martin Skrodzki[1], Ulrich Reitebuch[1], Konrad Polthier[1], and Shagnik Das[1]

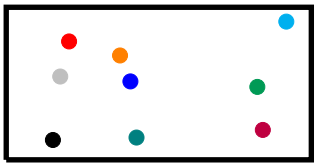
1 **Freie Universität Berlin**
martin.skrodzki@fu-berlin.de
ulrich.reitebuch@fu-berlin.de
konrad.polthier@fu-berlin.de
shagnik@mi.fu-berlin.de

Abstract

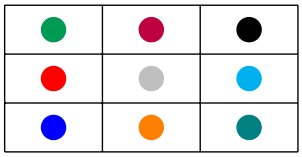
In 2009, Joselli et al. introduced the Neighborhood Grid data structure for fast computation of neighborhood estimates in point sets. Even though the data structure has been used in several applications and shown to be practically relevant, it is theoretically not yet well understood. The purpose of this paper is to give results on the complexity of building algorithms – both single-core and parallel – for the neighborhood grid. Furthermore, current investigations on related combinatorial questions are presented.

1 Introduction

The neighborhood grid data structure can be used to compute estimates of neighborhoods in point sets. That is, for a given point p_i in a point set P, it provides a point p_j that is close to p_i but not necessarily its nearest neighbor in P. It has been introduced by Joselli et al. [3, 4]. In order to give a short introduction to the data structure, consider the example in Figure 1. It shows how points from a point set (Figure 1a) are placed in a grid (Figure 1b). The order in which the points are given is random, thus their initial placement in the grid is also. After the placement, only the coordinates of the points are considered in the grid (Figure 1c).



(a) Point set with nine points in $\mathbb{R}^2$.



(b) The points are initially placed randomly.

8.23, 4.79	8.41, 1.96	1.53, 1.30
2.06, 7.76	1.77, 5.46	9.18, 9.05
4.07, 5.13	3.73, 6.84	4.27, 1.45

(c) The point coordinates are considered.

Figure 1 First part of the neighborhood grid pipeline.

The grid as obtained in Figure 1c will now be sorted. Each row should grow in the first coordinates from left to right, each column should grow in the second coordinates from bottom to top. A corresponding sorted grid is given in Figure 2a. Note how it – in this example – recovers the combinatorial neighborhood relation from the points.

In order to use the grid to determine a neighborhood estimate for a given point p_i, find that point in the sorted grid. Then, consider a small neighborhood around that point in the grid, e.g. the one-ring around it. The size of this neighborhood should not depend on the number of inserted points such that this lookup runs in asymptotically constant time $\mathcal{O}(1)$. From that neighborhood in the grid, find the closest point to the considered point and output it as estimated nearest neighbor to p_i.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2.06, 7.76	3.73, 6.84	9.18, 9.05
1.77, 5.46	4.07, 5.13	8.23, 4.79
1.53, 1.30	4.27, 1.45	8.41, 1.96

(a) Coordinates are sorted to grow in their x-values in rows and in their y-values in columns.

2.06, 7.76	3.73, 6.84	9.18, 9.05
1.77, 5.46	4.07, 5.13	8.23, 4.79
1.53, 1.30	4.27, 1.45	8.41, 1.96

(b) Determining the neighbors of the upper left point by looking at its neighbors in the grid.

Figure 2 Second part of the neighborhood grid pipeline.

In this report, we present new results on the neighborhood grid data structure. A more extensive version of our results can be found in a corresponding paper on ArXiv [6]. Malheiros and Walter [2] investigated several iterative building strategies for the data structure. Despite the evidences of practical relevance, as given in the publication cited above, neither Joselli nor Malheiros investigated the asymptotic building times of the grid or answered the question for a time-optimal building algorithm. Therefore, this paper contains:

- a polynomial-time algorithm to build a neighborhood grid (Theorem 2.2),
- a proof of asymptotic time-optimality of the presented algorithm (Section 3.1),
- a comparison with the parallel building algorithm of Malheiros and Walter (Section 3.2).

The mentioned ArXiv paper contains – apart from more examples and proofs – several combinatorial results on the number of possible sorted placements, a complete list of unique sorted placements for $n \in \{1, 2, 3\}$, and a proof of non-existence of unique sorted placements for $n \geq 4$. So far, the following question remains unanswered:

- For a given $n \in \mathbb{N}$, $n \geq 4$, what is a point set with the least or largest number of stable states?

For the case of the largest number we give a conjecture.

2 The Neighborhood Grid

In this first section, we present the neighborhood data structure, fix corresponding notation, and prove a first theorem on a polynomial-time building algorithm.

2.1 Definition of the Data Structure

Given a set of points $P = \{p_1, \ldots, p_N \mid p_i \in \mathbb{R}^d\}$. In the following we will assume that $N = n^2$ for some $n \in \mathbb{N}$ and $d = 2$. Therefore, each point is given by $p_i = (p_{i1}, p_{i2}) \in \mathbb{R}^2$, where p_{i1} will be referred to as x- and p_{i2} as y-value. Furthermore, we assume that $p_i \neq p_j$ for all $i \neq j$. Finally, we can restrict w.l.o.g. to $\{p_{ik} \mid i \in [N]\} = [N]$ for $k \in \{1, 2\}$, which will be important in Section 3.1. Consider [6] for the general case without these restrictions.

The points will be placed in an $n \times n$ matrix, where each cell of the matrix contains a point p_i. That is, we consider a matrix $M \in (\mathbb{R}^2)^{n \times n}$ which then has the form

$$M = \begin{array}{|c|c|c|} \hline (a_{1n}, b_{1n}) & \ldots & (a_{nn}, b_{nn}) \\ \hline \vdots & \ddots & \vdots \\ \hline (a_{11}, b_{11}) & \ldots & (a_{n1}, b_{n1}) \\ \hline \end{array}. \tag{1}$$

Ultimately, we want to order the points in the matrix such that the following state is reached.

▶ **Definition 2.1.** The matrix M as given in (1) is said to be in a **stable state** if and only if the following two conditions are satisfied for any $i, j \in [n]$, $i \neq j$.

1. For all $k \in [n]$ it is: $i < j \Rightarrow a_{ki} < a_{kj} \vee (a_{ki} = a_{kj} \wedge b_{ki} < b_{kj})$.
2. For all $\ell \in [n]$ it is: $i < j \Rightarrow b_{i\ell} < b_{j\ell} \vee (b_{i\ell} = b_{j\ell} \wedge a_{i\ell} < a_{j\ell})$.

In other words, a matrix M is in a stable state, if the points in each row of M are ordered lexicographically according to the first and then the second coordinate. Similarly, all columns of M have to be ordered lexicographically according to the second and then the first coordinate. An illustration of Definition 2.1 is given in Figure 3. We call a stable state **unique**, if there exists no other stable state for the same point set P.

$(95, 13)$	$(95, 65)$	
	$(26, 61)$	$(13, 69)$
$(55, 42)$	$(60, 49)$	

$(06, 69)$	$(26, 61)$	$(86, 89)$
$(02, 55)$	$(80, 34)$	$(86, 41)$
$(05, 19)$	$(47, 11)$	$(95, 13)$

Figure 3 On the left a partially filled matrix with a violation of Definition 2.1 marked red. On the right a 3×3 matrix M in stable state.

2.2 Polynomial-Time building algorithm

Now the following question arises naturally: For any set of points P as specified above, is there a bijective placement $\pi : [n^2] \to [n] \times [n]$, $i \mapsto (k, \ell)$ such that the matrix $M_\pi(P)$ with

$$M = \begin{array}{|c|c|c|} \hline (p_{\pi^{-1}(n,1)1}, p_{\pi^{-1}(n,1)2}) & \cdots & (p_{\pi^{-1}(n,n)1}, p_{\pi^{-1}(n,n)2}) \\ \hline \vdots & \ddots & \vdots \\ \hline (p_{\pi^{-1}(1,1)1}, p_{\pi^{-1}(1,1)2}) & \cdots & (p_{\pi^{-1}(1,n)1}, p_{\pi^{-1}(1,n)2}) \\ \hline \end{array}. \tag{2}$$

is in a stable state? In other words, given n^2 points, can these be written into an $n \times n$ matrix such that it is in a stable state as defined in Definition 2.1.

▶ **Theorem 2.2.** *For every set of points $P = \{p_1, \ldots, p_N \mid p_i \in \mathbb{R}^2\}$ there is a bijective placement π such that $M_\pi(P)$ is in a stable state. A placement π can be found in $\mathcal{O}(N \log(N))$.*

Proof. Consider the points $p_1, \ldots, p_N$ as a sequence. Sort this sequence according to the first condition given in Definition 2.1. Obtain a sequence

$$(q_{11}, q_{12}), (q_{21}, q_{22}), \ldots, (q_{N1}, q_{N2}),$$

where for $i, j \in [n]$, $i < j$ we have $q_{i1} < q_{j1}$ or $(q_{i1} = q_{j1} \wedge q_{i2} < q_{j2})$. Now split this sequence into n blocks as follows:

$$\underbrace{(q_{11}, q_{12}), \ldots, (q_{n1}, q_{n2})}_{=:Q_1}, \underbrace{(q_{(n+1)1}, q_{(n+1)2}), \ldots, (q_{(2n)1}, q_{(2n)2})}_{=:Q_2},$$
$$\ldots \underbrace{(q_{(n^2-n+1)1}, q_{(n^2-n+1)2}), \ldots, (q_{N1}, q_{N2})}_{=:Q_n},.$$

Now consider each sequence Q_i and sort it according to the second condition given in Definition 2.1. Obtain a sequence

$$R_k := (r_{11}, r_{12}), (r_{21}, r_{22}), \ldots, (r_{n1}, r_{n2}),\ k \in [n],$$

where for $i < j$ we have $r_{i2} < r_{j2}$ or $(r_{i2} = r_{j2} \wedge r_{i1} < r_{j1})$. That is, the points in the sequence R_k are sorted according to the second condition of Definition 2.1. Furthermore, for $i < j$, any point from R_i satisfies the first condition of Definition 2.1 when compared to any point from R_j, since the R_k derive from the Q_k. Therefore, placing the sequence R_k into the kth column of the matrix M results in a stable state.

Concerning the runtime, in the first step, N points were sorted, which takes $\mathcal{O}(N \log(N))$. In the second step, n sets of n points each were sorted, which takes

$$n \cdot \mathcal{O}(n \log(n)) = \mathcal{O}(n^2 \log(\sqrt{N})) = \mathcal{O}(N \log(N)),$$

as $N = n^2$. Hence, the stable state was computed in $\mathcal{O}(N \log(N))$. ◀

Theorem 2.2 imposes an upper bound on the runtime of any time-optimal comparison-based algorithm that creates a stable state of a matrix M. The next question is then: What is a lower bound?

3 Optimality of the Algorithm

3.1 A Lower Bound

To prove a lower bound, consider a comparison-based algorithm $\mathcal{A}$. The input to $\mathcal{A}$ is a point set P, the output is a stable placement π. Each query of $\mathcal{A}$ establishes $p_{ik} < p_{jk}$ for $i, j \in [N]$, $k \in \{1, 2\}$ and can be seen as a node of a decision-tree. The leafs of this tree correspond to placements of which some are stable for the given point set. A time-optimal algorithm builds this tree such that it is of depth $\log((n^2)!)$.

If we fix a placement π, we can say w.l.o.g. that π fixes the x-coordinates in the matrix such that they satisfy Definition 2.1. When counting the number of point sets for which π is stable, we can now pair the already placed x-values with y-values as follows: When setting up the y-values for the first column, one can pick n of the possible $N = n^2$ values, which then admit to a unique order in the column. Therefore, for the y-values in the first column, there are $\binom{n^2}{n}$ possibilities. For the second column, there are $\binom{n^2-n}{n}$ possibilities, until there is $\binom{n^2-(n-1)n}{n} = 1$ possibility for the last column. Overall, there are

$$\prod_{k=0}^{(n-1)} \binom{n^2 - kn}{n} = \frac{n^2!}{(n^2-n)!n!} \cdot \frac{(n^2-n)!}{(n^2-2n)!n!} \cdot \ldots \cdot \frac{(2n)!}{n!n!} \cdot \frac{n!}{n!} = \frac{(n^2)!}{(n!)^n}$$

possibilities to put y-values into the matrix and obtain a stable state from them utilizing the fixed π. That is, a placement π is always stable for exactly $\frac{(n^2)!}{(n!)^n}$ point sets.

Thus, when building its decision-tree, the algorithm $\mathcal{A}$ cannot stop at a subtree with more than $\frac{(n^2)!}{(n!)^n}$ leafs, as one of them will surely not be stable under the currently considered placement. That is, the tree has to be traversed to depth at least

$$\log((n^2)!) - \log\left(\frac{(n^2)!}{(n!)^n}\right) = \log\left(\frac{(n^2)! \cdot (n!)^n}{(n^2)!}\right) = \log((n!)^n) = n \cdot \log(n!) = \mathcal{O}(n^2 \cdot \log(n)).$$

Therefore, each decision-based algorithm building a stable state needs to perform at least $\Omega(n^2 \cdot \log(n))$ operations. Together with Theorem 2.2 this proves the following:

▶ **Theorem 3.1.** *The algorithm outlined in Theorem 2.2 is asymptotically time-optimal.*

3.2 Comparison to Malheiros and Walter

In the previous section, we have seen that a decision-based algorithm running on a single core has optimal asymptotic runtime $\mathcal{O}(n^2 \cdot \log(n))$. However, both Joselli et al. [3, 4] and Malheiros and Walter [2] utilize a parallelized version of odd-even sort. Assuming $n^2/2$ processors given, they alternately perform one step of the odd-even sort algorithm on rows and columns. By exchanging two points that violate Definition 2.1, they claim to converge to a stable state. Even though they do not prove this claim, it can easily be established when plugging the matrix M from Equation (1) into the energy

$$E(M) = \sum_{i,j=1}^{n} i \cdot a_{ij} + j \cdot b_{ij}, \tag{3}$$

which grows for each exchange, but is bounded from above. Thus, the procedure converges to a stable state.

As a point can only move by one row or column in each step, consider the element $(1, 1)$ that has to be placed in the lower left corner given our restrictions. In case it starts in the upper right corner, the algorithm needs to perform $2n - 2 = \mathcal{O}(n)$ steps to move the element to its designated position. Therefore, this parallel algorithm has a lower bound of $\omega(n)$. There are even examples for elements that cycle through the grid, consider [6] for an example.

Note that the algorithm presented in Theorem 2.2 needs to sort the given points. When utilizing $n^2/2$ processors, sorting can be performed in $\log(n^2)$ time, see [1]. Therefore, the presented algorithm can be parallelized to run in $\mathcal{O}(\log(n^2))$. However, this is of rather theoretical relevance, as the constants in [1] are comparably large.

Compare this to building a KdTree in parallel. In each step i, we have to sort i sets of $n^2/2^i$ points in the dimension with largest spread, which takes $\log(n^2) - i$ time for each of the $\log(n^2)$ levels of the tree, resulting in a total building time of $\mathcal{O}(\log^2(n))$. Therefore, the neighborhood grid can be build slightly faster, but only gives estimated answers, while the KdTree provides exact neighbor relations.

4 Combinatorial Questions

4.1 Point Set with a unique Stable State

In the previous section it was shown that the running time of the algorithm outlined in Theorem 2.2 is asymptotically time-optimal. However, the question remains whether the stable state found by the algorithm for a given point set P is unique. By iterating over all possible point sets P with $n = 4$, we found that none of the two-dimensional point sets on 16 points has a unique stable state. Utilizing an inductive argument, we show that given any point set P with $n \geq 5$, there exists no unique stable state. See [6] for details and a complete enumeration of unique stable states in the case of $n \in \{1, 2, 3\}$. The fact that for $n \geq 4$ there is no point set with a single unique stable state raises the following question:

Open Question. Given $n \in \mathbb{N}$, $n \geq 4$, what is a point set P with the minimum number of stable states among all point sets with n^2 points?

4.2 Point Set with largest number of Stable States

We proceed by turning the question from the last section around. What is the maximal number of stable states a point set can obtain for some given $n \in \mathbb{N}$? In order to investigate

this question, we first turn to a specific point set, for which we can count the number of stable states. Consider the *identity*: $\{(1,1),(2,2),\ldots,(n^2,n^2)\}$. Counting the number of stable states for the identity is equivalent to placing only one number in each field of the $n \times n$ matrix, which then has to satisfy both conditions of Definition 2.1. But this is exactly the number of standard Young tableaux of shape $(n,\ldots,n)$. See [5] for an introduction into the underlying combinatorics and [6] for the application of these to the given setup. The number of stable states of the identity is then given by

$$f^{(n,\ldots,n)} = \frac{N!}{\prod_{i=1}^{n}\prod_{j=1}^{n}(2n-i-j+1)}. \tag{4}$$

The results for $n \in \{1,2,3\}$ and computational experiments lead us to state the following conjecture.

▶ **Conjecture 4.1.** *Given $n \in \mathbb{N}$, the number of stable states of any point set P on n^2 points is less or equal to $f^{(n,\ldots,n)}$.*

5 Conclusion and Future Work

We have presented a polynomial-time algorithm to build a stable state for a given point set P. Furthermore, we have proven the parallel algorithm from [2–4] to converge to a stable state and provided a lower bound on its runtime. Finally, we have deduced two open combinatorial questions resulting from the investigations of the data structure. A question not addressed in this paper concerns the quality of neighborhood estimates obtained from the grid. Answering these is left as future work.

Acknowledgments. The first author acknowledges the support by the Einstein Center for Mathematics Berlin, the Berlin Mathematical School, and the German National Academic Foundation. Furthermore, this research was supported by the DFG Collaborative Research Center TRR 109, 'Discretization in Geometry and Dynamics'.

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in c logn parallel steps. *Combinatorica*, 3(1):1–19, 1983.

2 Marcelo de Gomensoro Malheiros and Marcelo Walter. Simple and efficient approximate nearest neighbor search using spatial sorting. In *Graphics, Patterns and Images (SIBGRAPI), 2015 28th SIBGRAPI Conference on*, pages 180–187. IEEE, 2015.

3 Mark Joselli, José Ricardo da S Junior, Esteban W Clua, Anselmo Montenegro, Marcos Lage, and Paulo Pagliosa. Neighborhood grid: A novel data structure for fluids animation with gpu computing. *Journal of Parallel and Distributed Computing*, 75:20–28, 2015.

4 Mark Joselli, Erick Baptista Passos, Marcelo Zamith, Esteban Clua, Anselmo Montenegro, and Bruno Feijó. A neighborhood grid data structure for massive 3d crowd simulation on gpu. In *2009 VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, pages 121–131. IEEE, 2009.

5 Bruce E. Saga. *The Symmetric Group*. Springer, 2001.

6 M. Skrodzki, U. Reitebuch, and K. Polthier. Combinatorial and Asymptotical Results on the Neighborhood Grid. *ArXiv e-prints*, October 2017. `arXiv:1710.03435`.

A Note on Planar Monohedral Tilings*

Oswin Aichholzer[1], Michael Kerber[1], István Talata[2], and Birgit Vogtenhuber[1]

1 **Graz University of Technology, Graz, Austria**
oaich@ist.tugraz.at, kerber@tugraz.at, bvogt@ist.tugraz.at

2 **Ybl Faculty of Architecture and Civil Engineering, Szent István University, Budapest, Hungary; University of Dunaújváros, Dunaújváros, Hungary**
Talata.Istvan@ybl.szie.hu

Abstract

A planar *monohedral tiling* is a decomposition of $\mathbb{R}^2$ into congruent *tiles*. We say that such a tiling has the *flag property* if for each triple of tiles that intersect pairwise, the three tiles intersect in a common point. We show that for convex tiles, there exist only three classes of tilings that are not flag, and they all consist of triangular tiles; in particular, each convex tiling using polygons with $n \geq 4$ vertices is flag. We also show that an analogous statement for the case of non-convex tiles is not true by presenting a family of counterexamples.

1 Introduction

Problem statement and results. A *plane tiling* in the plane is a countable family of planar sets $\{T_1, T_2, \ldots\}$, called *tiles*, such that each T_i is compact and connected, the union of all T_i is the entire plane and the T_i are pairwise interior-disjoint. We call such a tiling *monohedral* if each T_i is congruent to T_1. In other words, a monohedral tiling can be obtained from the shape T_1 by repeatedly placing (translated, rotated, or reflected) copies of T_1. Two of the simplest examples for such monohedral tilings are shown in Figure 1. These are also instances of *convex tilings*, where we require that each tile is convex. A comprehensive study of tilings with numerous examples can be found in the monograph by Grünbaum and Shephard [3].

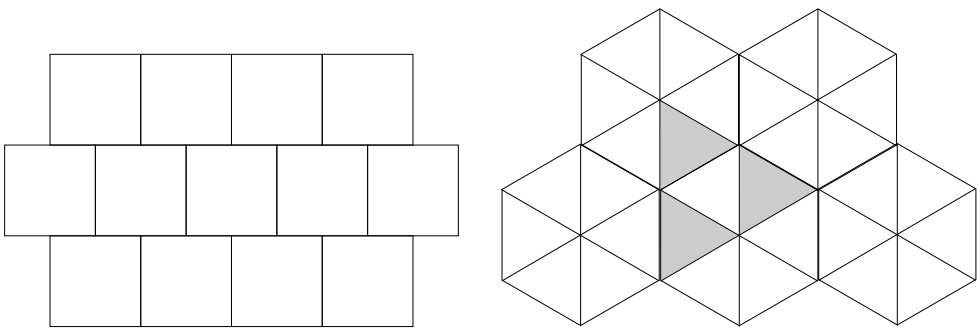


Figure 1 Monohedral tiling with squares (left) and equilateral triangles (right). On the right, an obstructing triple for the flag property is shaded.

We are interested in a special property of (monohedral) tilings: We say that a tiling is *flag* if whenever three tiles intersect pairwise, they also intersect in a point common to all three tiles. It can easily be verified that the left tiling in Figure 1 is flag, whereas the right

* Research for this work is supported by the Austrian Science Fund (FWF) grant W1230. MK is supported by the Austrian Science Fund (FWF) grant number P 29984-N35.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

tiling is not: the three edge neighbors of any triangle intersect pairwise (in single points), but have no common intersection. We call such a triple an *obstructing triple*. We are interested in the following question: which monohedral tilings have the flag property?

Our main result is that "most" convex monohedral tilings in the plane are flag. There are only three types of counterexamples, namely the ones depicted in Figure 1 (right) and in Figure 2. In particular, all counterexamples require triangles as tiles. As a consequence, every convex monohedral tiling with convex polygons having 4 or more vertices is flag.

To explain the three types of non-flag tilings, we observe that the union of the three tiles of an obstructing triple divides the complement into a bounded and an unbounded connected component. We call the closure of the bounded component the *cage* of the triple. Of course, the cage has to be filled out by copies of the same tile. We define the *cage number* of a cage as the number of tiles inside the cage, and the cage number of a tiling as the maximal cage number of all cages in the tiling. The three counterexamples correspond to tilings with cage number 1, 2, and 3. We show that no convex tiling with cage number 4 or higher exists.

The situation changes significantly for non-convex monohedral tilings. In that case, non-flag tilings exist for polygons with an arbitrary number of vertices and the cage number can go well beyond 3. As a further contribution, we present a general construction that, for an arbitrary fixed integer c, generates a tiling with cage number c.

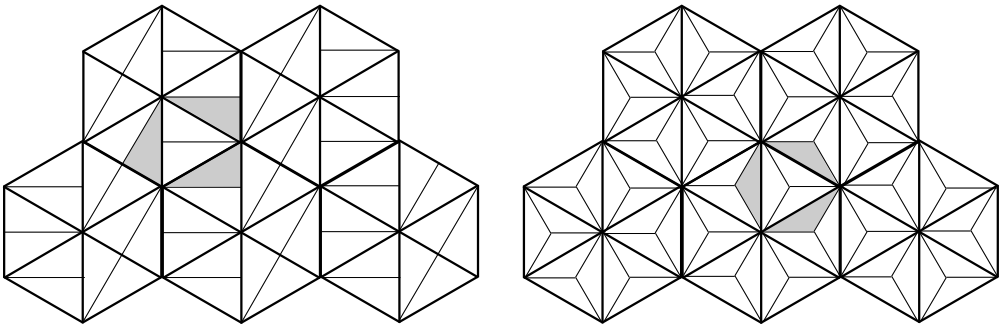


Figure 2 Non-flag Monohedral tilings with cage number 2 (left) and 3 (right). These tilings are obtained from the equilateral tiling from Figure 1 (right) by splitting each triangle in two congruent copies using an altitude, or by splitting each triangle in three congruent copies using the barycenter, respectively. An obstructing triple with the maximal cage number is shaded.

Motivation. The term "flag" originates from the following concepts: A simplicial complex C is called a *flag complex* (also *clique complex*) if it has the following property: if for vertices $\{v_0, \ldots, v_k\}$, all edges (v_i, v_j) are in C, then the k-simplex spanned by $\{v_0, \ldots, v_k\}$ is also in C. Equivalently, C is a flag complex if it is the inclusion-maximal simplicial complex that can be constructed out of the edges of C.

In our setup, a tiling gives rise to a dual simplicial complex, called the *nerve* of the tiling, obtained by defining one vertex per tile, and adding a k-simplex if the corresponding k+1 tiles have a non-empty common intersection. Note that this complex might be high-dimensional – for instance, the nerve of the triangular tiling in Figure 1 contains 5-simplices. The tiling being flag is a necessary condition for the nerve of the tiling being a flag complex. Indeed, if a triple of tiles violates the flag property, the dual complex consists of three edges forming the boundary of a 2-simplex, but the 2-simplex is missing as the three tiles do not commonly intersect. For convex tilings, the tiling is flag if and only if its nerve is a flag complex, which is a simple consequence of Helly's Theorem.

Our question is motivated from an application in computational topology. In [2], the d-dimensional Euclidean space is tiled with permutahedra, and the nerve of a subset of them is the major object of study. In that paper, it is proven (Lemma 10 of [2]) that this nerve is a flag complex (for all d), which simplifies the computation of the complex. The first part of the proof is to show that the tiling has the flag property; for that, two disjoint facets of a permutahedron are considered and it is proven that the neighboring permutahedra along these two facets do not intersect, which implies the flag property. This proof makes use of the special structure of permutahedra and explicitly defines a separating hyperplane for the two neighboring permutahedra, involving lengthy calculations. This note is a first step towards generalizing this useful property of permutahedra to a larger class of tilings, starting with a complete analysis of the planar case.

2 Convex non-flag tilings

We fix a convex monohedral non-flag tiling with an obstructing triple (T_1, T_2, T_3) throughout. Clearly, T_1 (and so, T_2 and T_3) must be a polygon, since any convex non-linear boundary component would require a neighboring tile with a concave boundary component. Since the triple (T_1, T_2, T_3) intersects pairwise, but not commonly, the union $T_1 \cup T_2 \cup T_3$ is a connected set with a hole. While this can also be shown with elementary geometric considerations, a short proof uses the Nerve theorem [1] [4, Ch 4.G], stating that the union of convex shapes is homotopically equivalent (see e.g. [4] for a definition) to their nerve, which in our case is a cycle with three edges. Hence, the union of the three tiles is homotopically equivalent to S^1, a circle.

We call the closure of the (unique) bounded connected component of the complement the *cage* X of the triple. We start with studying the structure of X, relating it with a structure from computational geometry: a *(polygonal) pseudotriangle* is a simple polygon in the plane that is bounded by three concave chains [5]. The degenerate case in which one or several concave chains are just line segments is allowed; hence triangles are a special case of pseudotriangles.

▶ **Lemma 2.1.** *The cage X is a pseudotriangle.*

Proof. The boundary of X consists of boundary curves of the three convex polygons T_1, T_2, and T_3. By convexity, these curves are convex with respect to T_i, and hence concave with respect to the complement. ◀

A pseudotriangle has three *corners* where two concave chains meet. In our case, these corners correspond to intersections of two tiles among $\{T_1, T_2, T_3\}$. The *diameter* of a compact point set is the maximal distance between any pair of points in the set. Two points realizing this distance are called a *diametral pair*. For pseudotriangles, it is easy to see that only corners can form diametral pairs.

▶ **Lemma 2.2.** *Let X be a cage, and let T_X be a tile in the cage. Then, T_X contains two corners of X that form a diametral pair. Moreover, the corresponding concave arc connecting these corners along the boundary of X is a line segment.*

Proof. We define the *latitude* of a compact set S in the plane as the length of the longest line segment that is contained in S. Clearly, congruent sets have the same latitude, and $S' \subseteq S$ implies that the latitude of S' is at most the latitude of S. Let $\ell = \ell(T_1)$ be the latitude of T_1. Then, X must have latitude at least ℓ because it contains at least one congruent copy of T_1.

On the other hand, the latitude of a set is upper bounded by the diameter and for convex sets, both values coincide. Note that for any pair of corners of X, the line segment connecting them is completely contained in some T_i, because the corners are intersection points of tiles. Because all T_i are congruent, the diameter of T_1 is at least the distance of any pair of corners. It follows that the diameter of T_1 is at least the diameter of X. Putting all together, we have

$$\operatorname{diam}(X) \geq \ell(X) \geq \ell(T_1) = \operatorname{diam}(T_1) \geq \operatorname{diam}(X)$$

which implies that all quantities coincide. Since T_X has the same latitude as T_1, it must contain a diametral pair of X, which consists of two corners. Moreover, since T_X is convex, it contains also the line segment between these two corners, implying that X is bounded by this line segment. ◀

Since each tile in a cage has to cover a line segment between two corners, it follows that:

▶ **Corollary 2.3.** *A cage contains at most* 3 *tiles.*

Finally, we can analyze the three possible numbers of tiles inside a cage to show that all of them can only appear for triangular tiles.

▶ **Theorem 2.4.** *If a convex monohedral tiling is not flag, then the tiles are triangles.*

Proof. Assume that tiles (T_1, T_2, T_3) exist that form a cage X. Let c be the number of tiles inside the cage. We know that $c \in \{1, 2, 3\}$ from Corollary 2.3.

If $c = 1$, then X is a tile itself, and hence convex. Because the cage is a pseudotriangle, it is convex if and only if it is a triangle.

If $c = 2$, Lemma 2.2 implies that X has two line segments as sides, and a third concave arc which might be a line segment or a polyline with two segments; a polyline with more vertices is impossible because X is the union of two convex sets. Let v be the corner of X opposite to that third concave arc. Since the two tiles inside the cage intersect in a line segment from v to a point on the opposite arc, the only possibility is that the tiles are triangles.

If $c = 3$, the three tiles inside the cage have to intersect in a common point x as otherwise, they would form a cage again, and X would contain at least 4 tiles. Moreover, by Lemma 2.2, X is a triangle, and each corner is an intersection point of two tiles inside the cage. It follows that the three line segments joining x with the corners of X are the boundaries of the three tiles. However, these line segments split X into three triangles. ◀

We remark that the converse of Theorem 2.4 is not true: there are triangular tilings which are flag (an example can be obtained from the square tiling in Figure 1 (left) by subdividing each square into two triangles arbitrarily). However, the converse becomes true with a further restriction: we call a tiling *face-to-face* if the intersection of two tiles is a facet of both tiles (that is, the tiling carries the structure of a cell complex). For a face-to-face tiling with triangles, it is easy to see that for any triangle T, the three neighboring tiles sharing an edge with T form a cage that contains exactly T. Hence, a planar monohedral face-to-face tiling is flag if and only if the tiles are not triangles.

3 Non-convex tilings

Non-convex monohedral tilings have a long history of research. A remarkable case of instances are *spiral tilings*, for instance the Voderberg tiling[1] or the spiral version of the "Bent Wedge

[1] See `https://en.wikipedia.org/wiki/Voderberg_tiling`

tiling"[2]. By inspecting these tilings, it is not difficult to detect obstructing triples, refuting the possibility that Theorem 2.4 remains true without the convexity assumption.

For an arbitrary integer $n \geq 3$, we describe a construction of a non-convex monohedral tiling with tiles having $2n+1$ vertices such that an obstructing triple with cage number $n-1$ exists. This shows that also Corollary 2.3 is a property that crucially relies on the convexity of the tiles. Our construction is a variant of so-called *radial tilings*[3]. Consider the regular $6n$-gon P inscribed in the unit circle and fix an arbitrary vertex B on that polygon (Figure 3 (left)). Let D be a point on the unit circle such that the triangle OBD is equilateral. In fact, D is a vertex of P. Let c be the circular arc between O and B of the (unit) circle centered at D. Divide c in n sub-arcs of identical length, using $n-1$ additional subdivision points. Let p_1 denote the polyline from O to B defined by these subdivision points.

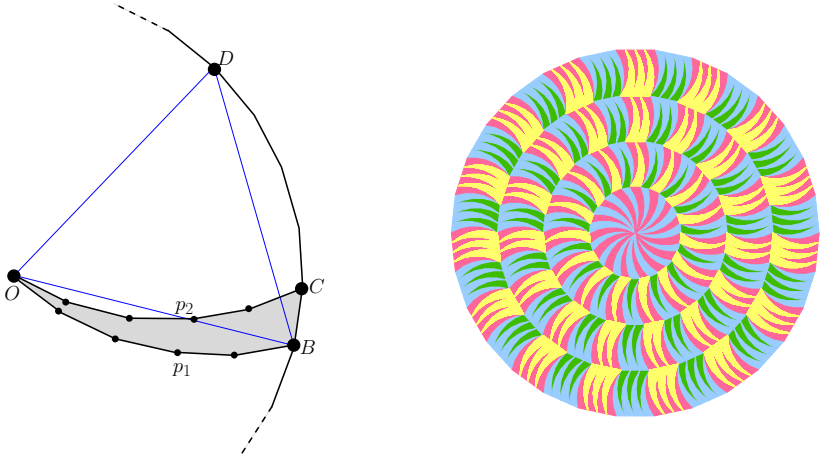


Figure 3 Left: Illustration of the construction of T for $n = 5$. Right: Radial tiling using T.

Next, apply a rotation around the origin (in either direction) by $\frac{2\pi}{6n}$, so that B is mapped to a neighboring vertex C of P. This rotation maps p_1 into a polyline p_2 from O to C. The polygon T bounded by p_1, p_2, and the line segment BC is a polygon with $2n+1$ vertices.

We argue that T indeed admits a monohedral tiling. First of all, by rotating T around the origin by multiples of $\frac{2\pi}{6n}$, $6n$ copies of T cover P. To cover the polygonal annulus between P and $2P$, we observe that the $6n$ reflections of the inner tiles can be completed with $12n$ congruent tiles to fill out the annulus. Extending this idea for the annulus between iP and $(i+1)P$, we can cover the entire plane with copies of T (see Figure 3 (right)).

Finally, to construct a large cage, we modify the tiling inside P: we split the $6n$ tiles into 6 pairwise disjoint groups, each consisting of n consecutive copies of T. Consider such a group G and denote with B and D its two extreme vertices on P. Note that the triangle OBD is equilateral and that the boundary of G consists of three identical polygonal chains (two of them convex and one reflex). It is therefore possible to reflect the whole group G, such that it again covers the same space, and that all tiles in the group intersect at D instead of O. We reflect 3 of the 6 groups inside P, alternating between reflected and unreflected groups. The tiles outside of P are left unchanged. See Figure 4 for two examples. We observe that the cage number of these tilings is $n-1$.

[2] See Steve Dutch's webpage `https://www.uwgb.edu/dutchs/symmetry/radspir1.htm`
[3] See also `https://www.uwgb.edu/dutchs/symmetry/rad-spir.htm`

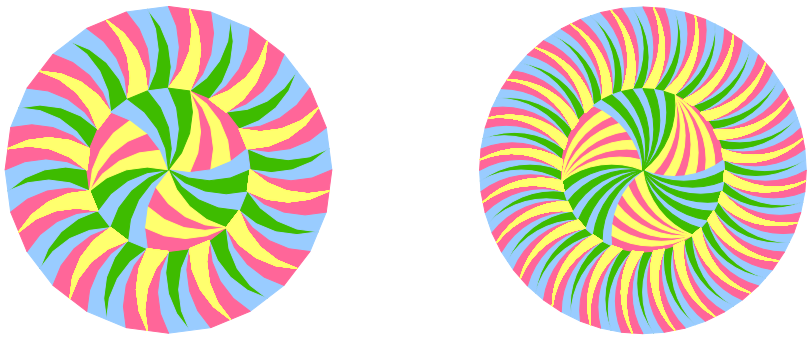


Figure 4 The final outcome of our construction after rearranging the innermost tiles for $n = 4$ (left) and $n = 8$ (right). In both cases, there are 6 groups of tiles around the origin, and three of them are rotated. The tile of a rotated group at the boundary of the $6n$-gon together with the extremal tiles of the neighboring (unrotated) groups form an obstructing triple with cage number 3 on the left, and 7 on the right.

4 Conclusion

Various questions remain open for the non-convex case. For instance: is there a *monohedral* tiling that is flag such that its nerve is not a flag complex? While it is rather simple to give an example of four non-convex shapes whose nerve is the boundary of a tetrahedron, it is not so simple to provide such an example with congruent shapes, and even less so to construct such a scenario in a monohedral tiling. Another question is what would be the maximal cage number possible for a monohedral tiling with a k-vertex polygon. Our paper establishes the lower bound of $\frac{k-3}{2}$. We are currently not able to provide any upper bound.

More in line with our original motivation, we plan to investigate convex monohedral tilings in higher dimension next. In detail, we want to characterize large classes of such tilings for which the nerve is a flag complex. Already in three dimensions, the natural generalization of Theorem 2.4 that all non-tetrahedral tilings have this property fails because we can simply extend Figure 1 (right) to the third dimension using triangular prisms. A statement in reach seems to be the following: restricting to face-to-face tilings, we call a tiling in $\mathbb{R}^d$ *generic* if at most $d + 1$ tiles meet in a common point. We claim that the nerve of a generic tiling is a flag complex. This would include the permutahedral scenario considered in [2].

References

1 Karol Borsuk. On the imbedding of systems of compacta in simplicial complexes. *Fundamenta Mathematicae*, 35(1):217–234, 1948.

2 Aruni Choudhary, Michael Kerber, and Sharath Raghvendra. Polynomial-sized topological approximations using the permutahedron. In *32nd International Symposium on Computational Geometry, SoCG 2016*, pages 31:1–31:16, 2016.

3 Branko Grünbaum and G C Shephard. *Tilings and Patterns*. W. H. Freeman & Co., New York, NY, USA, 1986.

4 Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

5 Günter Rote, Francisco Santos, and Illeana Streinu. Pseudo-triangulations - a survey. In Jacob E. Goodman, János Pach, and Richard Pollack, editors, *Surveys on Discrete and Computational Geometry-Twenty Years Later.*, Contemporary Mathematics, pages 343–410. American Mathematical Society, 2008.

NP-Completeness of Max-Cut for Segment Intersection Graphs

Oswin Aichholzer[1], Wolfgang Mulzer[2], Partick Schnider[3], and Birgit Vogtenhuber[1]

1 **Institute of Software Technology, Graz University of Technology, Austria.**
`oaich@ist.tugraz.at`, `bvogt@ist.tugraz.at`
2 **Institut für Informatik, Freie Universität Berlin, Germany.**
`mulzer@inf.fu-berlin.de`
3 **Department of Computer Science, ETH Zürich, Switzerland.**
`patrick.schnider@inf.ethz.ch`

Abstract

We consider the problem of finding a *maximum cut* in a graph $G = (V, E)$, that is, a partition $V_1 \dot\cup V_2$ of V such that the number of edges between V_1 and V_2 is maximum. It is well known that the decision problem whether G has a cut of at least a given size is in general NP-complete. We show that this problem remains hard when restricting the input to *segment intersection graphs*. These are graphs whose vertices can be drawn as straight-line segments, where two vertices share an edge if and only if the corresponding segments intersect. We obtain our result by a reduction from a variant of Planar Max-2-SAT that we introduce and also show to be NP-complete.

1 Introduction

For a graph $G = (V, E)$, consider a partition $V = V_1 \dot\cup V_2$ of V. The set $E_{12} \subseteq E$ of edges with one endpoint in V_1 and one endpoint in V_2 is called a *cut* (induced by V_1 and V_2), and the cardinality $|E_{12}|$ is called the *size* of the cut. A *maximum* cut of G is a cut whose size is as large as possible. The problem MaxCut is to find the size of a maximum cut in a given graph G. MaxCut can also be cast as a vertex coloring problem: what is the maximum number of bichromatic edges that can be obtained by coloring each vertex with one of two possible colors? The decision version of MaxCut asks whether G contains a cut of size at least k, for a given $k \in \mathbb{N}$. It is NP-complete for general graphs [3]. Moreover, MaxCut is hard to approximate [7, 8]. On the other hand, there exists a PTAS for MaxCut in dense graphs [1]. For planar graphs, MaxCut can be solved in polynomial time [6], and the same is true for several other graph classes [2].

A *segment intersection graph* is a graph whose vertices can be drawn as straight-line segments (that pairwise intersect in at most one point, in their relative interiors), such that two vertices share an edge if and only if the corresponding segments intersect. In a (representation of a) segment intersection graph, a maximum cut corresponds to a 2-coloring of the segments such that the number of bichromatic crossings, i.e., crossings of segments with different colors, is maximum. So far, the complexity status of MaxCut on line segment intersection graphs seems to be open [2]. We show that the decision version of MaxCut is NP-complete even when the input is restricted to segment intersection graphs. We obtain this result via a reduction from a variant of Planar Max-2-SAT, that we introduce and show to be NP-complete as well in Section 2.

This project has been supported by the Austrian Science Fund (FWF) grant W1230 and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

In addition to the intrinsic interest of the problem, our study is motivated by the following question that was posed by Ruy Fabila-Monroy at the workshop "Reunión de Optimización, Matemáticas y Algoritmos" in the framework of the project CONNECT: let D be a straight-line drawing of the complete graph K_n on n vertices. A *k-edge-coloring* χ of K_n assigns to each edge of K_n a color from $\{1, \ldots, k\}$. Let $\bar{cr}_k(D, \chi)$ be the number of monochromatic edge crossings in D for the χ, that is, crossings of edges with the same color. What is the best drawing D and the best k-edge-coloring χ of K_n in order to minimize $\bar{cr}_k(D, \chi)$? During the workshop, Francisco Javier Zaragoza Martínez observed the following relation to maximum cuts: For a fixed drawing D, the total number of crossings is fixed. Thus, a k-edge-coloring χ with the minimum number of monochromatic crossings maximizes the number of bichromatic crossings. Further, any geometric graph can be interpreted as a segment intersection graph. Hence finding a 2-edge coloring of K_n with the minimum number of monochromatic crossings is equivalent to finding a maximum cut in the segment intersection graph D. We remark that our construction does not show hardness of MaxCut for straight-line drawings of K_n.

2 Planar Max-2-SAT

We will use a reduction from a variant of Max2Sat. In Max2Sat, we are given a Boolean formula ϕ in conjunctive normal form (CNF) with at most two literals per clause and an integer k. We need to determine whether there is an assignment to the variables of ϕ that satisfies at least k clauses. Max2Sat it NP-complete [4]. We will consider a variant of Max2Sat where we require the 2-CNF formula ϕ to be *planar* and *clause-tree-linked*, two notions that we will now define.

Given a CNF formula ϕ with clause set C and variable set V, the *incidence graph* $G_\phi = (C \cup V, E)$ is the graph that contains an edge between a variable and a clause if and only if the variable or its negation appear as a literal in the clause. We say that ϕ is *planar* if G_ϕ is a planar graph. The problems Planar 3-SAT and Planar Max-2-SAT are 3-SAT and Max2Sat restricted to planar formulas. Planar 3-SAT is NP-complete [9]. To see that Planar Max-2-SAT is NP-hard, it can be checked that the reduction from 3-SAT to Max-2-SAT in [4] preserves planarity; see for example Theorem 2 in [5] and the proof of Theorem 2.1 below.

For Planar 3-SAT, we can enforce even more conditions without making the problem tractable: we say that a planar 3-CNF formula ϕ is *clause-linked* if there exists a path P connecting the clauses in $G(\phi)$ such that $G(\phi) \cup P$ is still a planar graph. Clause-Linked Planar 3-SAT, which is 3-SAT restricted to clause-linked planar formulas, is still NP-complete, see for example [10].

Similarly, we can add more conditions on Max-2-SAT: we say that a planar 2-CNF formula ϕ is clause-tree-linked if there exists a spanning tree T of the clauses in $G(\phi)$ such that $G(\phi) \cup T$ is still a planar graph. We define Clause-Tree-Linked Planar Max-2-SAT as Max-2-SAT restricted to clause-tree-linked planar formulas.

▶ **Theorem 2.1.** Clause-Tree-Linked Planar Max-2-SAT *is NP-complete.*

Proof. To show NP-completeness of Clause-Tree-Linked Planar Max-2-SAT, we need to show its membership in NP and its NP-hardness. Membership in NP directly follows from the fact that Clause-Tree-Linked Planar Max-2-SAT is a special case of the NP-complete problem Max-2-SAT. We prove NP-hardness by reduction from Clause-Linked Planar 3-SAT.

In Clause-Linked Planar 3-SAT, we have as input a 3-CNF formula ϕ with variable set V and clause set C, together with a linear ordering o of the elements of C. Further,

the incidence graph $G(\phi) = (C \cup V, E)$ together with the path $P(o) = (C, E_P)$ on C that is induced by the linear ordering o is still planar. To transform this input to an input of Clause-Tree-Linked Planar Max-2-SAT, we utilize the following reduction function of the well known reduction from 3-SAT to Max-2-SAT [4]: Every clause $c = (x, y, z)$ in ϕ is replaced by a 2-CNF formula c' of the form

$$c' := x \wedge y \wedge z \wedge w \wedge (\neg x \vee \neg y) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg w) \wedge (y \vee \neg w) \wedge (z \vee \neg w),$$

where w is an additional variable that is used exclusively used for one clause of ϕ. The complete 2-CNF formula for the Max-2-SAT is then $\phi' := \bigwedge_{c \in C} c'$. The target value for the number of clauses that should be satisfied in ϕ' is $k' := 7|C|$. The reduction from 3-SAT to Max-2-SAT follows from the fact any variable assignment that does not satisfy a clause c in ϕ satisfies at most six of the clauses in c', while an assignment satisfying c satisfies exactly seven clauses in c'. What remains to be proven is that the resulting incidence graph $G'(\phi')$ admits a tree $T(o) = (C', E_T)$ such that $G'(\phi')$ together with $T(o)$ is still a planar graph. To this end, consider a plane embedding D of the graph $G(\phi)$ together with the path $P(o) = (C, P)$. We first construct a plane embedding[1] of $G'(\phi')$ from D. For a clause $c = (x \vee y \vee z)$ in ϕ, the sub graph in $G(\phi)$ induced by c and its variables x, y, and z is a tree with center c and leaves x, y, and z; see Figure 1 (left). To obtain an embedding of $G(\phi')$, we start with the embedding of $G(\phi)$. For every clause c in ϕ replace the tree of c (and its variables) by an embedding of the sub graph induced by c' (and its variables) in $G'(\phi')$ as depicted in Figure 1 (right). Because the variable vertices x, y and z all lie in the unbounded face of this drawing, the resulting embedding of $G'(\phi')$ is again plane.

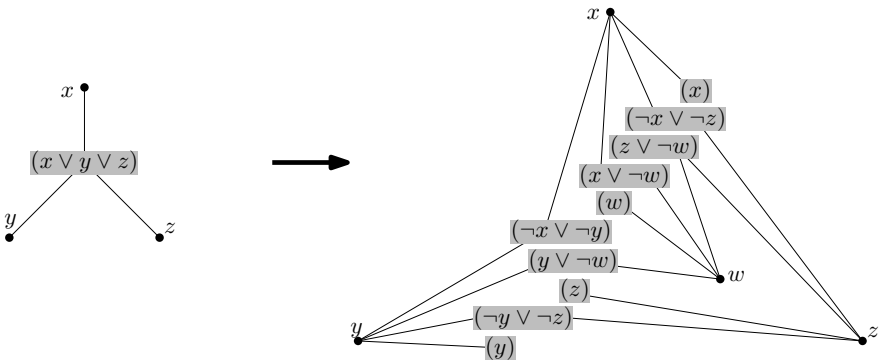


Figure 1 The subgraph of a clause c and its variables in ϕ (left), and the according subgraph of the transformation c' and its variables (right). Variable vertices are drawn as dots while clause vertices are drawn shaded.

Further, in $P(o)$, c is incident to one or two edges going to its neighbor(s) in the linear order o on C. We extend the drawing of $P(o)$ in D to a drawing of a tree through all clauses of ϕ' in D' such that the total drawing remains plane. It is easy to see that the drawing in Figure 1 (right) can be extended by a path P' through all the clauses that starts and ends in the unbounded face. If c is an endpoint of $P(o)$ and in D, the edge of $P(o)$ at c is between the ones to z and x or y, respectively, then we replace c in the drawing $P(o)$ by (x) or (y), respectively, and append P' to the drawing of $P(o)$. If in D, the two path edges

[1] It has been known that the reduction from 3-SAT to Max-2-SAT preserves planarity [5]. We reprove the statement via a concrete embedding, which we then utilize to also show clause-tree-linkedness.

at c are neighboring and between the ones to z and x or y, respectively, then we replace c in the drawing $P(o)$ by (x) or (y), respectively, and append P' as a branch to the drawing of $P(o)$. If in D, the path separates z from x and y in the order around c, then we replace the vertex c in the drawing of $P(o)$ by the path P'. Finally, note that the drawing of c' and its variables is not symmetric, but c' itself is. Hence, an appropriate permutation of x, y, and z in the drawing always yields a drawing of c' that fits one of the above cases. This finishes the reduction. ◀

3 Max-Cut for Segment Intersection Graphs

▶ **Theorem 3.1.** *The decision version of the* Max-Cut *problem is NP-complete even when restricted to segment intersection graphs.*

Proof. We prove NP-hardness by reduction from Clause-Tree-Linked Planar Max-2-SAT. For any clause-tree-linked planar 2-SAT formula ϕ with m clauses we construct a line segment arrangement S with the property that there is an assignment satisfying at least $m - k$ clauses of ϕ if and only if there is a 2-coloring of the segments of S with at most $m + 2k$ monochromatic crossings.

Let ϕ be a clause-tree-linked planar 2-SAT formula and let $G(\phi)$ be its associated graph and T the tree through its clauses. Consider a plane drawing of $G(\phi) \cup T$. We will mimic the formula ϕ by constructing line segment configurations, called *gadgets*, that serve as variables, wires, splits, negations and clauses, and concatenating them according to the drawing of the graph $G(\phi)$. We will use wire gadgets and split gadgets to propagate the truth assignment of a variable along the edges between the variable and the clauses containing it, while negation gadgets will serve to invert the truth assignment of a variable (for negative literals).

As variable gadget, we just take a single line segment. Each line segment will be colored with one of two colors, without loss of generality red and blue, one of them representing the true state, the other one the false state. For a wire gadget, we draw two segments a and b that do not cross each other and $2m + 1$ other segments, each of which crosses a and b but no other segment. See Figure 2 (left) for an illustration. It follows that if a and b get the same color, we can color the gadget without monochromatic crossings, whereas if a and b get different colors, any coloring of the remaining edges yields exactly $2m + 1$ monochromatic crossings. To build a split gadget, we repeat the construction of the wire gadget twice; see Figure 2 (middle). For the negation gadget, we again draw two segments a and b that do not cross each other. Further, we draw two families C and D of $2m + 1$ pairwise non-crossing segments each, such that each segment of C crosses a, each segment of D crosses b, and each segment of C crosses each segment of D; see Figure 2 (right). Note that for the negation gadget we have at least $2m+1$ monochromatic crossings if a and b have the same color. However, if a and b have different colors, this gadget can again be colored without monochromatic crossings.

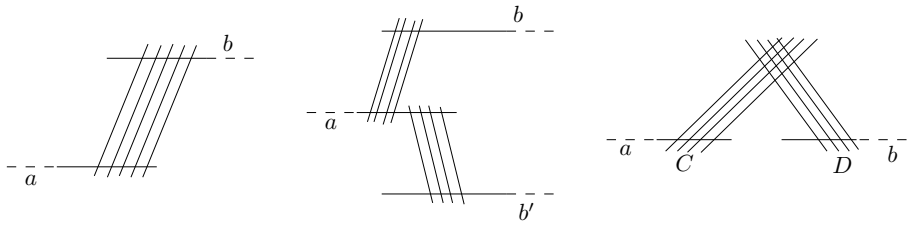


Figure 2 A wire gadget (left), a split gadget (middle) and a negation gadget (right).

It remains to construct the clause gadgets. For any two literals that form a clause, draw two corresponding segments a and b and a segment t, called *tree segment*, such that both a and b cross t. Further, we draw two additional segments c and d, where c crosses only a and d and d crosses only b and c. See Figure 3 for an illustration. Assume that t is colored red. If both a and b are blue, coloring c and d without obtaining a monochromatic crossing is impossible, but we can color c and d such that we have only one monochromatic crossing. The same holds if both a and b are red, but in this case there are also two monochromatic crossings between t, a, and b. If a is red and b is blue or vice versa, we have a monochromatic crossing between a or b and t, but we can color c and d such that they are not involved in any monochromatic crossing. So, to summarize, every clause requires at least one monochromatic crossing and we have a coloring with exactly one such crossing unless a and b have the same color as t, in which case the clause requires at least three monochromatic crossings. In our construction, the colors of the tree segments will represent the false state. Hence, any satisfied clause can be drawn with only one monochromatic crossing, while unsatisfied clauses require at least three monochromatic crossings.

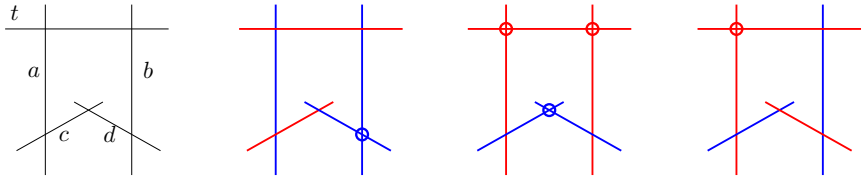


Figure 3 A clause gadget and some possible colorings of it. Monochromatic crossings are marked with small circles.

Using these gadgets, we construct a line segment arrangement that goes essentially along the edges of the given drawing of $G(\phi)$. To enforce that the tree segments have the same color, we connect them using wire gadgets according to the drawing of T. Let S be the line segment arrangement obtained by this construction. See Figure 4 for a small example.

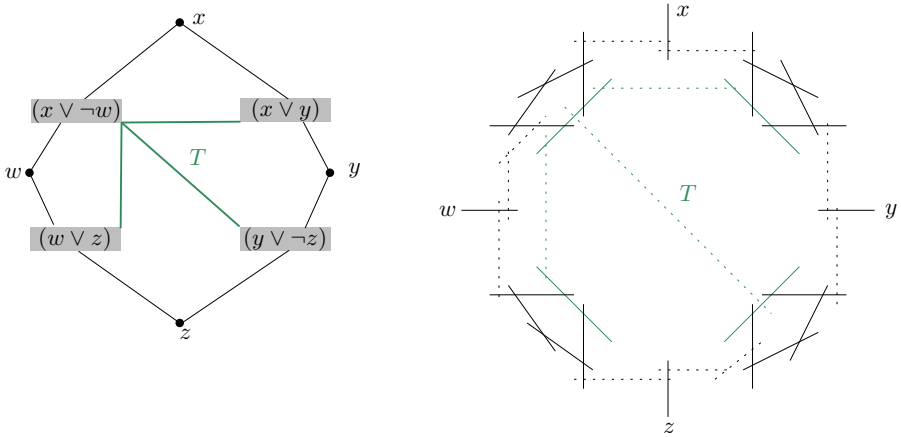


Figure 4 A drawing of $G(\phi)$ for the 2-SAT formula $\phi = (x \vee \neg w) \wedge (x \vee y) \wedge (w \vee z) \wedge (y \vee \neg z)$, with a tree T connecting the clauses (left) and the segment arrangement derived from this drawing (right). Dashed edges correspond to sets of $2m + 1$ line segments.

Next we show that there is an assignment satisfying at least $m-k$ out of the m clauses of ϕ if and only if there is a 2-coloring of the segments of S with at most $m + 2k$ monochromatic crossings, for any $0 \leq k \leq m$.

First assume that there is a 2-coloring of the segments of S with at most $2k + m \leq 3m$ monochromatic crossings. As each of the m clause gadgets needs at least one monochromatic crossing, at most k clause gadgets can have three (or more) monochromatic crossings. Furthermore, in each wire gadget, the segments corresponding to a and b in the illustration in Figure 2 (left) must have the same color. Otherwise the gadget alone would already contain at least $2m + 1$ monochromatic crossings and hence the whole drawing would contain at least $2m + 1 + m \geq 3m + 1$ monochromatic crossings, a contradiction. For the same reason, all tree segments have the same color and furthermore the segments corresponding to a, b, and b' in a split gadget share the same color; and in any negation gadget, the segments corresponding to a and b must have different colors. Hence, interpreting the color of the tree segments as representing the false state and assigning the truth states to the variables in ϕ according to the color of their respective variable gadgets, we obtain a variable assignment for ϕ with at most k unsatisfied clauses.

For the other direction, assume that there is an assignment satisfying at least $m - k$ clauses of ϕ. Color the variable gadgets blue if the corresponding variable is assigned the true state, and red otherwise. Color the tree segments in red and all the gadgets, except the clause gadgets, without monochromatic crossings. Then the only monochromatic crossings occur in the clause gadgets. Each of them induces one monochromatic crossing, and two more if and only if the corresponding clause is unsatisfied. As there are at most k unsatisfied clauses the coloring has at most $2k + m$ monochromatic crossings.

It is not hard to see that the line segment arrangement S can be constructed in polynomial time, which concludes the NP-hardness part. Furthermore, the problem is clearly in NP as it is a restricted version of the NP-complete problem Max-Cut, which finishes the proof. ◀

References

1. S. Arora, D. R. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *J. Comput. Syst. Sci.*, 58(1):193–210, 1999.
2. H. N. de Ridder et al. Information system on graph classes and their inclusions. `www.graphclasses.org`, 2016.
3. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
4. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976. URL: `http://www.sciencedirect.com/science/article/pii/0304397576900591`, `doi:https://doi.org/10.1016/0304-3975(76)90059-1`.
5. L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *Proceedings of the 2nd International Symposium on Algorithms*, ISA '91, pages 151–162, London, UK, UK, 1991. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=648003.743125`.
6. F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975. `doi:10.1137/0204019`.
7. J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
8. S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM J. Comput.*, 37(1):319–357, 2007.
9. D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. `doi:10.1137/0211025`.
10. A. Pilz. Planar 3-SAT with a clause/variable cycle. *CoRR*, abs/1710.07476, 2017. URL: `http://arxiv.org/abs/1710.07476`, `arXiv:1710.07476`.

Non-Monochromatic and Conflict-Free Colorings in Tree Spaces*

B. Aronov[1], M. de Berg[2], A. Markovic[2], and G. Woeginger[3]

1 **Department of Computer Science and Engineering, New York University, USA.**
2 **TU Eindhoven, the Netherlands.**
3 **RWTH Aachen, Germany.**

Abstract

We study non-monochromatic and conflict-free colorings on tree spaces, that is, one-dimensional spaces with a tree topology. More specifically, we analyze the number of colors needed to color a set $\mathcal{A}$ of n objects in a tree space $\mathcal{T}$ with k leaves, with each object being a connected subset of $\mathcal{T}$, in a non-monochromatic or conflict-free fashion. We prove that there exists a non-monochromatic coloring with $O(\min(\ell, \sqrt{k}))$ colors, where ℓ denotes the maximum number of leaves of any object in $\mathcal{A}$. This bound is tight in the worst case. This result implies that there exists a conflict-free coloring with $O(\ell \log k)$ colors.

1 Introduction

Conflict-free colorings, or CF-colorings for short, were introduced by Even *et al.* [4] and Smorodinsky [8] to model frequency assignment to base stations in wireless networks. In the basic setting one is given a set S of objects in the plane—often disks are considered—and the goal is to assign a color to each object such that the following holds: for any point p in the plane such that the set $S_p := \{D \in S \mid p \in D\}$ of objects containing p is non-empty, S_p must contain an object whose color is different from the colors of the other objects in S_p. Even *et al.* proved, among other things, that any set of disks admits a CF-coloring with $O(\log n)$ colors. Since then many different geometric variants of CF-colorings have been studied. For example, Har-Peled and Smorodinsky [5] generalized the result to objects with near-linear union complexity, while Even *et al.* [4] considered the dual version of the problem. See the survey by Smorodinsky [10] for an overview. A restricted type of CF-colorings are *unique-maximum colorings* (UM-colorings), in which the colors are identified with integers, and the maximum color in the set S_p is required to be unique. Another type of coloring, often used as an intermediate step to obtain a CF-coloring, is *non-monochromatic* (NM). In an NM-coloring—sometimes called *a proper coloring*—we only require that, for any point p in the plane, if the set S_p contains at least two elements, not all of them have the same color. Smorodinsky [9] showed that if an NM-coloring on n elements using $\beta(n)$ colors is given, one can create a CF-coloring using $O(\beta(n) \log n)$ colors.

CF-colorings can also be defined in a more abstract setting. Here one is given a hypergraph $\mathcal{H} = (V, E)$ and the goal is to color V such that for every (non-empty) hyperedge $e \in E$, there is a vertex in e whose color is different from that of the other vertices in e. Ashok *et al.* [2] showed that deciding whether a given hypergraph can be CF-colored using k colors is fixed-parameter tractable. Note that the basic geometric version mentioned above—coloring objects in $\mathbb{R}^2$ with respect to points—can be phrased in terms of hypergraphs by letting the objects be the vertex set V and for each point p in the plane creating a hyperedge $e := S_p$.

* MdB and AM are supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003. BA has been supported by NSF Grants CCF-11-17336, CCF-12-18791, and CCF-15-40656, and by BSF grant 2014/170.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Another avenue for constructing a hypergraph $\mathcal{H}$ to be colored is to start with a graph $\mathcal{G}$, let the vertices of $\mathcal{H}$ be the vertices of $\mathcal{G}$ and create hyperedges for (the sets of vertices of) certain subgraphs of $\mathcal{G}$. For example, Pach and Tardos [7] considered the case where hyperedges are all the vertex neighborhoods. For this case, Abel *et al.* [1] recently showed that a planar graph can always be colored with only three colors, if we allow some vertices to be uncolored. (Otherwise, we can use a dummy color, increasing the number of colors to four.) As another example, we let the hyperedges be induced by all the paths. This setting is equivalent to an older notion of *vertex ranking* [3], also known as *ordered coloring* [6].

In this paper we study CF- and NM-colorings in a setting that is closely related to both the geometric and the graph-based setting. More precisely, the spaces that we consider are *tree spaces*—that is, one-dimensional spaces with a tree topology—and the objects that we want to color are connected subsets (in other words, subtrees) of the given tree space. In this setting, we are interested in how the complexity of the given tree space and of the objects to be colored influence the chromatic number. Note that, if the given tree space is a single curve, the problem reduces to coloring intervals on the real line.

Our contributions. Let $\mathcal{T}$ be the given tree space. It may be convenient to visualize $\mathcal{T}$ as being embedded in $\mathbb{R}^2$, although the embedding is actually immaterial. We assume without loss of generality that $\mathcal{T}$ is bounded—it does not have infinitely long branches—and define the *vertices* of $\mathcal{T}$ in the natural manner. Any vertex of $\mathcal{T}$ is either an *internal vertex* (a branching point of degree at least three) or a *leaf*. The curves connecting the vertices, whose union is $\mathcal{T}$, are called the *edges* of the tree space. We denote the number of leaves of $\mathcal{T}$ by k.

Let $\mathcal{A}$ be the set of n objects that we wish to color, where each object $T \in \mathcal{A}$ is a connected subset of $\mathcal{T}$. Thus each object itself is also a tree. From now on, we will refer to the objects in $\mathcal{A}$ as "trees", and always use "tree space" when talking about $\mathcal{T}$. We denote the maximum number of leaves of any tree in $\mathcal{A}$ by ℓ. Note that internal vertices of a tree are necessarily internal vertices of $\mathcal{T}$, but leaves of a tree may also lie in the interior of an edge of $\mathcal{T}$. CF-colorings of such a set $\mathcal{A}$ are now defined as above: for any point $p \in \mathcal{T}$, the set $S_p := \{T \in \mathcal{A} \mid p \in T\}$ (if non-empty) should have a tree with a unique color. We now define the CF-chromatic number $X_{\text{cf}}^{\text{tree,tree}}(k, \ell; n)$ as the minimum number of colors sufficient to CF-color any set $\mathcal{A}$ of n trees of at most ℓ leaves each in a tree space of at most k leaves. The NM-chromatic number $X_{\text{nm}}^{\text{tree,tree}}(k, \ell; n)$ is defined similarly. We will show that[1]

▶ **Theorem 1.1** (Main result).
(i) $X_{\text{nm}}^{\text{tree,tree}}(k, \ell; n) \leqslant \min(\ell + 3, 2\sqrt{6k} + 2)$*, and (ii)* $X_{\text{cf}}^{\text{tree,tree}}(k, \ell; n) = O(\ell \log k)$.

In the full version we also (a) show how to use two fewer colors in part (i) of the theorem and (b) provide two lower bounds for NM-colorings, namely $X_{\text{nm}}^{\text{tree,tree}}(k, \ell; n) \geqslant \min\left(\ell + 1, \left\lfloor \frac{\sqrt{1+8k}}{2} \right\rfloor, n\right)$, which clearly also apply to CF-colorings, and $X_{\text{cf}}^{\text{tree,tree}}(k, \ell; n) \geqslant \lfloor \log_2 \min(k, n) \rfloor$; and (c) study other variants, for example by considering more general network spaces (rather than tree spaces) and other types of objects to be colored.

2 The coloring algorithms

Preliminaries: The chain method. We start by describing a folklore technique, called the *chain method*, to color intervals in $\mathbb{R}^1$ in a non-monochromatic fashion using at most two

[1] Obviously the number of trees, n, is an upper bound as well. To avoid cluttering the bounds, we usually omit this trivial bound.

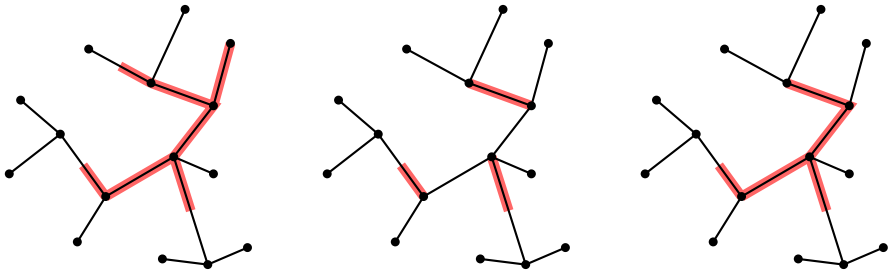


Figure 1 The original tree T (left), the set $\bigcup_{e \in E(T)} e \cap T$ (middle), and the new tree T' (right).

colors. We order the intervals left-to-right by their left endpoints (in case of ties, we take the longest interval first) and color them in this order using the so-called *active color* which is defined as follows. We start with blue as the active color. We color the first interval, then change the active color to red. We then use the following procedure: we color the next interval I in the ordering using the active color, then if the right endpoint of I is not contained in any other colored interval, we change the active color from red to blue or blue to red. It is easy to show the resulting coloring is non-monochromatic.

Overview of the coloring procedure. Let $\mathcal{T}$ be a tree space and let $\mathcal{A}$ be a set of n trees on $\mathcal{T}$, each with at most ℓ leaves. We will NM-color $\mathcal{A}$ in two phases: first, we select a subset $\mathcal{C} \subseteq \mathcal{A}$ of size at most $6k-12$ and color it with at most $\min(\ell+1, 2\sqrt{6k})$ colors. In the second phase we extend this coloring to the whole set $\mathcal{A}$ using at most two extra colors.

An edge e of $\mathcal{T}$ is a *leaf edge* if it is incident to a leaf; the remaining edges are *internal*. We define $\mathcal{C} \subseteq \mathcal{A}$ as the set of at most $6k-12$ trees selected as follows. For every pair (e, v), where e is an edge of $\mathcal{T}$ and v is an endpoint of e that is not a leaf of $\mathcal{T}$, we choose two trees containing v and extending the furthest into e (if they exist), that is, trees T of $\mathcal{A}$ containing v for which $\text{length}(T \cap e)$ is maximal, and place them in $\mathcal{A}(e, v)$. Note that if two or more trees of $\mathcal{A}$ fully contain e, then $\mathcal{A}(e, v)$ contains two of them, chosen arbitrarily. Note also that, if a tree contains an internal edge e fully, it may be chosen by both endpoints. We now define $\mathcal{A}(e) := \mathcal{A}(e, u) \cup \mathcal{A}(e, v)$ for each internal edge $e = \{u, v\}$, define $\mathcal{A}(e) := \mathcal{A}(e, v)$ for each leaf edge $e = \{u, v\}$ with v being its non-leaf endpoint. Finally, we define $\mathcal{C} := \bigcup \mathcal{A}(e)$, with the union taken over all edges e of $\mathcal{T}$. Since $\mathcal{A}(e)$ contains at most four trees for any internal edge e and at most two trees for any leaf edge e, and since the number of internal edges of $\mathcal{T}$ is at most $k-3$ and the number of leaf edges is at most k, where k is the number of leaves of $\mathcal{T}$ (which, as a topological tree, does not have degree-two vertices), $|\mathcal{C}| \leqslant 6k-12$, as claimed. We first explain how to color $\mathcal{C}$.

Coloring $\mathcal{C}$. We color $\mathcal{C}$ in two steps. Let $E(T)$ be the set of edges e of $\mathcal{T}$ with $T \in \mathcal{A}(e)$. Firstly, if $\ell > 2\sqrt{6k}$ we select all subtrees T with $|E(T)| \geqslant \sqrt{6k}$, and give each of them a unique color. Since $\sum_e |\mathcal{A}(e)| \leqslant 6k-12$ there are at most $\sqrt{6k}-1$ such trees, so we use at most $\sqrt{6k}-1$ colors. Then for each uncolored $T \in \mathcal{C}$ we create a new tree T', defined as the smallest tree containing $\bigcup_{e \in E(T)} e \cap T$; see Fig. 1. Note that T' has at most $\ell' := \min(\ell, \sqrt{6k})$ leaves because $|E(T)| < \sqrt{6k}$. Define $\mathcal{C}' := \{T' \mid T \in \mathcal{C}\}$. The second step is to color $\mathcal{C}'$. We need the following lemma, which shows that an NM-coloring of $\mathcal{C}'$ carries over to $\mathcal{C}$.

▶ **Lemma 2.1.** *Any NM-coloring of $\mathcal{C}'$ corresponds to an NM-coloring of $\mathcal{C}$, that is, if we give each tree $T \in \mathcal{C}$ the color of the corresponding tree $T' \in \mathcal{C}'$ then we obtain an NM-coloring.*

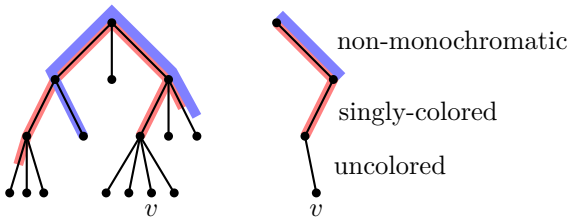


Figure 2 A coloring of trees (left) and an illustration of the invariant for v (right).

Proof. Let q be a point on an edge e of $\mathcal{T}$ contained in at least two trees of $\mathcal{C}$ (if no such trees exists, the coloring is trivially non-monochromatic at q). Since q is contained in at least two trees of $\mathcal{C}$, it is also contained in two trees of $\mathcal{A}(e)$. Call these trees T_1 and T_2. Note that T_1 either receives a color in the first coloring step—namely when $|E(T_1)| \geqslant 2\sqrt{6k}$—or $T_1' \in \mathcal{C}'$ contains q (since $e \in E(T_1)$). A similar statement holds for T_2. Since the colors used in the first step are unique and $\mathcal{C}'$ is NM-colored, this implies that T_1 and T_2 have different colors. Hence, $\mathcal{C}$ is NM-colored. ◀

Next we show how to NM-color $\mathcal{C}'$. Fix an arbitrary root r of the tree space $\mathcal{T}$. Our coloring procedure for $\mathcal{C}'$ maintains the following invariant: any path from r to a leaf v of $\mathcal{T}$ consists of three disjoint consecutive subpaths (some possibly empty), in this order, as illustrated in Fig. 2:

- a *non-monochromatic* subpath containing the root on which at least two trees are colored with at least two different colors,
- a *singly-colored* subpath containing exactly one colored tree, and
- an *uncolored* subpath containing the leaf on which no tree is colored.

▶ **Observation 2.2.** *Any set of trees containing r and satisfying the invariant described above is NM-colored if we disregard uncolored trees.*

We color the trees $T \in \mathcal{C}'$ that contain r in an arbitrary order, using $\ell' + 1$ colors, as follows: for each leaf v of T, we follow the path from v to the root r to find a singly-colored part. Note that if we find a singly-colored part—by the invariant there is at most one such part on the path from v to r—we cannot use that color for T. Since T has at most ℓ' leaves, this eliminates at most ℓ' colors. Hence, at least one color remains for T.

▶ **Lemma 2.3.** *The procedure described above maintains the invariant and colors all trees of $\mathcal{C}'$ containing r with at most $\ell' + 1$ colors.*

Proof. Suppose the invariant holds before the coloring of T. Then we need to make sure the invariant still holds after T has been colored. Let w be a leaf of $\mathcal{T}$ and π_w the path from w to the root. If π_w does not contain a leaf of T then the invariant obviously still holds on π_w. Now suppose π_w contains a leaf v of T, and let $\pi_v \subseteq \pi_w$ be the path from v to r. The part of π_v that was uncolored (if it was non-empty) now is singly-colored. The part that was singly-colored now becomes non-monochromatic, as we eliminated that color for T. And the part that was already non-monochromatic stays so. Therefore the invariant is indeed maintained for π_w, concluding the proof. ◀

Once all the trees containing r are colored we delete r from $\mathcal{T}$, that is, we consider the space $\mathcal{T} \setminus \{r\}$, and we take the closures of the resulting connected components. This creates a number of subspaces such that each uncolored tree in $\mathcal{C}'$ is contained in exactly one of

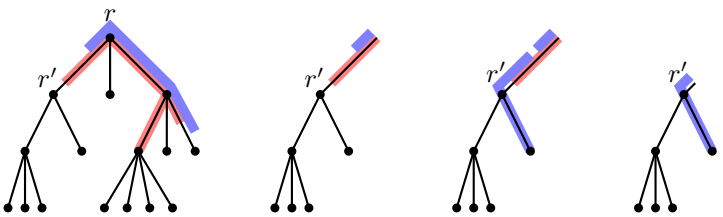


Figure 3 When recursing on the subspace rooted at r' (leftmost), the invariant does not hold anymore (middle left), as the parts are switched on the edge between r and r'. To remedy this, we first color the tree extending the furthest into that edge (middle right), starting from r'. We then trim the tree to fix the invariant (rightmost).

them. Consider such a subspace $\mathcal{T}'$ and let r' be the neighbor of r in $\mathcal{T}'$. We now want to recursively color the uncolored trees in $\mathcal{T}'$, taking r' as the root of $\mathcal{T}'$. However, the invariant might not hold on the edge e from r' to the old root r: Since now r is considered a child of r', the order of the three parts might switch on e—see Fig. 3. Suppose this is the case, and let c_e be the color of the singly-colored part on the edge e. Note that for the order to switch, the non-monochromatic part needs to end on e, and therefore the only color used in any singly-colored part of the tree rooted at r' is c_e. We overcome this problem by carefully choosing the order in which we color the trees containing r'. Namely, we first color the tree T extending the furthest in e. In this case, there is only one color forbidden, namely c_e. We can therefore easily color T. We can then trim the tree space $\mathcal{T}'$ to remove any non-monochromatic part and hence restore the invariant and continue with the coloring.

▶ **Lemma 2.4.** *$\mathcal{C}$ admits an NM-coloring with $\min(\ell+1, 2\sqrt{6k})$ colors.*

Proof. The fact that the procedure above produces an NM-coloring follows from Lemmas 2.1 and 2.3. When $\ell > 2\sqrt{6k}$ we use $\sqrt{6k}-1$ colors to deal with trees T with $|E(T)| \geqslant \sqrt{6k}$ and $\ell'+1 \leqslant \min(\ell, 2\sqrt{6k})+1 \leqslant \sqrt{6k}+1$ colors for the other trees, giving $2\sqrt{6k}$ colors in total. When $\ell \leqslant 2\sqrt{6k}$ we do not treat the trees with $|E(T)| \geqslant \sqrt{6k}$ separately, so we just use $\ell'+1 \leqslant \min(\ell, \sqrt{6k})+1 \leqslant \ell+1$ colors. ◀

Extending the coloring from $\mathcal{C}$ to $\mathcal{A}$. Let $c\colon \mathcal{C} \to \mathbb{N}$ be an NM-coloring on $\mathcal{C}$. We extend the coloring to $\mathcal{A}$ as follows. We start by coloring all trees containing an internal vertex of $\mathcal{T}$ using an arbitrary color already used. Then, for each edge $e = \{r, r'\}$ we color the set of uncolored trees contained in e using the chain method. For this we use two new colors, which are used for all chains—we can re-use the same two colors for the chains, since trivially the chains in any two edges e, e' do not interact. (In the full version we describe a more careful approach, which avoids using two new colors.) The following lemma proves the extended coloring is non-monochromatic.

▶ **Lemma 2.5.** *Any NM-coloring c on $\mathcal{C}$ can be extended to $\mathcal{A}$ by using two extra colors.*

Proof. Let $\mathcal{A}_1$ be the subset of trees in $\mathcal{A} \setminus \mathcal{C}$ that contain an internal vertex of $\mathcal{T}$, and let $\mathcal{A}_2$ be the remaining trees in $\mathcal{A} \setminus \mathcal{C}$. By Lemma 2.4 we have an NM-coloring on $\mathcal{C}$, and the chain method gives us an NM-coloring for the trees in $\mathcal{A}_2$ using two additional colors. It is easy to see that together this gives us an NM-coloring on $\mathcal{C} \cup \mathcal{A}_2$. The trees in $\mathcal{A}_1$ received an arbitrary color already used. To prove that this gives an NM-coloring for $\mathcal{A} = \mathcal{C} \cup \mathcal{A}_1 \cup \mathcal{A}_2$,

it suffices to prove that each tree $T \in \mathcal{A}_1$ is *doubly-covered* by $\mathcal{C}$, that is, any point $q \in T$ is contained in at least two trees in $\mathcal{C}$. To this end, let e be an edge such that $q \in e$. Then, since $T \notin \mathcal{C}$ and T contains an endpoint v of e, the two trees in $\mathcal{A}(e, v)$ contain q. Hence, T is doubly-covered by $\mathcal{C}$, as claimed. ◀

Proof of Theorem 1.1. For the NM-coloring part of the theorem, we use Lemmas 2.4 and 2.5. For the second part, if $\ell > 2\sqrt{6k}$ we again reduce $\mathcal{C}$ to $\mathcal{C}'$ using at most $\sqrt{6k} - 1$ colors. Then use the result by Smorodinsky [9] on the NM-coloring on $\mathcal{C}'$ provided by Lemma 2.3. Since this coloring uses at most $\ell' + 1$ colors and $|\mathcal{C}'| \leqslant 6k - 12$, the CF-coloring uses $O(\ell \log k)$ colors. We then extend the coloring to $\mathcal{A}$ using similar techniques as for the NM-coloring. This coloring uses $O(\sqrt{k} \log k)$ colors if $\ell > 2\sqrt{6k}$, which is in $O(\ell \log k)$, and directly $O(\ell \log k)$ colors otherwise. Note that a direct application of the result by Smorodinsty [9] would give a $O(\ell \log n)$ bound instead. ◀

3 Concluding remarks

We studied NM- and CF-colorings on tree spaces, where the objects to be colored are connected subsets of the tree space. We showed that the number of colors can be bounded as a function of the complexity (that is, number of leaves) of the tree space and the objects, rather than on the number of objects. In the full version we show that this is also the case for balls on network spaces. It would be interesting to find more settings where this is the case.

References

1. Z. Abel, V. Alvarez, E. D. Demaine, S. P. Fekete, A. Gour, A. Hesterberg, P. Keldenich, and C. Scheffer. Three colors suffice: Conflict-free coloring of planar graphs. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1951–1963, 2017.
2. P. Ashok, A. Dudeja, and S. Kolay. Exact and FPT algorithms for max-conflict free coloring in hypergraphs. In *Proceedings of the 26th International Symposium of Algorithms and Computation*, pages 271–282, 2015.
3. H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Z. Tuza. Ranking of graphs. In *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 292–304, 1994.
4. G. Even, Z. Lotker, D. Ron, and S. Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003.
5. S. Har-Peled and S. Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry*, 34(1):47–70, 2005.
6. M. Katchalski, W. McCuaig, and S. M. Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995.
7. J. Pach and G. Tardos. Conflict-free colourings of graphs and hypergraphs. *Combinatorics, Probability & Computing*, 18(5):819–834, 2009.
8. S. Smorodinsky. *Combinatorial Problems in Computational Geometry*. PhD thesis, Tel-Aviv University, 2003.
9. S. Smorodinsky. On the chromatic number of some geometric hypergraphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 316–323, 2006.
10. S. Smorodinsky. Conflict-free coloring and its applications. In I. Bárány, K. J. Böröczky, G. F. Tóth, and J. Pach, editors, *Geometry — Intuitive, Discrete, and Convex: A Tribute to László Fejes Tóth*, pages 331–389. Springer Berlin Heidelberg, 2013. See also arXiv:abs/1005.3616.

Fair Voronoi Split-Screen for N-Player Games

Tobias Lenz[1]

1 **Hochschule für Technik und Wirtschaft Berlin**
`tobias.lenz@htw-berlin.de`

Abstract

A consistent expansion of the well-spread dynamic two player split-screen to a larger number of players is introduced and formally defined. Unfortunately such a pure solution does not exist, as is proven in this paper. A visually appealing approximation is presented and discussed.

1 Introduction

1.1 Motivation through Multiplayer Games

In the early days of 2d computer games, most local multiplayer games were played on a single screen. Newer gaming consoles revived this scenario for multiple players in front of a single television set or console display.

This leaves two options to the developer: either all players have to stay very close to each other in the virtual realm, or each player has its own independent window to the game's world and is allowed to stroll around freely. The former case severely limits gameplay while the latter case requires a subdivision of the physical screen space into as many independent windows as players participating. The most obvious and often used subdivisions for two players are horizontally or vertically in the respective center. Applying both subdivisions simultaneously solves the typical four player scenario.

The mentioned stationary horizontal and/or vertical subdivisions have multiple drawbacks, we are going to tackle in this paper:

1. If two players stand right next to each other, both their windows would show the exact same surroundings, essentially wasting half of the total screen space.
2. Multiplayer games might want to hint the players on their relative positions, e.g. player one being left and slightly above player two. This requires additional display elements like arrows, further cluttering the screen.

1.2 Dynamic Split-Screens to the Rescue

The two problems mentioned above can be fixed with a single simple concept often called "dynamic split-screen". A simple variation of it already appeared in a game in 1983 [2] basically only solving the first mentioned drawback. Developers sporadically used and improved it to also solve the second drawback ever since without much scientific interest. It realizes the simple idea, that the separating line between two players need not be static but might change and even vanish according to the player positions, as indicated in figure 1.

For two players there is not a lot of math to it. The separator in screen space is perpendicular to the vector between the player positions in world space, and can therefore be computed with a single arctan.

1.3 Formal Problem Formulation

Note that there is no ground truth. All criteria were chosen with aesthetics, fluent graphics and gameplay in mind.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

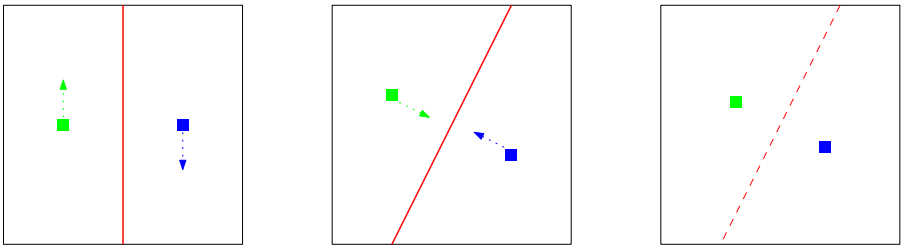


Figure 1 The separator (red) changes its orientation dynamically according to the player positions (green, blue) while they move (arrows). It will vanish for players being close to each other (right image).

The following definition is helpful to simplify writing.

▶ **Definition 1.1.** Let the sum of a polygon P and a vector v be the polygon translated by that vector, i.e. the vector added to every point of the polygon: $P + v = \{p + v \mid p \in P\}$.

Today game worlds are massive in size but usually not infinite. We consider them finite and surrounded by thick impassable walls, so we can treat them as infinite here.

▶ **Definition 1.2.** The *world space* is $\mathbb{R}^2$. Players move continuously through the world with position $w_i(t) \in \mathbb{R}^2$ for player i at time t.

The display hardware has a fixed amount of pixels and a given aspect ratio, but we abstract this and use a square with real coordinates.

▶ **Definition 1.3.** The *screen space* is $[-1, 1]^2$. The screen space position for player i at time t is denoted by $s_i(t) \in [-1, 1]^2$.

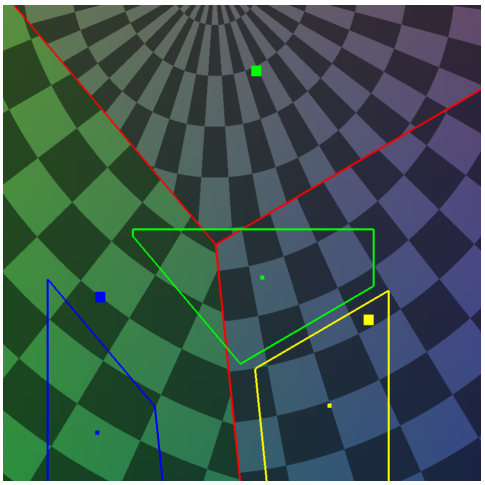
Since every coordinate used here is time dependent, the parameter t is omitted throughout the paper to avoid cluttering. A "scene" with fixed t is considered and the following main definition must hold for every t.

▶ **Definition 1.4.** A *fair voronoi split-screen* for n players comprises a set of n convex polygons $S_0, \ldots, S_{n-1}$ forming a subdivision of the screen space, one for each player, and one designated point (screen space position) inside each S_i, fulfilling the following criteria:

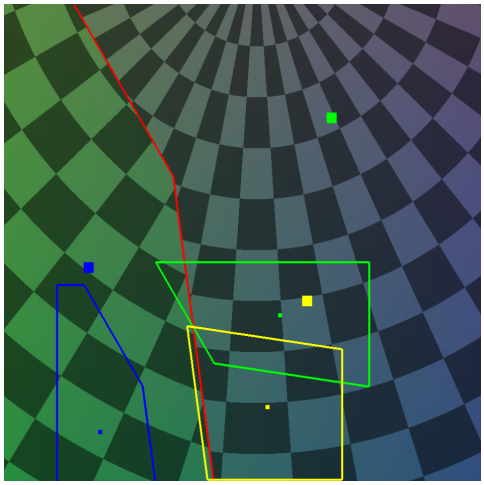
1. **Fair**: All S_i have equal area.
2. **Direction-indicating**: If S_i and S_j share a boundary, this boundary would be parallel to the bisector of w_i and w_j.
3. **Fusible**: If S_i and S_j overlap in world space, formally $(S_i - s_i + w_i) \cap (S_j - s_j + w_j) \neq \emptyset$, the boundary between S_i and S_j would be omitted, thereby *fusing* S_i and S_j.
4. **Centered**: s_i is the center of the inscribed circle of S_i for all non-fused S_i. If S_i is fused with one or more other polygons $S_j, S_k, \ldots$, the centroid c_i^W of $w_i, w_j, w_k, \ldots$ is mapped to the centroid c_i^S of the inscribed circles of $S_i, S_j, S_k, \ldots$ and $s_i \leftarrow c_i^S + w_i - c_i^W$.
5. **Continuous**: Just as w_i moves continuously, so too s_i and the boundary vertices of S_i.

Fairness is obvious from a gameplay perspective, direction-indication and fusibility are the requested features from section 1.1, centeredness helps providing good visibility in every direction, and continuity is required for aesthetics and fluent animation.

Figure 2 illustrates parts of the definition. The three equally sized (fair) S_i are depicted including the corresponding s_i (centered). The red boundaries have the correct angles (direction-indicating). In figure 2b two screen space regions are fused.



(a) All three players are sufficiently far away from each other in world space, resulting in three disjoint screen space regions.



(b) The screen space regions for the green and yellow player overlap in world space, hence they become visually fused in screen space.

Figure 2 Three players (thick squares in green, blue, yellow) with their respective region in screen space, the shared boundaries of the S_i in red, and the relative positions of the S_i projected to world space (colored polygons at the bottom).

Parts of definition 1.4 resemble the well-known *voronoi diagram* [3]. Hence the name *voronoi split-screen* became commonly accepted. Do not jump to conclusions because observation 1.6 tells us that we are not dealing with normal voronoi diagrams here.

▶ **Observation 1.5.** *The* dynamic split-screen *for two players mentioned in section 1.2 equals the voronoi diagram of the two player positions and fulfills definition 1.4.*

▶ **Observation 1.6.** *The* voronoi diagram *of $n > 2$ players in world space scaled uniformly to screen space usually violates the fairness and centeredness conditions, while providing direction-indication and being fusible. Therefore it is not a fair voronoi split-screen.*

1.4 Related Results

The two player dynamic/voronoi split-screen appears quite often in games without special emphasis, but only a few approaches for more than two players exist.

At GDC 2016 Eiserloh presented an approach [4]. They build the voronoi diagram of the player positions in world space, map it to screen space, and reposition the player in screen space to be the center of the inscribed circle in their corresponding area. Although the last step guarantees a nice centeredness, the split-screen is neither fair, nor continuously fusible.

A different implementation, utilizing only the GPU, was made freely available by an author with the pseudonym gorsman [5]. The voronoi diagram is used directly and hence the cells are nicely fusible, but the split-screen is neither fair, nor centered.

Both mentioned approaches are expandable to an arbitrary number of players without further complications. They lack fairness and either centeredness or fusibility.

2 Fair Voronoi Split-Screens are Almost Impossible

The fact that no fair split-screen by definition 1.4 is known for at least three players becomes quite understandable, as the following theorem states their non-existence.

The proof is quite lengthy, so it is given in two lemmas. Both consider the following case to derive a contradiction:

Let three players be positioned in world space at $w_0 = (0, 2), w_1 = (0, -2), w_2 = (100, 0)$. w_0 and w_1 are close to each other, but their distance is slightly larger than the screen space's side length. w_2 is very far to the right, vertically between w_0 and w_1. Obviously no two regions can overlap in world space, hence fusion is prohibitive in this setting.

▶ **Lemma 2.1.** *S_0, S_1, S_2 all pairwise share a bounding edge ("they are neighbors").*

Proof. Since S_0, S_1, S_2 are convex polygons with equal area and they form a subdivision of a square, either all are neighbors and we are done, or one polygon must separate the other two. Since the angles of possible boundary edges are fixed by the direction-indication property, one can go through all three cases and show that a correct subdivision is impossible under these circumstances. Illustrated in figure 3a is the case where S_2 should separate S_0 and S_1. Hence no polygon can separate the other two. ◀

▶ **Lemma 2.2.** *Fair Voronoi Split-screens are impossible for three players.*

Proof. Since S_0, S_1, S_2 must be pairwise neighbors in screen space by lemma 2.1, and the angles of their pairwise boundaries are given by the direction-indication property, we have two possibilities:

- *Try to keep all $\overline{s_i s_j}$ parallel to the corresponding $\overline{w_i w_j}$.* The only angle preserving transformation of the player positions from world space to screen space is uniform scaling. Since w_2 is far away, the scaling factor must be very small, and therefore the distance between s_0 and s_1 becomes very small.
  The common boundary of S_0 and S_1 must be between s_0 and s_1 which are very close to each other, hence there is no placement for the boundary with s_0 and s_1 being centered in their respective polygons, see figure 3b.
- *Parallelity is not preserved from world space to screen space.* Let x, y be two players where the shared boundary between S_x and S_y is perpendicular to $\overline{w_x w_y}$ due to direction-indication, but not perpendicular to $\overline{s_x s_y}$. Let player x and y move towards each other on the line $\overline{w_x w_y}$, effectively not changing the angle of the boundary of S_x and S_y. At some point $w_x = w_y$, but $s_x \neq s_y$, see figure 3c. This motion is not continuously fusible.

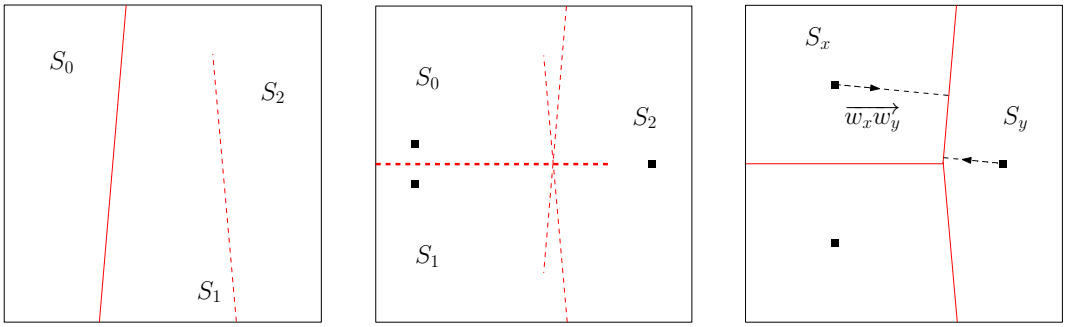
The only two possibilities are either not centered or not continuous and therefore contradict definition 1.4. ◀

▶ **Theorem 2.3.** *Fair Voronoi Split-screens are impossible for three or more players.*

Proof. For any number larger than three, we place the first three players as in the proof for lemma 2.2 and the others reasonably far to the right of the first three points. For the local situation of the first three players, the proof of the three player case applies respectively. ◀

3 A Quasi-Solution for Three Players

As established in the previous sections, a fair voronoi split-screen exists and is easy to implement for two players, while being impossible—and therefore obviously quite hard to implement—for three or more players.



(a) Placing S_1 not next to S_0 is impossible with given angles for boundary edges.

(b) The common boundary of S_0 and S_1 prohibits centeredness.

(c) Displacing centers leads to non-continuous fusion problems.

Figure 3 Illustrations for several parts of the main lemma's proof.

Since the proposed split-screen might appear in fast-paced games and the human eye is sluggish, some minor violations of the properties in definition 1.4 could be tolerable. The maybe most noticeable violation would be disruptions in the continuous movement and fusion, while a slight deviation in area sizes or centeredness could go unnoticed.

The presented algorithm 1 will utilize a relaxed centeredness condition to achieve fairness, fusibility and a smooth movement. It starts by computing the screen space regions. Collinear (or "almost collinear" for numeric stability) player positions are handled as a special case because their voronoi diagram has no voronoi vertex with finite coordinates. Otherwise the voronoi diagram is computed and moved around until all voronoi cells in the intersection with the screen space have almost equal area. Since we work with a finite amount of pixels in the end, a reasonable error threshold would be 0.01.

Computing the s_i involves the "cheating" and violates the centeredness condition for close-by player positions. The real center is computed and then slightly offset towards the centroid from definition 1.4 scaled by the distance to the other players.

Afterwards fusibility is checked for all regions. If all are fusible, all three players would act in the same window to the game world. Otherwise only two or none are fused. The OpenGL stencil buffer [1] or a similar tool can be used to limit the rendering to an arbitrarily shaped region of the screen space. Rendering the game world is always the same procedure, only translated individually for each region.

4 Conclusion & Room for Lots of Variations

Fairness and centeredness are both very important gameplay aspects. This is the first solution to provide both almost always, and its the first fair split-screen ever. The impossibility to achieve all naturally desired features is shown. The approximation is visually quite close to the (non-existing) optimum and barely interfering with fast-paced gameplay.

Quasi-solutions for more than three players remain a mystery. It seems, that fairness becomes much harder to achieve for more than three players. Maybe fairness and direction-indication together are impossible for a large enough number of players.

Many variations are possible: different kinds of "center", i.e. intentionally misplaced with more visibility in front than in the back, allow different sizes where stronger/faster units have a larger area, relaxed direction-indication condition by a few degrees. Let's discuss...

if w_0, w_1, w_2 *are collinear on line* L **then**
  $S_0, S_1, S_2 \leftarrow$ find two copies of $L^\perp$ dividing the screen space into three equally sized parts
else
  $VD \leftarrow$ compute voronoi diagram of w_0, w_1, w_2
  $vv \leftarrow$ sole voronoi vertex of VD
  translate vv (and the complete diagram with it) to $(0, 0)$
  repeat
    $S_0, S_1, S_2 \leftarrow$ cells of $V \cap$ screen space
    compute area sizes s_i of all S_i
    $\mathsf{error} \leftarrow \max_{i=0}^{2} s_i - \min_{i=0}^{2} s_i$
    translate vv by an amount scaled by error in the general direction of largest S_i
  until error *small enough*;
for $i \leftarrow 0$ **to** 2 **do**
  $\mathsf{center}_i \leftarrow$ compute center of S_i
for $i \leftarrow 0$ **to** 2 **do**
  $s_i \leftarrow$ is obtained by linear interpolating between center_i, and the midpoint of center_i and the center of its closest neighbor, weighted with their distance
foreach $(i, j) \in \{(0, 1), (0, 2), (1, 2)\}$ **do**
  if $(S_i - \mathsf{center}_i + w_i) \cap (S_j - \mathsf{center}_j + w_j) \neq \emptyset$ **then**
    mark S_i and S_j as fusible
if S_0, S_1, S_2 *all marked as fusible* **then**
  render world centered at $\frac{1}{3}(s_0 - w_0 + s_1 - w_1 + s_2 - w_2) - vv$
else if *only two regions marked as fusible:* S_a, S_b **then**
  set stencil mask to $S_{\text{non-fusible}}$
  render world centered at $s_{\text{non-fusible}} - w_{\text{non-fusible}} - vv$
  invert stencil mask
  render world centered at $\frac{1}{2}(s_a - w_a + s_b - w_b) - vv$
else
  for $i \leftarrow 0$ **to** 2 **do**
    set stencil mask to S_i
    render world centered at $s_i - w_i - vv$

Algorithm 1: Computing an almost fair voronoi split-screen for three players

References

1 Opengl stencil test. URL: https://www.khronos.org/opengl/wiki/Stencil_Test.

2 Gray Chang. Bumpomov's dogs. Atari 2600, 1983.

3 G. Lejeune Dirichlet. Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. *Journal für die reine und angewandte Mathematik*, 40:209–227, 1850.

4 Squirrel Eiserloh. Juicing your cameras with math. *Game Developers Conference (GDC)*, 2016. URL: http://www.mathforgameprogrammers.com/gdc2016/GDC2016_Eiserloh_Squirrel_JuicingYourCameras.pdf.

5 gorsman. Voronoi multiplayer split screen, 2016-05-11. URL: https://www.shadertoy.com/view/4sVXR1.

Short Plane Support Trees for Hypergraphs*

Thom Castermans[1], Mereke van Garderen[2], Wouter Meulemans[1], Martin Nöllenburg[3], and Xiaoru Yuan[4]

1 **Dept. of Mathematics and Computer Science, TU Eindhoven, the Netherlands**
[t.h.a.castermans|w.meulemans]@tue.nl
2 **Dept. of Computer and Information Sciences, Universität Konstanz, Germany**
mereke.van.garderen@uni-konstanz.de
3 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
noellenburg@ac.tuwien.ac.at
4 **Peking University, Beijing, China**
xiaoru.yuan@pku.edu.cn

Abstract

In many domains, the aggregation or classification of data elements leads to various intersecting sets. To allow for intuitive exploration and analysis of such data, set visualization aims to represent the elements and sets graphically. In more theoretical literature, such set systems are often referred to as hypergraphs. A support graph is a notion for drawing such a hypergraph, understood as a regular graph spanning the same vertices (elements), in which each hyperedge (set) induces a connected subgraph.

In this paper, we investigate finding a support graph of a hypergraph with fixed vertex locations under various constraints. We focus on enforcing planarity using a straight-line embedding, while minimizing the total length of the edges of the support graph, and consider the effect of the additional requirement that the support graph is acyclic.

1 Introduction

Intersecting sets are used in many domains to model various ways of clustering, grouping or aggregating measurements or data elements. To allow for effective exploration and analysis of such set systems, visualization is often used. Indeed, set visualization is an active subfield of information visualization; Alsallakh *et al.* [3] recently surveyed it. We focus on the case where elements have fixed positions in the plane, arising e.g. from geospatial locations.

On the theoretical side, such a set system is often referred to as a *hypergraph* $H = (V, S)$, with a set of vertices V (elements) and hyperedges S (sets), where each hyperedge $s \in S$ is some nonempty subset of V. A *support graph* of a hypergraph $H = (V, S)$ is a (regular) graph $G = (V, E)$ on the same vertex set such that every hyperedge $s \in S$ induces a connected subgraph in G [8]. In the remainder, we assume that V is a set of points in the plane and that a support graph is embedded using straight-line edges.

Though there are various ways of visualizing sets, support graphs match to a popular style in set visualization, namely that of connecting elements using colored links, such as seen for example in Kelp-style diagrams [9, 13] (see also Fig. 1) or LineSets [2]. Finding an embedded support graph that satisfies certain criteria therefore readily translates into a good rendering of the corresponding set system. A "good" support graph should avoid edge crossings, a standard quality criterion in the graph-drawing literature [14]. Moreover, as per Tufte's principle of ink minimization [15], it should have small total edge length.

* This work was started at Dagstuhl seminar 17332, Scalable Set Visualizations. T. Castermans is supported by the Netherlands Organisation for Scientific Research (NWO, 314.99.117). W. Meulemans is partially supported by the Netherlands eScience Centre (NLeSC, 027.015.G02).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

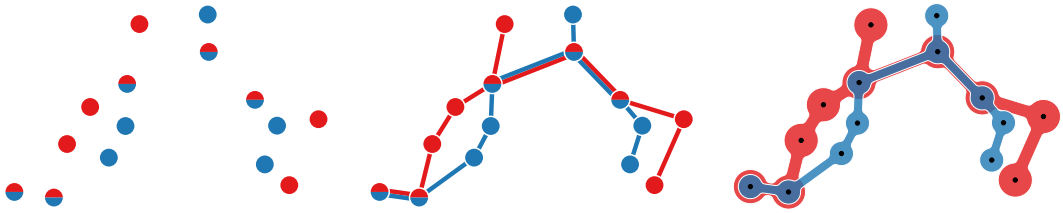


Figure 1 (a) A set system with colors indicating set membership. (b) The shortest plane support of the corresponding hypergraph. (c) A Kelp-style rendering of the set system.

Contributions In Section 2, we explore theoretical properties of requiring planarity. In particular, if there is at least one vertex that occurs in all hyperedges, then a plane support tree exists. However, we show that the minimum spanning tree on these vertices is not necessarily contained in any support tree or graph that is an approximation of the shortest one. An analogous statement holds for plane versus nonplane support trees. In Section 3, we turn to computational aspects and show that finding a plane support tree or graph with minimal total edge length is NP-hard, even for only two hyperedges that are disjoint or if one contains the other. In Section 4, we sketch an integer linear program to solve the problem.

Related Work Regarding support graphs for elements with fixed locations, some results are already known. The results of Bereg *et al.* [5] imply that existence of a plane support tree for two disjoint hyperedges can be decided in polynomial time; this readily implies the same result for a plane support graph. To the best of our knowledge, the problem is still open for $|S| > 2$; our result proves a sufficient condition but not a necessary one. This problem has also been studied in a Steiner setting [4], where additional points may be placed. Van Goethem *et al.* [16] enforce a stricter planarity than that of planar supports and investigate the resulting properties for elements on a regular grid, where only neighboring elements can be connected. However, length of the solution is of no concern in their results.

Without planarity, existence and length minimization of a (nonplane) support tree for fixed elements can be solved in polynomial time [11, 12]. However, acyclicity makes this problem easier. Indeed, length minimization of a (nonplane) support graph is NP-hard for three or more hyperedges [1]. In contrast, we show that this is in fact hard for two hyperedges if we require a plane support graph or tree. Without the planarity requirement, Hurtado *et al.* [10] show that length minimization for two hyperedges is solvable in polynomial time.

Planar support graphs without fixed elements have also received attention. For example, Buchin *et al.* [8] show that deciding whether a planar support graph exists is NP-hard in general; this proof readily works for elements with fixed locations, but note that it requires many hyperedges. In contrast, our result requires only two hyperedges, but uses length minimization. Brandes *et al.* [7] investigate support trees without fixed vertex locations, under the additional constraint that the induced subgraph of each hyperedge is Hamiltonian, and show that the existence of such a support can be checked in polynomial time.

2 Existential results

Before we study the computational problem of finding shortest plane support trees for a given hypergraph, we observe that a plane support tree always exists if there is at least one vertex that is contained in all hyperedges. To see why this is the case, consider the minimum spanning tree T of the nonempty set $A = \bigcap_{s \in S} s$, which is crossing-free. We can connect

each remaining point to its closest point in A. The resulting graph is obviously a tree since we add only leaves to T, it is a support as every hyperedge induces a connected subgraph, and it is plane as no edge crossings are created when connecting to the closest point in A.

▶ Observation 1. Consider a hypergraph $H = (V, S)$ with no three vertices in V on a line, such that $\bigcap_{s \in S} s \neq \emptyset$. H has a plane support tree.

We note that in a support tree the subgraph induced by A must be a connected subtree in order to satisfy the support property for all hyperedges. Next we show that using the above idea to start with a minimum spanning tree of A, the length of the resulting support tree cannot be bounded to be within a constant factor of the shortest plane support tree.

▶ **Lemma 2.1.** *There is a family of n-vertex hypergraphs $H = (V, S)$ with two hyperedges $S = \{r, b\}$ and $A = r \cap b \neq \emptyset$ such that any plane support tree of H that includes a minimum spanning tree of A is a factor $O(n)$ longer than the shortest plane support tree.*

Proof. The hypergraph family is illustrated in Fig. 2. The set $A = \{u, v, w\}$ consists of three vertices whose minimum spanning tree T has length $\ell + 1$ and is indicated by the black edges in Fig. 2(a). The remaining vertices in $V \setminus A$ are indicated in red and blue (indicating membership of r and b) and placed inside a disk of radius ε just left of the midpoint of edge uv. The vertices alternate in colors from left to right and form two mirrored convex chains.

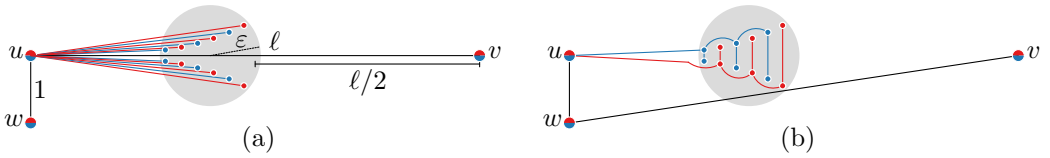


Figure 2 An n-point instance with approximation ratio $O(n)$ if using a minimum spanning tree on A. All edges are straight-line segments; curvature just emphasizes the effect of the convex chain.

Since edge uv of T splits the vertices in $V \setminus A$ and by their placement on convex chains, the shortest extension of T into a plane support tree is to connect every vertex to u (Fig. 2(a)). This yields a total length of the support tree of $O(n) \cdot \ell$. If, however, A is connected by a slightly longer tree, the remaining vertices in $V \setminus A$ can be joined by two comb-shaped structures as shown in Fig. 2(b). The resulting plane support tree has length of $O(1) \cdot \ell$. ◀

The above result also holds if we allow a general plane support graph. By removing the vertex w from the instance of Fig. 2 one can show in a similar fashion that a plane support tree, which now necessarily includes the edge uv, is a factor $O(n)$ longer than a shortest nonplane support tree; this corollary does not immediately generalize to support graphs.

▶ **Corollary 2.2.** *There is a family of n-vertex hypergraphs $H = (V, S)$ with two hyperedges $S = \{r, b\}$ and $A = r \cap b \neq \emptyset$ such that any plane support tree of H is a factor $O(n)$ longer than the shortest nonplane support tree.*

3 Computing a shortest plane support graph is NP-hard

Let us now turn towards the computational problem of finding the shortest plane support graph. Unfortunately, this problem and several restricted variants are NP-hard.

▶ **Theorem 3.1.** *Let $H = (V, S)$ be a hypergraph with vertices V having fixed locations in $\mathbb{R}^2$ and with S containing two hyperedges r and b such that $r \subseteq b$. It is NP-hard to decide whether H admits a plane support tree with length at most L for some $L > 0$.*

Proof. We use a reduction from (rectilinear) planar monotone 3-SAT [6]. Here, we are given a 3-CNF formula ϕ with n variables $v_1, \ldots, v_n$ and m clauses $c_1, \ldots, c_m$ such that every clause either has three positive literals or three negative literals. Moreover, we are given an embedding of ϕ as a plane graph, with rectangular vertices for variables on a horizontal line, and clauses as rectangles above or below the line (depending on whether the clause is positive or negative). Vertical edges connect clauses to the variables of their literals. We assume without loss of generality that the clauses are numbered according to their nesting: that is, $c_i < c_j$ if c_i is closer to the line of vertices than c_j in the embedding.

We must construct a hypergraph $H = (V, \{r, b\})$ such that $r \subseteq b$. In the remainder, we assign vertices to either r (red) or b (blue), understanding that any red vertex is also in b.

First, we place $3(n+1)$ red vertices using coordinates $(3i \cdot (m+1), y)$ for integers $i \in [0, n]$ and integers $y \in [-1, 1]$. Furthermore, we place $n \cdot (3m+2)$ blue vertices using coordinates $(3i(m+1) + j, 0)$ for integers $i \in [0, n-1]$ and $j \in [1, 3m+2]$.

We now place additional blue vertices for each clause c_a. We assume that this clause has positive literals for variable v_i, v_j, and v_k; the construction for clauses with negative literals is symmetric, using negative y-coordinates instead. First, we place $3a+1$ blue vertices from $(3(i-1)(m+1) + 3p, 2)$ to $(3(i-1)(m+1) + 3p, 2+3a)$ at unit distance, to represent the incidence from c_a to variable v_i, using the given embedding to determine that c_a is the pth clause incident from above to v_i. Analogously, we place the blue vertices for v_j and v_k. Now, we place further blue vertices at unit distance with y-coordinate $2+3a$ from the leftmost to the rightmost top vertex we just placed. The result is given in Fig. 3.

One clause requires at most $3(3m+1)$ vertices for the variable incidence and less than $3n \cdot (m+1)$ for the horizontal line connecting these. We can now readily measure the length of the minimum spanning tree on the blue vertices of one clause. We use L_a to denote this length; note that L_a is an integer at most $3(3m+1) + 3n \cdot (m+1)$.

The value of L that we select is $2(n+1) + 3n \cdot (m+1) + n(3m+2) + 2m + \sum_{a \in [1,m]} L_a$.

This finalizes the construction. It is polynomial since we placed $3(n+1)$ red vertices and $n \cdot (3m-2)$ blue vertices for the variables and at most $m \cdot (3(3m+1) + 3n \cdot (m+1))$ for the clauses: this is $O(nm^2)$ vertices. Moreover, we claim that our constructed hypergraph admits a plane support tree of length at most L, if and only if ϕ is satisfiable.

Assume we have a plane support tree of length at most L. First, we observe that all

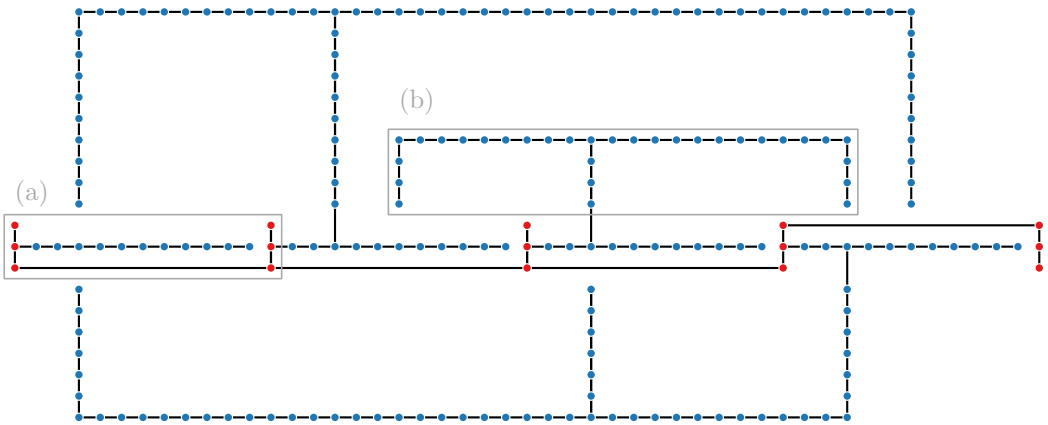


Figure 3 Construction for $\phi = (v_2 \vee v_3 \vee v_4) \wedge (\overline{v_1} \vee \overline{v_3} \vee \overline{v_4}) \wedge (v_1 \vee v_2 \vee v_4)$. Vertices in r and b are red, vertices in b are blue. A plane support tree with length at most L is given in black lines. (a) Representation of variable v_1; the solution sets v_1 to true. (b) Representation of the first clause.

points in r must be connected: the minimal way of doing so connects the three vertices with the same x-coordinate and uses one horizontal line to connect one triplet to the next. This has exactly length $2(n+1)+3n\cdot(m+1)$, corresponding to the first two terms defining L. The minimal way of connecting the lines inside the variables to the red tree takes length $n(3m+2)$ in total: this is the third term defining L. Finally, to connect the clause vertices, we need length at least L_a per clause, the last term of L. We note that any solution must use these constructions on the blue vertices, since all vertices are at unit distance; other blue vertices are at distance at least 2. However, the support tree is connected: thus it must still have connections from each gadget to either a red vertex or a blue vertex of a variable. The budget we have for this is $2m$ in total. Since each clause needs a connection of length at least 2, all clauses use exactly length 2. The only vertices within distance 2 of a clause are the three blue vertices of the variables with y-coordinate zero (one of each literal of the clause). Thus, each clause must have exactly one length-2 edge to one of these variable vertices. Since the support tree is plane, this cannot cross the horizontal links used to connect the red vertices. We can now readily obtain a satisfying assignment for ϕ, by looking at which of the two horizontal lines is used to connect the red vertices: if the one at the top is used, that variable is set to false; it is set to true otherwise.

To prove the converse, assume that we have a satisfying assignment. Using the same reasoning as above, we can construct the plane support tree by picking the connecting horizontal lines for the red vertices according to the satisfying assignment: this readily leads us to conclude that we can connect each clause using a length-2 connection that does not intersect the horizontal lines for the red vertices. ◀

We observe that the above proof readily implies that finding the shortest plane support graph is also NP-hard, as is the case that r is not a subset of b. Moreover, the proof can be easily adapted to show the other special case of disjoint r and b: this needs slightly more spacing such that we can add a few extra blue vertices that can be used to connect all the blue vertices of the variables into a single component using only length-1 edges.

4 Integer linear program

We showed in Section 3 that finding the shortest plane support is NP-hard, and so are several restricted versions of that problem. It is however possible to formulate these problems as *integer linear programs* (ILP), allowing us to leverage effective ILP solvers. Below, we briefly sketch how to obtain an ILP for a hypergraph $H=(V,S)$.

We introduce variables $e_{u,v}\in\{0,1\}$, indicating whether edge uv is selected for the support graph. This readily allows us to represent a graph with fixed vertices. Because the vertex locations are fixed, we can precompute edge lengths $d_{u,v}$ as well as which pairs of edges intersect. This gives the following basic program

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{u,v\in V} d_{u,v}\cdot e_{u,v} \\ \text{subject to} \quad & e_{u,v}+e_{w,x}\le 1 \qquad \text{for all } u,v,w,x\in V \text{ if edges } uv \text{ and } wx \text{ intersect.} \end{aligned}$$

What remains is to ensure that the graph is also a support: we need additional constraints that imply that each hyperedge in S induces a connected subgraph. To this end, we construct a flow tree for each hyperedge s. We pick an arbitrary sink for the hyperedge, $\sigma_s\in s$, that may receive flow, and let the remaining vertices in s generate one unit of flow. To formalize this, we introduce variables $f_{s,u,v}\in\{0,1,\dots,|s|-1\}$ for each $s\in S$ and $u,v\in s$ with $u\neq v$. We now need the following constraints: (a) the incoming flow at σ_s is exactly $|s|-1$; (b)

the outgoing flow at σ_s is zero; (c) except for σ_s, each vertex in s sends out one unit of flow more than it receives; (d) flow can be sent only over selected edges.

$$\begin{array}{lll} \text{(a)} & \sum_{u \in s \setminus \{\sigma_s\}} f_{s,u,\sigma_s} = |s| - 1 & \text{for all } s \in S \\ \text{(b)} & f_{s,\sigma_s,v} = 0 & \text{for all } s \in S, v \in s \setminus \{\sigma_s\} \\ \text{(c)} & \sum_{v \in s \setminus \{u\}} (f_{s,u,v} - f_{s,v,u}) = 1 & \text{for all } s \in S, u \in s \setminus \{\sigma_s\} \\ \text{(d)} & f_{s,u,v} \leq e_{u,v} \cdot (|s| - 1) & \text{for all } s \in S, u, v \in s \text{ with } u \neq v \end{array}$$

Variants The above ILP results in the shortest plane support graph for H. It can easily be modified to give a (shortest plane) support tree as well as to penalize or admit a limited number of intersections. The latter requires additional variables to indicate whether both edges of a crossing pair are used.

References

1 H. A. Akitaya, M. Löffler, and C. D. Tóth. Multi-colored spanning graphs. In *Graph Drawing and Network Visualization (GD'16)*, LNCS 9801, pp. 81–93. Springer, 2016.

2 B. Alper, N. Henry Riche, G. Ramos, and M. Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE TVCG*, 17(12):2259–2267, 2011.

3 B. Alsallakh, L. Micallef, W. Aigner, H. Hauser, S. Miksch, and P. Rodgers. The state of the art of set visualization. *CGF*, 35(1):234–260, 2016.

4 S. Bereg, K. Fleszar, P. Kindermann, S. Pupyrev, J. Spoerhase, and A. Wolff. Colored non-crossing Euclidean Steiner forest. In *Algorithms and Computation (ISAAC'15)*, LNCS 9472, pp. 429–441. Springer, 2015.

5 S. Bereg, M. Jiang, B. Yang, and B. Zhu. On the red/blue spanning tree problem. *TCS*, 412(23):2459–2467, 2011.

6 M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *Computing and Combinatorics (COCOON'10)*, LNCS 6196, pp. 216–225. Springer, 2010.

7 U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry. Path-based supports for hypergraphs. In *Combinatorial Algorithms (IWOCA'10)*, LNCS 6460, pp. 20–33. Springer, 2010.

8 K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. *JGAA*, 14(4):533–549, 2011.

9 K. Dinkla, M. van Kreveld, B. Speckmann, and M. Westenberg. Kelp Diagrams: Point set membership visualization. *CGF*, 31(3pt1):875–884, 2012.

10 F. Hurtado, M. Korman, M. van Kreveld, M. Löffler, V. Sacristán, A. Shioura, R. I. Silveira, B. Speckmann, and T. Tokuyama. Colored spanning graphs for set visualization. *CGTA*, 68:262–276, 2018.

11 B. Klemz, T. Mchedlidze, and M. Nöllenburg. Minimum tree supports for hypergraphs and low-concurrency Euler diagrams. In *Algorithm Theory (SWAT'14)*, LNCS 8503, pp. 253–264. Springer, 2014.

12 E. Korach and M. Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming, Series B*, 98:385–414, 2003.

13 W. Meulemans, N. Henry Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A hybrid set visualization technique. *IEEE TVCG*, 19(11):1846–1858, 2013.

14 H. Purchase. Metrics for graph drawing aesthetics. *J Visual Languages & Computing*, 13(5):501–516, 2002.

15 E. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001.

16 A. van Goethem, I. Kostitsyna, M. van Kreveld, W. Meulemans, M. Sondag, and J. Wulms. The painter's problem: covering a grid with colored connected polygons. In *Graph Drawing and Network Visualization (GD'17)*, 2017. `arXiv:1709.00001`.

Integer and Mixed Integer Tverberg Numbers

J.A. De Loera[1], T. Hogan[2], F. Meunier[3], and N. Mustafa[4]

1,2 Department of Mathematics, University of California, Davis
deloera@math.ucdavis.edu,tahogan@math.ucdavis.edu
3 CERMICS, Ecole Nationale des Ponts et Chaussées, Univ. Paris-Est, France
frederic.meunier@enpc.fr
4 Laboratoire d'Informatique Gaspard-Monge, Univ. Paris-Est, ESIEE Paris, Marne-la-Vallée, France
mustafan@esiee.fr

Abstract

We show the exact values of Tverberg numbers of $\mathbb{Z}^2$ and improve the bounds for $\mathbb{Z}^3$ and $\mathbb{Z}^j \times \mathbb{R}^k$.

1 Introduction

Consider n points in $\mathbb{R}^d$ and a positive integer $m \geq 2$. If $n \geq (m-1)(d+1)+1$, the points can always be partitioned into m subsets whose convex hulls contain a common point. This is the celebrated theorem of Tverberg [11], which has been the topic of many generalizations and variations since it was first proved in 1966. In this paper we formalize new versions of Tverberg's theorem where the coordinates of the points are integer. Our opening result closes a gap in the literature. It deals with a Tverberg-type theorem in the case of $\mathbb{Z}^2$. According to Eckhoff [6] it was stated by Doignon in a conference. Doignon (personal communication) confirmed that this was not published.

▶ **Theorem 1.** *Consider n points in $\mathbb{Z}^2$ and a positive integer $m \geq 3$. If $n \geq 4m-3$, then the points can be partitioned into m subsets whose convex hulls contain a common point in $\mathbb{Z}^2$.*

Such a partition is an *integer m-Tverberg partition* and such a common point is an *integer Tverberg point* for that partition. Regarding the case $m = 2$, the integer 2-Tverberg partitions are *integer Radon partitions*. Any configuration of at least 6 points admits an integer Radon partition. This was proved by Doignon in his PhD thesis [5] and later discovered independently by Onn [10]. All these values are optimal as shown by following examples. The 5-point configuration $\{(0,0),(0,1),(2,0)(1,2),(3,2)\}$, exhibited by Onn in the cited paper, has no Radon partition. To address the optimality when $m \geq 3$, consider the set $\{(i,i),(i,-i+1)\colon i = -m+2,-m+3,\ldots,m-2,m-1\}$. (According to Eckhoff [6], this set was proposed by Doignon during a conference.) It has $4m-4$ points and a moment of reflection might convince the reader that it has no integer m-Tverberg partition.

More generally, one can define the *Tverberg number* $\mathrm{Tv}(S,m)$ for a subset S of $\mathbb{R}^d$ and an integer $m \geq 2$ as the smallest integer number n such that any multiset of n points in S admits a partition into m subsets $A_1, A_2, \ldots, A_m$ with

$$\left(\bigcap_{i=1}^{m} \mathrm{conv}(A_i)\right) \cap S \neq \varnothing.$$

(Here, by "partition of a multiset", we mean that each element of a multiset A is contained in a number of subsets that does not exceed its multiplicity in A.) Theorem 1 together with the discussion that follows can then be rephrased as

$$\mathrm{Tv}(\mathbb{Z}^2,m) = \begin{cases} 6 & \text{if } m = 2, \\ 4m-3 & \text{otherwise.} \end{cases}$$

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Our second main result improves the upper bound for the case $S = \mathbb{Z}^3$.

▶ **Theorem 2.** *The following inequality holds for all $m \geq 2$:*

$$\mathrm{Tv}(\mathbb{Z}^3, m) \leq 24m - 31.$$

Proofs of Theorems 1 and 2 are respectively given in Sections 2 and 3. The strategy of both proofs is standard: we show that there exists an integer centerpoint (which we define at the end of this section) of sufficient depth and that this centerpoint is actually a *Tverberg point* of an m-Tverberg partition.

Choosing S of the form $\mathbb{Z}^j \times \mathbb{R}^k$ leads to the "mixed integer" case, which is the common generalization of the real and the integer cases. Our third main result is an inequality simultaneously involving the three already considered instantiations of S: real, integer, and mixed integer.

▶ **Theorem 3.** *The following inequality holds for all positive integers j and k and all $m \geq 2$:*

$$\mathrm{Tv}(\mathbb{Z}^j \times \mathbb{R}^k, m) \leq \mathrm{Tv}(\mathbb{Z}^j, \mathrm{Tv}(\mathbb{R}^k, m)).$$

Finally, in Section 4, we prove Theorem 3 and collect some consequences of the main theorems presented above, including the following result:

$$2^j(m-1)(k+1) + 1 \leq \mathrm{Tv}(\mathbb{Z}^j \times \mathbb{R}^k, m) \leq j2^j(m-1)(k+1) + 1. \tag{1}$$

To conclude the introduction we mention a key lemma about integral centerpoints that is used for proving Theorems 1 and 2. Given a multiset A of points, a point $\boldsymbol{p}$ is a *centerpoint of depth σ* in A if every closed half-space containing $\boldsymbol{p}$ contains at least σ points of A.

▶ **Lemma 4.** *Consider a multiset A of points in $\mathbb{Z}^d$. If $|A| \geq 2^d(m-1) + 1$ (counting multiplicities), then there is a centerpoint $\boldsymbol{p} \in \mathbb{Z}^d$ of depth m in A.*

Although the present version is new, similar lemmas have been used throughout the literature and their proofs typically rely on some version of Helly's theorem [7]. We omit the classical details here, and simply mention that we need the following theorem of Doignon [4]: If $\mathcal{F}$ is a finite family of at least 2^d convex subsets of $\mathbb{Z}^d$ such that any 2^d members of $\mathcal{F}$ have an intersection point in $\mathbb{Z}^d$, there is a point $\boldsymbol{p} \in \mathbb{Z}^d$ in every set in $\mathcal{F}$.

In Sections 2 and 3 when we refer to Tverberg partitions or Tverberg points we focus on *integer* Tverberg partitions.

Related Results from the Literature

The problem of computing the Tverbeg number for $\mathbb{Z}^d$ with $d \geq 3$ seems to be challenging. It has been identified as an interesting problem since the 1970's and yet the following inequalities are almost all that is known about this problem: For the general case, De Loera et al. [8] proved

$$2^d(m-1) + 1 \leq \mathrm{Tv}(\mathbb{Z}^d, m) \leq d2^d(m-1) + 1 \quad \text{for } d \geq 1 \text{ and } m \geq 2. \tag{2}$$

Two special cases get better bounds:

$$\mathrm{Tv}(\mathbb{Z}^3, 2) \leq 17 \qquad \text{and} \qquad 5 \cdot 2^{d-2} + 1 \leq \mathrm{Tv}(\mathbb{Z}^d, 2) \quad \text{for } d \geq 2. \tag{3}$$

The left-hand side inequality is due to Bezdek and Blokhuis [2] and the right-hand side was proved by Doignon in his PhD thesis (and rediscovered by Onn).

The bounds for the "mixed integer" case include the bounds for the Radon number (2-Tverberg number) found by Averkov and Weismantel [1].

$$2^j(k+1)+1 \leq \mathrm{Tv}(\mathbb{Z}^j \times \mathbb{R}^k, 2) \leq (j+k)2^j(k+1) - j - k + 2.$$

Later, De Loera et al. [8] gave the following general bound for all Tverberg numbers:

$$\mathrm{Tv}(\mathbb{Z}^j \times \mathbb{R}^k, m) \leq (j+k)2^j(m-1)(k+1) + 1.$$

Note that (1) above is a simultaneous improvement of both of these.

2 Tverberg Numbers over $\mathbb{Z}^2$: Proof of Theorem 1

The theorem will follow easily from the following two lemmas, the first covering the case $m \geq 3$ and the second the case $m = 2$.

▶ **Lemma 5.** *Consider a multiset A of points in $\mathbb{Z}^2$ with $|A| \geq 4m-3$ and $m \geq 3$. If $\boldsymbol{p} \notin A$ is a centerpoint of depth m, then there is an m-Tverberg partition with $\boldsymbol{p}$ as Tverberg point.*

▶ **Lemma 6.** *Consider a multiset A of points in $\mathbb{Z}^2$ with $|A| \geq 6$. If $\boldsymbol{p} \notin A$ is a centerpoint of depth two, then there is a Radon partition with $\boldsymbol{p}$ as Tverberg point.*

Proof of Theorem 1. Consider a multiset A of at least $4m-3$ points in $\mathbb{Z}^2$. By Lemma 4, A has an integer centerpoint $\boldsymbol{p}$ of depth m. If $\boldsymbol{p}$ is an element of A with multiplicity $\mu \geq 0$, then take the singletons $\{\boldsymbol{p}\}$ as μ of the sets in the Tverberg partition. Then $\boldsymbol{p}$ is a centerpoint of depth $m-\mu$ of the remaining $4m-\mu-3$ points. If $\mu \geq m$, we are done, and if $\mu = m-1$, the point $\boldsymbol{p}$ is in the convex hull of the remaining points and we take them to be the last set in the desired partition. If $\mu \leq m-3$, according to Lemma 5, there is an $(m-\mu)$-Tverberg partition of the remaining points with $\boldsymbol{p}$ as Tverberg point. There is thus an m-Tverberg partition of A with $\boldsymbol{p}$ as Tverberg point. The case $\mu = m-2$ is treated similarly with the help of Lemma 6 in place of Lemma 5 ◀

Proof of Lemma 5. Since $\boldsymbol{p}$ is not in A, up to a radial projection, we can assume that the points of A are arranged in a circle around $\boldsymbol{p}$. Define q and r to be respectively the quotient and the remainder of the Euclidean division of $|A|$ by m. Define moreover e to be $\lceil \frac{r}{q} \rceil$.

Suppose first that $\boldsymbol{p}$ is a centerpoint of depth $m+e$. In such a case, we arbitrarily select a first point in A, and label clockwise the points with elements in $[m]$ according to the following pattern:

$$1, 2, \ldots, m, 1, 2, \ldots, e, 1, 2, \ldots, m, 1, 2, \ldots, e, 1, 2, \ldots m, 1, 2, \ldots k,$$

where $k = |A| - qm - (q-1)e$. Note that we have $k \leq e$. Each half-plane delimited by a line passing through $\boldsymbol{p}$ contains at least $m+e$ consecutive points in this pattern and thus has at least one point with each of the m different labels. Partitioning the points so that each subset consists of all points with a fixed label, we therefore obtain an m-Tverberg partition with $\boldsymbol{p}$ as Tverberg point.

Suppose now that $\boldsymbol{p}$ is not a centerpoint of depth $m+e$. There is thus a closed half-plane H_+ delimited by a line passing through $\boldsymbol{p}$ with $|H_+ \cap A| < m+e$. The complementary closed half-plane to H_+, which we denote by H_-, is such that $|H_- \cap A| > 4m-3-(m+e)$. Define ℓ to be $|H_- \cap A|$. Since $e \leq \frac{m}{3}$, we have $\ell \geq 2m$. Denote the points in $H_- \cap A$ by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$, where the indices are increasing when we move clockwise. We label $\boldsymbol{x}_i$ with $r+i$ from $\boldsymbol{x}_1$ to $\boldsymbol{x}_{m-r}$, and then label $\boldsymbol{x}_{m-r+j}$ with j from $\boldsymbol{x}_{m-r+1}$ to $\boldsymbol{x}_m$. We then continue labeling

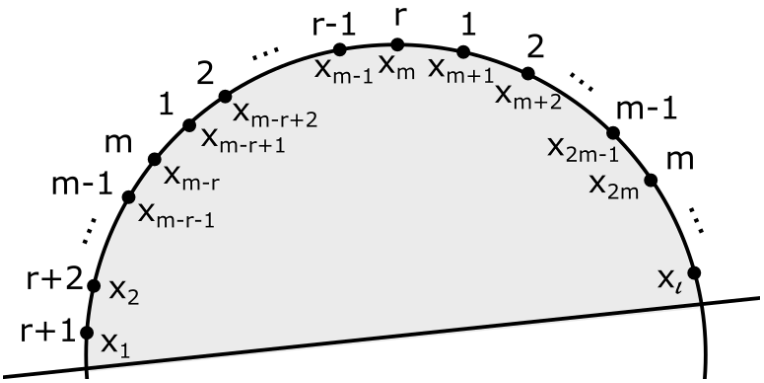


Figure 1 Labeling of the points in the half-plane H_-.

the points of A, still moving clockwise, using labels $1, 2, \ldots, m, \ldots, 1, 2, \ldots m, 1, 2, \ldots r$. See Figure 1 for an illustration of the labeling scheme.

The labeling pattern is such that any sequence of m consecutive points either has all m labels, or contains the two consecutive points $\boldsymbol{x}_m$ and $\boldsymbol{x}_{m+1}$. Let us prove that any closed half-plane H delimited by a line passing through $\boldsymbol{p}$ contains at least one point with each label. Once this is proved, the conclusion will be immediate by taking as subsets of points those with same labels, as above.

If such an H does not simultaneously contain $\boldsymbol{x}_m$ and $\boldsymbol{x}_{m+1}$, then H contains at least one point with each label. Consider thus a closed half-plane H delimited by a line passing through $\boldsymbol{p}$ and containing $\boldsymbol{x}_m$ and $\boldsymbol{x}_{m+1}$. Note that according to Farkas' lemma, $\boldsymbol{x}_{m+1}$ cannot be separated from $\boldsymbol{x}_1$ and $\boldsymbol{x}_\ell$ by a line passing through $\boldsymbol{p}$, since they are all in H_-. This means that either H contains $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{m+1}$, or H contains $\boldsymbol{x}_{m+1}, \boldsymbol{x}_{m+2}, \ldots, \boldsymbol{x}_\ell$. In any case, H contains a point with each label. ◀

The proof of Lemma 6 is similar and left to the reader for brevity.

3 Tverberg Numbers over $\mathbb{Z}^3$: Proof of Theorem 2

We will make use of the following two lemmas. Lemma 7 is a consequence, upon close inspection of the argument, of the proof of the main theorem in the already mentioned paper by Bezdek and Blokhuis [2].

▶ **Lemma 7.** *Consider a multiset A of at least 17 points in $\mathbb{R}^3$ and a centerpoint $\boldsymbol{p}$ of depth 3 in A. There is a bipartition of A into two subsets whose convex hulls contain $\boldsymbol{p}$.*

▶ **Lemma 8.** *Consider a multiset A of points in $\mathbb{R}^3$ with $|A| \geq 24m - 31$ and $m \geq 2$. If $\boldsymbol{p} \notin A$ is a centerpoint of depth $3m - 3$, then there is an m-Tverberg partition of A with $\boldsymbol{p}$ as Tverberg point.*

Proof. Since $\boldsymbol{p}$ is not an element of A, we assume without loss of generality that the points of A are located on a sphere centered at $\boldsymbol{p}$, as in the proof of Lemma 5.

We claim that we can find pairwise disjoint subsets $X_1, X_2, \ldots, X_{m-2}$ of A, each having $\boldsymbol{p}$ in its convex hull and each being of cardinality at most four. (Here "pairwise disjoint" means that each element of A is present in a number of X_i's that does not exceed its multiplicity in A.) We proceed by contradiction. Suppose that we can find at most $s < m - 2$ such subsets X_i's. Then, by Carathéodory's theorem [3], $\boldsymbol{p}$ is not in the convex hull of the remaining

points in A.Therefore there is a half-space H_+ delimited by a plane containing $\boldsymbol{p}$ such that $H_+ \cap A \subseteq \bigcup_{i=1}^{s} X_i$. On the other hand, since each X_i contains $\boldsymbol{p}$ in its convex hull (and we can assume the X_i are minimal with respect to containing $\boldsymbol{p}$), we have $|H_+ \cap X_i| \leq 3$ for all $i \in [s]$. Therefore $|H_+ \cap A| \leq |H_+ \cap (\bigcup_{i=1}^{s} X_i)| \leq 3s < 3(m-2)$, which is a contradiction since $\boldsymbol{p}$ is a centerpoint of depth $3m-3$ in A. There are thus $m-2$ disjoint subsets $X_1, X_2, \ldots, X_{m-2}$ as claimed.

Let X denote $\bigcup_{i=1}^{m-2} X_i$. Consider an arbitrary half-space H_+ delimited by a plane containing $\boldsymbol{p}$. Since $|H_+ \cap X_i| \leq 3$ for all i, we have $|H_+ \cap X| \leq 3(m-2)$. Furthermore $|H_+ \cap A| \geq 3m-3$, so $|H_+ \cap (A \setminus X)| \geq 3$. Since H_+ is arbitrary, $\boldsymbol{p}$ is a centerpoint of depth 3 of $A \setminus X$. Also, $|A \setminus X| \geq |A| - 4(m-2) \geq 20m - 23 \geq 17$, so Lemma 7 implies that $A \setminus X$ can be partitioned into two sets whose convex hulls contain $\boldsymbol{p}$. With the subsets X_i, we have therefore an m-Tverberg partition of A, with $\boldsymbol{p}$ as Tverberg point. ◀

From these two lemmas we can now finish the proof of Theorem 2.

Proof of Theorem 2. Consider a multiset A of $24m-31$ points in $\mathbb{Z}^3$. The case $m=2$ is the already mentioned result by Bezdek and Blokhuis. Assume that $m \geq 3$. Applying Lemma 4, A has an integer centerpoint $\boldsymbol{p}$ of depth $3m-3$. If $\boldsymbol{p}$ is an element of A with multiplicity $\mu \geq 0$, then take the singletons $\{\boldsymbol{p}\}$ as μ of the sets in the Tverberg partition.

If $\mu \geq m$, we are done. If $\mu = m-1$, the point $\boldsymbol{p}$ is still in the convex hull of points in A, and thus we are done. And if $\mu \leq m-2$, the point $\boldsymbol{p}$ is still a centerpoint of depth $3m-\mu-3 \geq 3(m-\mu)-3$ of the remaining $24m-\mu-31 \geq 24(m-\mu)-31$ points. Thus, we may apply Lemma 8 to get an $(m-\mu)$-Tverberg partition of the remaining points, with $\boldsymbol{p}$ as Tverberg point, and conclude the result. ◀

4 Tverberg Numbers over $\mathbb{Z}^j \times \mathbb{R}^k$

In this section, we prove Theorem 3. We adapt an approach by Mulzer and Werner [9, Lemma 2.3] and show how all the results of our paper can be combined to improve known bounds and to determine new exact values for the Tverberg number in the mixed integer case.

Proof of Theorem 3. Let $t = \mathrm{Tv}(\mathbb{R}^k, m) = (m-1)(k+1)+1$. Choose a multiset A in $\mathbb{Z}^j \times \mathbb{R}^k$ with $|A| \geq \mathrm{Tv}(\mathbb{Z}^j, t)$. It suffices to prove that A can be partitioned into m subsets whose convex hulls contain a common point in $\mathbb{Z}^j \times \mathbb{R}^k$.

Let A' be the projection of A onto $\mathbb{Z}^j$. Since $|A'| \geq \mathrm{Tv}(\mathbb{Z}^j, t)$, there is a partition of A' into t subsets $Q'_1, \ldots, Q'_t$ whose convex hulls contain a common point $\boldsymbol{q}$ in $\mathbb{Z}^j$. The Q'_i are the projections onto $\mathbb{Z}^j$ of t disjoint subsets Q_i forming a partition of A. For each $i \in [t]$, we can find a point $\boldsymbol{q}_i \in \mathrm{conv}(Q_i)$ projecting onto $\boldsymbol{q}$.

The t points $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_t$ belong to $\{\boldsymbol{q}\} \times \mathbb{R}^k$. As $t = \mathrm{Tv}(\mathbb{R}^k, m)$, there exists a partition of $[t]$ into $I_1, \ldots, I_m$ and a point $\boldsymbol{p} \in \{\boldsymbol{q}\} \times \mathbb{R}^k$ such that $\boldsymbol{p} \in \mathrm{conv}\left(\bigcup_{i \in I_\ell} \boldsymbol{q}_i\right)$ for all $\ell \in [m]$. For each $\ell \in [m]$, define A_ℓ to be $\bigcup_{i \in I_\ell} Q_i$. We have for each $\ell \in [m]$

$$\boldsymbol{p} \in \mathrm{conv}\left(\bigcup_{i \in I_\ell} \boldsymbol{q}_i\right) \subseteq \mathrm{conv}\left(\bigcup_{i \in I_\ell} \mathrm{conv}(Q_i)\right) = \mathrm{conv}(A_\ell)$$

and the A_ℓ form the desired partition. ◀

Here are the new bounds and exact values we get:

(a) $\mathrm{Tv}(\mathbb{Z} \times \mathbb{R}^k, m) = 2(m-1)(k+1)+1$.

(b) $\mathrm{Tv}(\mathbb{Z}^2 \times \mathbb{R}^k, m) = 4(m-1)(k+1)+1$.
(c) $\mathrm{Tv}(\mathbb{Z}^3 \times \mathbb{R}^k, m) \leq 24(m-1)(k+1)-7$.
(d) $2^j(m-1)(k+1)+1 \leq \mathrm{Tv}(\mathbb{Z}^j \times \mathbb{R}^k, m) \leq j2^j(m-1)(k+1)+1$.

The lower bound in (d) is obtained by repeated applications of Lemma 9 below, whose proof, almost identical to that of Proposition 2.1 in [10], is omitted for brevity. The upper bounds follow from Theorem 3, combined with the fact that $\mathrm{Tv}(\mathbb{Z}, m) = 2m-1$ (consider the median), Theorem 1, Theorem 2, and the upper bound in Equation (2), respectively.

▶ **Lemma 9.** *Let j and k be two non-negative integers. Then we have*

$$\mathrm{Tv}(\mathbb{Z}^{j+1} \times \mathbb{R}^k, m) > 2\,\mathrm{Tv}(\mathbb{Z}^j \times \mathbb{R}^k, m) - 2.$$

5 Acknowledgments

This work was partially supported by NSF grant DMS-1440140, while the first and third authors were in residence at the Mathematical Sciences Research Institute in Berkeley, California, during the Fall 2017 semester. The first and second author were also partially supported by NSF grant DMS-1522158. The fourth author was supported by ANR grant SAGA (ANR-14-CE25-0016).

References

1 G. Averkov and R. Weismantel. Transversal numbers over subsets of linear spaces. *Adv. Geom.*, 12(1):19–28, 2012. URL: http://dx.doi.org/10.1515/advgeom.2011.028.
2 K. Bezdek and A. Blokhuis. The Radon number of the three-dimensional integer lattice. *Discrete & Computational Geometry*, 30(2):181–184, 2003. URL: http://dx.doi.org/10.1007/s00454-003-0003-8, doi:10.1007/s00454-003-0003-8.
3 C. Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, Mar 1907. URL: https://doi.org/10.1007/BF01449883, doi:10.1007/BF01449883.
4 J.-P. Doignon. Convexity in crystallographical lattices. *J. Geom,*, 3(1):71–85, March 1973.
5 J.-P. Doignon. *Segments et ensembles convexes.* PhD thesis, Universite Libre de Bruxelles, 1975.
6 J. Eckhoff. The partition conjecture. *Discrete Math.*, 221(1-3):61–78, 2000.
7 E. Helly. *Über Mengen konvexer Körper mit gemeinschaftlichen Punkte.* Jahresbericht der Deutschen Mathematiker- . . . , 1923.
8 J.A. De Loera, R. La Haye, D. Rolnick, and P. Soberón. Quantitative Tverberg theorems over lattices and other discrete sets. *Discrete & Computational Geometry*, pages 1–14, 2017. URL: http://dx.doi.org/10.1007/s00454-016-9858-3, doi:10.1007/s00454-016-9858-3.
9 W. Mulzer and D. Werner. Approximating Tverberg points in linear time for any fixed dimension. *Discrete & Computational Geometry*, 50(2):520–535, 2013.
10 S. Onn. On the Geometry and Computational Complexity of Radon Partitions in the Integer Lattice. *SIAM J. Discrete Math.*, 1991.
11 H. Tverberg. A generalization of Radon's theorem. *J. London Math. Soc.*, 41(1):123–128, 1966.

Deletion in abstract Voronoi diagrams in expected linear time*

Kolja Junginger[1] and Evanthia Papadopoulou[1]

1 **Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland**
kolja.junginger@usi.ch, evanthia.papadopoulou@usi.ch

Abstract

Updating an abstract Voronoi diagram in linear time, after deletion of one site, has been an open problem for a long time. Similarly for concrete Voronoi diagrams of generalized sites, other than points. In this abstract we present a simple, expected linear-time algorithm for this task. We introduce the concept of a *Voronoi-like diagram*, a relaxed version of a Voronoi construct, that has a structure similar to an abstract Voronoi diagram without however being one. Voronoi-like diagrams serve as intermediate structures, which are considerably simpler to compute, thus, making an expected linear-time construction possible.

1 Introduction

The Voronoi diagram of a set S of n simple geometric objects, called sites, is a well-known geometric partitioning structure revealing proximity information for the input sites. *Abstract Voronoi diagrams* [7] offer a unifying framework to various concrete instances. Some classic Voronoi diagrams have been well investigated and optimal construction algorithms exist in many cases, see [2] for references and more information.

For certain *tree-like* Voronoi diagrams linear-time construction algorithms are well known to exist, e.g., [1, 4, 9, 5]. The first technique was introduced by Aggarwal et al. [1] for the Voronoi diagram of points in convex position, given their convex hull. It can be used to derive linear-time construction algorithms for other fundamental problems such as updating a Voronoi diagram of point-sites in linear time, after deleting one site. A much simpler randomized approach for the same problem has been introduced by Chew [4]. Klein and Lingas [9] adapted the linear-time framework of [1] to abstract Voronoi diagrams under restrictions, and showed that a *Hamiltonian abstract Voronoi diagram* can be computed in linear time, given the order of Voronoi regions along an unbounded simple curve, which visits each region *exactly once* and can intersect each bisector only *once*. This construction has been extended recently to include forest structures [3] under similar conditions, where no region can have multiple faces within the domain enclosed by a curve. The medial axis of a simple polygon is another well-known problem to admit a linear-time construction [5].

In this abstract we consider the problem of updating an abstract Voronoi diagram after deletion of one site and provide an expected linear-time algorithm to achieve this task. To the best of our knowledge, no linear-time construction algorithms are known for concrete diagrams of non-point sites, nor for abstract Voronoi diagrams. Related is our expected linear-time algorithm for the farthest-segment Voronoi diagram [6]. The approach in [6], however, is geometric, relying on star-shapeness and visibility properties of segment Voronoi regions that do not extend to the abstract model. In this abstract we provide a new formulation.

Abstract Voronoi diagrams (AVDs). AVDs were introduced by Klein [7]. Instead of sites and distance measures, they are defined in terms of bisecting curves that satisfy some

* Supported in part by the Swiss National Science Foundation under DACH project SNF 200021E-154387.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

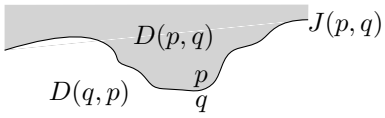


Figure 1 A bisector $J(p,q)$ and its dominance regions; $D(p,q)$ is shown shaded.

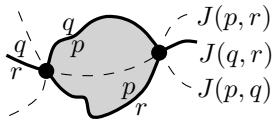


Figure 2 The Voronoi diagram $\mathcal{V}(\{p,q,r\})$ in solid lines. The shaded region is $\mathrm{VR}(p,\{p,q,r\})$.

simple combinatorial properties. Given a set S of n abstract sites, the bisector $J(p,q)$ of two sites $p,q \in S$ is an unbounded curve, homeomorphic to a line, that divides the plane into two open domains: the *dominance region of* p, $D(p,q)$ (with *label* p), and the *dominance region of* q, $D(q,p)$ (with *label* q), see Fig. 1. The Voronoi region of p is: $\mathrm{VR}(p,S) = \bigcap_{q\in S\setminus\{p\}} D(p,q)$ and the *(nearest-neighbor) abstract Voronoi diagram* of S is $\mathcal{V}(S) = \mathbb{R}^2 \setminus \bigcup_{p\in S} \mathrm{VR}(p,S)$, see Fig. 2. Following the traditional model of abstract Voronoi diagrams [7] an *admissible* system of bisectors $\mathcal{J}$ is assumed to satisfy the following axioms, for every subset $S' \subseteq S$:

(A1) Each nearest Voronoi region $\mathrm{VR}(p,S')$ is non-empty and pathwise connected.
(A2) Each point in the plane belongs to the closure of a nearest Voronoi region $\mathrm{VR}(p,S')$.
(A3) After stereographic projection to the sphere, each bisector is a Jordan curve through the north pole.
(A4) Any two bisectors $J(p,q)$ and $J(r,t)$ intersect transversally and in a finite number of points. (It is possible to relax this axiom, see [8]).

$\mathcal{V}(S)$ is a plane graph of structural complexity $O(n)$ and its regions are simply-connected. It can be computed in time $O(n\log n)$, randomized or deterministic, see [2]. To update $\mathcal{V}(S)$, after deleting one site $s \in S$, we compute $\mathcal{V}(S\setminus\{s\})$ within $\mathrm{VR}(s,S)$. The sequence of sites along $\partial\mathrm{VR}(s,S)$ forms a Davenport-Schinzel sequence (DSS) of order 2 and this constitutes a major difference from the respective problem for points where no repetition can occur.

Our results. We give a simple randomized algorithm to compute $\mathcal{V}(S\setminus\{s\})$ within $\mathrm{VR}(s,S)$ in expected time linear on the complexity of $\partial\mathrm{VR}(s,S)$. The algorithm is simple, not more complicated than its counterpart for points [4], and this is achieved by computing simplified intermediate structures. These are *Voronoi-like* diagrams, having a structure similar to an abstract Voronoi diagram, without however being such diagrams. We prove that Voronoi-like diagrams are well-defined and robust under an *insertion operation*, thus, making possible a randomized incremental construction for $\mathcal{V}(S\setminus\{s\}) \cap \mathrm{VR}(s)$ in expected linear time. Our approach can be adapted (in fact, simplified) to compute, in expected linear time, the farthest abstract Voronoi diagram after the sequence of its faces at infinity is known; the latter can be computed in time $O(n\log n)$. Our technique can be applied to concrete diagrams that may not strictly fall under the AVD model such as Voronoi diagrams of line segments that may intersect and of planar straight-line graphs (including simple and non-simple polygons). For intersecting line segments, $\partial\mathrm{VR}(s,S)$ is a Davenport-Schinzel sequence of order 4 [10].

2 Problem formulation

Let S be a set of n abstract *sites* that define an admissible system of bisectors $\mathcal{J} = \{J(p,q) : p \neq q \in S\}$. Bisectors that have a site p in common are called *p-related*. Two related bisectors $J(p,q)$ and $J(p,r)$ can intersect at most twice; bisector $J(q,r)$ also intersects with them at the same point(s) [7]. Since related bisectors in $\mathcal{J}$ intersect at most twice, the sequence of site occurrences along $\partial\mathrm{VR}(p,S)$, $p \in S$, forms a DSS of order 2 (by [11, Theorem 5.7]).

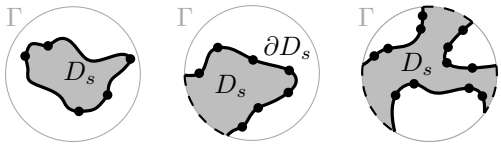


Figure 3 The domain $D_s = \mathrm{VR}(s, S) \cap D_\Gamma$.

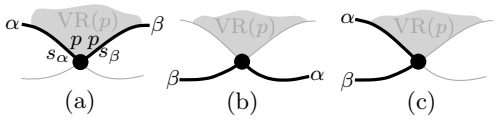


Figure 4 (a) Arcs α, β fulfill the p-monotone path condition; they do not fulfill it (b) and (c).

To update $\mathcal{V}(S)$ after deleting one site $s \in S$, we compute $\mathcal{V}(S \setminus \{s\})$ within $\mathrm{VR}(s, S)$, i.e., we compute $\mathcal{V}(S \setminus \{s\}) \cap \mathrm{VR}(s, S)$; its structure is given in the following lemma.

▶ **Lemma 1.** *$\mathcal{V}(S \setminus \{s\}) \cap VR(s, S)$ is a forest having exactly one face for each Voronoi edge of $\partial VR(s, S)$. Its leaves are the Voronoi vertices of $\partial VR(s, S)$, and points at infinity if $VR(s, S)$ is unbounded. If $VR(s, S)$ is bounded then $\mathcal{V}(S \setminus \{s\}) \cap VR(s, S)$ is a tree.*

We make a general position assumption that no three p-related bisectors intersect at the same point. This implies that Voronoi vertices have degree 3. We consider a closed Jordan curve Γ large enough to enclose all intersections of bisectors in $\mathcal{J}$, and such that each bisector crosses Γ exactly twice and transversally; the interior of Γ is denoted D_Γ. Our *domain* of computation is $D_s = \mathrm{VR}(s, S) \cap D_\Gamma$ and we compute $\mathcal{V}(S \setminus \{s\}) \cap D_s$, see Figure 3.

Let $\mathcal{S}$ denote the sequence of Voronoi edges along $\partial\mathrm{VR}(s, S)$, i.e., $\mathcal{S} = \partial\mathrm{VR}(s, S) \cap D_\Gamma$. Each arc $\alpha \in \mathcal{S}$ is induced by a site $s_\alpha \in S \setminus \{s\}$, where $\alpha \subseteq J(s, s_\alpha)$. We can interpret the arcs in $\mathcal{S}$ as sites that induce a Voronoi diagram $\mathcal{V}(\mathcal{S})$, where $\mathcal{V}(\mathcal{S}) = \mathcal{V}(S \setminus \{s\}) \cap D_s$, see Figure 7(a). In this respect, each arc $\alpha \in \mathcal{S}$ has a Voronoi region, $\mathrm{VR}(\alpha, \mathcal{S})$, which is the face of $\mathcal{V}(S \setminus \{s\}) \cap D_s$ incident to α.

In the remaining of this section we define a *Voronoi-like diagram* for a subset $\mathcal{S}'$ of arcs in $\mathcal{S}$ (Def. 2). To this aim we need some definitions. For a site $p \in S$ and $S' \subseteq S$, let $\mathcal{J}_{p,S'} = \{J(p, q) \,|\, q \in S', q \neq p\}$ denote the set of all *p-related* bisectors involving sites in S'.

A *path* P in a bisector system $\mathcal{J}_{p,S'}$ is a connected subset of alternating edges and vertices in the arrangement of $\mathcal{J}_{p,S'}$. An *arc* α of P is a maximal connected set, along P, of consecutive edges and vertices of the arrangement, which belong to the same bisector. The common endpoint of two consecutive arcs of P is a *vertex of* P; an arc of P is also called an *edge*. For an arc $\alpha \in P$, let $s_\alpha \in S$ be the site that *induces* α, i.e., $\alpha \subseteq J(p, s_\alpha)$.

A path P in $\mathcal{J}_{p,S'}$ is called *p-monotone*, if any two consecutive arcs $\alpha, \beta \in P$ correspond to the Voronoi edges in $\partial\mathrm{VR}(p, \{p, s_\alpha, s_\beta\})$ that are incident to the common endpoint of α, β (see Fig. 4). The *envelope of* $\mathcal{J}_{p,S'}$, with respect to site p, is the boundary of the Voronoi region $\mathrm{VR}(p, S' \cup \{p\})$, $\mathrm{env}(\mathcal{J}_{p,S'}) = \partial\mathrm{VR}(p, S' \cup \{p\})$. Fig. 5 illustrates two p-monotone paths, where (a) is an envelope. Notice, $\mathcal{S} = \mathrm{env}(\mathcal{J}_{s,S\setminus\{s\}}) \cap D_\Gamma$.

Consider $\mathcal{S}' \subseteq \mathcal{S}$ and let $S' = \{s_\alpha \in S \,|\, \alpha \in \mathcal{S}'\} \subseteq S \setminus \{s\}$ be its corresponding set of sites. A *boundary curve for* $\mathcal{S}'$ is a closed s-monotone path in $\mathcal{J}_{s,S'} \cup \Gamma$ that contains all arcs in $\mathcal{S}'$. Note that we include Γ in the definition of a boundary curve so that we unify the various connected components of $\mathcal{J}_{s,S'}$ and obtain a single curve. The part of the plane enclosed in

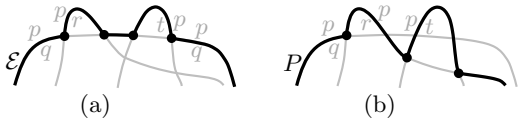


Figure 5 (a) The envelope $\mathcal{E} = \mathrm{env}(\mathcal{J}_{p,\{q,r,t\}})$. (b) A p-monotone path P in $\mathcal{J}_{p,\{q,r,t\}}$.

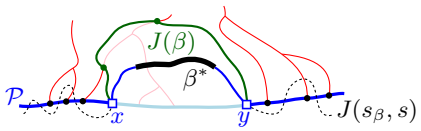


Figure 6 $\mathcal{P}_\beta = \mathcal{P} \oplus \beta$, core arc β^* is bold and black. The endpoints of $\beta \supseteq \beta^*$ are x and y.

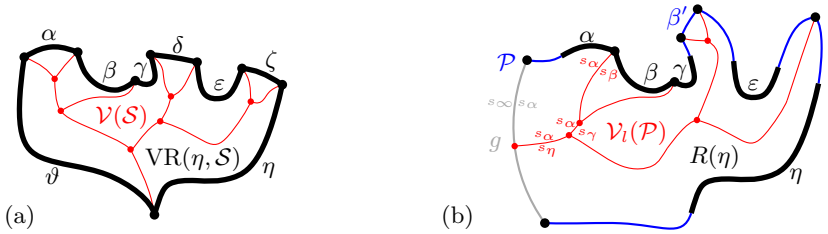


Figure 7 (a) illustrates $\mathcal{S}$ in black (bold) and $\mathcal{V}(\mathcal{S})$ in red, $\mathcal{S} = (\alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \eta, \vartheta)$. (b) illustrates $\mathcal{V}_l(\mathcal{P})$ for boundary curve $\mathcal{P} = (\alpha, \beta, \gamma, \beta', \varepsilon, \eta, g)$. $\mathcal{S}' = (\alpha, \beta, \gamma, \varepsilon, \eta)$ is shown in bold. The arcs of $\mathcal{P}$ are original except the auxiliary arc β' and the Γ-arc g.

a boundary curve $\mathcal{P}$ is called the *domain* of $\mathcal{P}$, denoted by $D_{\mathcal{P}}$. Given $\mathcal{P}$, we also use $S_{\mathcal{P}}$ to denote the set of sites S'.

Figure 7(b) illustrates a boundary curve for $\mathcal{S}' \subset \mathcal{S}$, where $\mathcal{S}$ is shown bold in Figure 7(a). $\mathcal{S}'$ can admit several different boundary curves, one being the envelope $\text{env}(\mathcal{J}_{s,S'} \cup \Gamma)$. A boundary curve $\mathcal{P}$ consists of pieces of bisectors in $\mathcal{J}_{s,S'}$, called *boundary arcs*, and pieces of Γ, called Γ-*arcs*; Γ-arcs indicate the openings of the domain to infinity. Among the boundary arcs in $\mathcal{P}$, those that contain an arc of $\mathcal{S}'$ are called *original* and others are called *auxiliary arcs*. Original arcs are expanded versions of the arcs in $\mathcal{S}'$; to differenciate among them the arcs in $\mathcal{S}$ are called *core arcs* (shown bold in Figure 7).

▶ **Definition 2.** Given a boundary curve $\mathcal{P}$ in $\mathcal{J}_{s,S'} \cup \Gamma$, a *Voronoi-like diagram* of $\mathcal{P}$ is a plane graph on $\mathcal{J}(S') = \{J(p,q) \in \mathcal{J} \mid p, q \in S'\}$ inducing a subdivision on the domain $D_{\mathcal{P}}$ as follows (see Figure 7(b)): (1) There is exactly one face $R(\alpha)$ for each boundary arc α of $\mathcal{P}$; $\partial R(\alpha)$ consists of the arc α and an s_α-monotone path in $\mathcal{J}_{s_\alpha,S'} \cup \Gamma$. (2) $\bigcup_{\alpha \in \mathcal{P} \setminus \Gamma} \overline{R(\alpha)} = \overline{D_{\mathcal{P}}}$. The Voronoi-like diagram of $\mathcal{P}$ is $\mathcal{V}_l(\mathcal{P}) = D_{\mathcal{P}} \setminus \bigcup_{\alpha \in \mathcal{P}} R(\alpha)$.

In the full paper, we prove that Voronoi-like regions are related to real Voronoi regions as supersets. For example, in Figure 7, the Voronoi-like region $R(\eta)$ (shown in 7(b)) is a superset of Voronoi region $\text{VR}(\eta, \mathcal{S})$ in 7(a); similarly for $R(\alpha)$. Real Voronoi regions are induced by the envelope $\mathcal{E}$ of $\mathcal{S}'$, where $\mathcal{E} = \text{env}(\mathcal{J}_{s,S'} \cup \Gamma)$, and $\mathcal{V}(\mathcal{E}) = \mathcal{V}(S') \cap D_{\mathcal{E}}$. It is not hard to see that $\mathcal{V}(\mathcal{E}) = \mathcal{V}_l(\mathcal{E})$. Thus, $\mathcal{V}_l(\mathcal{S})$ coincides with the real Voronoi diagram $\mathcal{V}(\mathcal{S})$.

In the full paper, we also prove that the Voronoi-like diagram of a boundary curve is unique (if it exists). The complexity of $\mathcal{V}_l(\mathcal{P})$ is $O(|\mathcal{P}|)$, where $|\cdot|$ denotes complexity, since it is a planar graph with exactly one face per boundary arc and vertices of degree 3 (or 1).

3 Insertion in a Voronoi-like diagram

Consider a boundary curve $\mathcal{P}$ for $\mathcal{S}' \subset \mathcal{S}$ and its Voronoi-like diagram $\mathcal{V}_l(\mathcal{P})$ within the domain $D_{\mathcal{P}}$. Let β^* be an arc in $\mathcal{S} \setminus \mathcal{S}'$, thus, β^* is contained in the closure of the domain $\overline{D_{\mathcal{P}}}$. We define arc $\beta \supseteq \beta^*$ as the connected component of $J(s, s_\beta) \cap \overline{D_{\mathcal{P}}}$ that contains β^* (see Figure 6). We also define an insertion operation $\oplus$ that inserts arc β in $\mathcal{P}$, deriving a new boundary curve $\mathcal{P}_\beta = \mathcal{P} \oplus \beta$, and inserts $R(\beta)$ in $\mathcal{V}_l(\mathcal{P})$, deriving $\mathcal{V}_l(\mathcal{P}_\beta) = \mathcal{V}_l(\mathcal{P}) \oplus \beta$. $\mathcal{P}_\beta$ is the boundary curve obtained by deleting the portion of $\mathcal{P} \cap D(s_\beta, s)$ between the endpoints of β and substituting it with β, see Figure 6. Figure 8 enumerates the possible cases of $\mathcal{P} \oplus \beta$ which is summarized in the following observation.

▶ Observation 3. Possible cases of inserting arc β in $\mathcal{P}$, see Figure 8. $D_{\mathcal{P}_\beta} \subseteq D_{\mathcal{P}}$.

(a) β straddles the endpoint of two consecutive boundary arcs; no arcs in $\mathcal{P}$ are deleted.

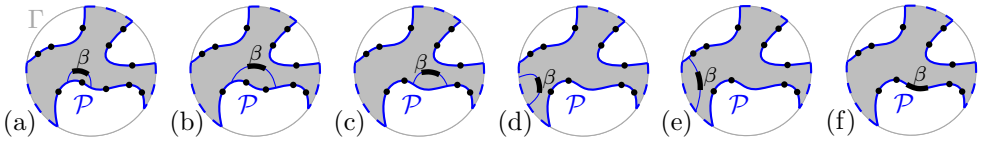


Figure 8 Insertion cases for an arc $\beta \supseteq \beta^*$. The domain $D_{\mathcal{P}}$ is shown shaded.

(b) (Auxiliary) arcs in $\mathcal{P}$ are deleted by β; their regions are also deleted from $\mathcal{V}_l(\mathcal{P}_\beta)$.
(c) An arc $\alpha \in \mathcal{P}$ is split into two arcs by β; $R(\alpha)$ in $\mathcal{V}_l(\mathcal{P})$ will also be split.
(d) A Γ-arc is split in two; $\mathcal{V}_l(\mathcal{P}_\beta)$ may switch from being a tree to being a forest.
(e) A Γ-arc is deleted or shrunk by inserting β. $\mathcal{V}_l(\mathcal{P}_\beta)$ may become a tree.
(f) $\mathcal{P}$ already contains a boundary arc $\bar{\beta} \supseteq \beta^*$; then $\beta = \bar{\beta}$ and $\mathcal{P}_\beta = \mathcal{P}$.

Note that $\mathcal{P}_\beta$ may contain fewer, the same number, or even one extra auxiliary arc compared to $\mathcal{P}$.

▶ **Lemma 4.** *The curve $\mathcal{P}_\beta = \mathcal{P} \oplus \beta$ is a boundary curve for $\mathcal{S}' \cup \{\beta^*\}$.*

Given $\mathcal{V}_l(\mathcal{P})$ and arc β, where $\beta^* \in \mathcal{S} \setminus \mathcal{S}'$, we define a *merge curve* $J(\beta)$, within $D_{\mathcal{P}} \cup \Gamma$, which delimits $\partial R(\beta)$ in $\mathcal{V}_l(\mathcal{P}_\beta)$, see Figure 6. We define $J(\beta)$ incrementally, starting at an endpoint of β. Let x and y denote the endpoints of β, where x, β, y are assumed in counterclockwise order around $\mathcal{P}_\beta$.

▶ **Definition 5.** Given $\mathcal{V}_l(\mathcal{P})$ and arc $\beta \subset J(s, s_\beta)$, the *merge curve* $J(\beta)$ is a path $(v_1, \ldots, v_m)$ in the arrangement of s_β-related bisectors $\mathcal{J}_{s_\beta, S_{\mathcal{P}}} \cup \Gamma$, connecting the endpoints of β, $v_1 = x$ and $v_m = y$. Each edge $e_i = (v_i, v_{i+1})$ is an arc of a bisector $J(s_\beta, \cdot)$ or an arc on Γ. For $i = 1$: if $x \in J(s_\beta, s_\alpha)$, then $e_1 \subseteq J(s_\beta, s_\alpha)$; if $x \in \Gamma$, then $e_1 \subseteq \Gamma$. Given v_i, vertex v_{i+1} and edge e_{i+1} are defined as follows. (Wlog we assume a clockwise ordering of $J(\beta)$).

1. If $e_i \subseteq J(s_\beta, s_\alpha)$, let v_{i+1} be the other endpoint of the component $J(s_\beta, s_\alpha) \cap R(\alpha)$ incident to v_i. If $v_{i+1} \in J(s_\beta, \cdot) \cap J(s_\beta, s_\alpha)$, then $e_{i+1} \subseteq J(s_\beta, \cdot)$. If $v_{i+1} \in \Gamma$, then $e_{i+1} \subseteq \Gamma$.
2. If $e_i \subseteq \Gamma$, let g be the Γ-arc incident to v_i. Let $e_{i+1} \subseteq J(s_\beta, s_\gamma)$, where $R(\gamma)$ is the first region, incident to g clockwise from v_i, such that $J(s_\beta, s_\gamma)$ intersects $g \cap \overline{R(\gamma)}$; let v_{i+1} be this intersection point.

In the full paper we prove the following theorem, which shows that $J(\beta)$ is well defined.

▶ **Theorem 6.** *$J(\beta)$ is a unique s_β-monotone path in $\mathcal{J}_{s_\beta, S_{\mathcal{P}}} \cup \Gamma$, which connects the endpoints of β. $J(\beta)$ can contain at most one edge per region of $\mathcal{V}_l(\mathcal{P})$, with the exception of the first and last edge, if v_1 and v_m are incident to the same face in $\mathcal{V}_l(\mathcal{P})$. $J(\beta)$ cannot intersect the interior of arc β.*

We define $R(\beta)$ as the area enclosed by $\beta \cup J(\beta)$. Let $\mathcal{V}_l(\mathcal{P}) \oplus \beta = \big((\mathcal{V}_l(\mathcal{P}) \setminus R(\beta)) \cup J(\beta)\big) \cap D_{\mathcal{P}_\beta}$ be the subdivision of $D_{\mathcal{P}_\beta}$ obtained by inserting $J(\beta)$ in $\mathcal{V}_l(\mathcal{P})$ and deleting any portion of $\mathcal{V}_l(\mathcal{P})$ enclosed by $J(\beta)$.

▶ **Theorem 7.** *$\mathcal{V}_l(\mathcal{P}) \oplus \beta$ is a Voronoi-like diagram for $\mathcal{P}_\beta = \mathcal{P} \oplus \beta$, denoted $\mathcal{V}_l(\mathcal{P}_\beta)$.*

The time complexity to compute $J(\beta)$ and update $\mathcal{V}_l(\mathcal{P}_\beta)$ is as follows: Let $\tilde{\mathcal{P}}$ denote the finer version of $\mathcal{P}$ as obtained by intersecting $\mathcal{P}$ with $\mathcal{V}_l(\mathcal{P})$. $|\tilde{\mathcal{P}}|$ is $O(|\mathcal{P}|)$, since $|\mathcal{V}_l(\mathcal{P})|$ is $O(|\mathcal{P}|)$. Let α and γ be the first original arcs on $\mathcal{P}_\beta$ occurring before and after β. Let $d(\beta)$ be the number of arcs in $\tilde{\mathcal{P}}$ between α and γ (both boundary and Γ-arcs). Given α, γ, and $\mathcal{V}_l(\mathcal{P})$, in all cases of Observation 3, except (c), the merge curve $J(\beta)$ and the diagram $\mathcal{V}_l(\mathcal{P}_\beta)$ can be computed in time $O(|R(\beta)| + d(\beta))$. In case (c), where an arc is split and a new arc ω is created by the insertion of β, the time is $O(|\partial R(\beta)| + |\partial R(\omega)| + d(\beta))$.

4 A randomized incremental algorithm

Consider a random permutation of the set of arcs $\mathcal{S}$, $o = (\alpha_1, \ldots, \alpha_h)$. For $1 \leq i \leq h$ define $\mathcal{S}_i = \{\alpha_1, \ldots, \alpha_i\} \subseteq \mathcal{S}$ to be the subset of the first i arcs in o. Given $\mathcal{S}_i$, let $\mathcal{P}_i$ denote a boundary curve for $\mathcal{S}_i$, which induces a domain $D_i = D_{\mathcal{P}_i}$. The randomized algorithm is inspired by the randomized, two-phase, approach of Chew [4] for the Voronoi diagram of points in convex position; however, it constructs Voronoi-like diagrams of boundary curves $\mathcal{P}_i$ within a series of shrinking domains $D_i \supseteq D_{i+1}$. In phase 1, the arcs in $\mathcal{S}$ get deleted one by one in reverse order of o, while recording the neighbors of each deleted arc at the time of its deletion. Let $\mathcal{P}_1 = \partial(D(s, s_{\alpha_1}) \cap D_\Gamma)$ and $D_1 = D(s, s_{\alpha_1}) \cap D_\Gamma$. Let $R(\alpha_1) = D_1$. $\mathcal{V}_l(\mathcal{P}_1) = \emptyset$ is the Voronoi-like diagram for $\mathcal{P}_1$. In phase 2, we start with $\mathcal{V}_l(\mathcal{P}_1)$ and incrementally compute $\mathcal{V}_l(\mathcal{P}_{i+1})$, $i = 1, \ldots, h-1$, by inserting arc α_{i+1} in $\mathcal{V}_l(\mathcal{P}_i)$, where $\mathcal{P}_{i+1} = \mathcal{P}_i \oplus \alpha_{i+1}$ and $\mathcal{V}_l(\mathcal{P}_{i+1}) = \mathcal{V}_l(\mathcal{P}_i) \oplus \alpha_{i+1}$. At the end, we obtain $\mathcal{V}_l(\mathcal{P}_h)$, where $\mathcal{P}_h = \mathcal{S}$. We have already established that $\mathcal{V}_l(\mathcal{S}) = \mathcal{V}(\mathcal{S})$, thus, the algorithm is correct. $\mathcal{P}_i$ may contain at most $2i$ arcs (see Observation 3), thus, the complexity of $\mathcal{V}_l(\mathcal{P}_i)$ is $O(i)$.

Given the results on Voronoi-like diagrams in Sections 2 and 3, the time analysis becomes similar to the one for the farthest-segment Voronoi diagram [6], with some additional cases to consider since $\mathcal{V}_l(\mathcal{P}_i)$ is a forest and not necessarily a tree.

▶ **Lemma 8.** *The expected number of arcs in $\tilde{\mathcal{P}}_i$ (auxiliary boundary arcs and fine Γ-arcs) that are visited while inserting α_{i+1} is $O(1)$.*

▶ **Theorem 9.** *Given an abstract Voronoi diagram $\mathcal{V}(S)$, $\mathcal{V}(S \setminus \{s\}) \cap VR(s, S)$ can be computed in expected $O(h)$ time, where h is the complexity of $\partial\, VR(s, S)$. Thus, $\mathcal{V}(S \setminus \{s\})$ can also be computed in expected time $O(h)$.*

References

1 A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Comput. Geometry*, 4:591–604, 1989.

2 F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations.* World Scientific, 2013.

3 C. Bohler, R. Klein, and C. Liu. Forest-like abstract Voronoi diagrams in linear time. In *Proc. 26th Canadian Conference on Computational Geometry (CCCG)*, 2014.

4 L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical report, Dartmouth College, Hanover, USA, 1990.

5 F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete & Computational Geometry*, 21(3):405–420, 1999.

6 E. Khramtcova and E. Papadopoulou. An expected linear-time algorithm for the farthest-segment Voronoi diagram. arXiv:1411.2816v3 [cs.CG], 2017. Preliminary version in *Proc. 26th Int. Symp. on Algorithms and Computation (ISAAC), LNCS* 9472, 404-414, 2015.

7 R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

8 R. Klein, E. Langetepe, and Z. Nilforoushan. Abstract Voronoi diagrams revisited. *Computational Geometry: Theory and Applications*, 42(9):885–902, 2009.

9 R. Klein and A. Lingas. Hamiltonian abstract Voronoi diagrams in linear time. In *Algorithms and Computation, 5th ISAAC*, volume 834 of *LNCS*, pages 11–19, 1994.

10 E. Papadopoulou and M. Zavershynskyi. The higher-order Voronoi diagram of line segments. *Algorithmica*, 74(1):415–439, 2016.

11 M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge university press, 1995.

Data Gathering in Faulty Sensor Networks Using a Mule

Stav Ashur[1]

1 **Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel**
stavshe@post.bgu.ac.il

Abstract

We study the problem mentioned in the title, assuming the underlying sensor network is a unit disk graph. That is, let S be a set of n sensors with transmission range 1. We wish to find a data gathering tree (i.e., a rooted spanning tree) for the network, and to augment it with a data mule based at one of the nodes of the tree. The mule's job is to collect the data from the children of a node u, if u is faulty. The goal is to find a gathering tree and to locate the mule at one of its nodes, so that the expected length of the mule's tour is minimized, where the mule can move freely in the plane. We present an $O(n \log n)$-time constant-factor approximation algorithm for this problem. Our algorithm is faster by a linear factor than the previous one due to Yedidsion et al. [5].

1 Introduction

Given a set S of n sensors with wireless capabilities deployed in the plane, one would like to gather the data collected by the sensors using a *data gathering tree* — a hierarchical structure that determines the paths in which data flows from the sensors to the storage center (i.e., a directed rooted tree). The transmission range of the sensors is 1, so our starting point is the *unit disk graph* (UDG) G induced by S, that is, the graph over S in which there is an edge between two sensors $s_1, s_2 \in S$ if and only if the Euclidean distance between them, $d(s_1, s_2)$, is at most 1.

Assuming G is connected, our goal is to find a rooted spanning tree T of G. However, it is possible that some node u of the tree is faulty, in which case, in order not to lose the data at the children of u, we employ a data mule that visits each of u's children and collects the data from them. The mule's location is fixed when the gathering tree is determined; it is at one of the nodes of the tree. Then, if some node u is faulty, the mule must leave its base, travel to each of u's children and return to its base, where the mule can move freely in the plane. Thus, we would like to find a rooted spanning tree T of G and to determine the mule's location, such that the expected length of the mule's tour is minimum. In other words, our goal is to find a rooted tree T and a node v of T, such that the sum of $TSP^v(u)$, over all internal nodes u of T, is minimum, where $TSP^v(u)$ is the shortest tour beginning and ending at v and visiting each of the children of u.

This problem was introduced by Crowcroft et al. [2], who only studied its one-dimensional version. Subsequently, Yedidsion et al. [5] considered the two-dimensional version of the problem and presented an $O(n^2 \log n)$-time constant-factor approximation algorithm for it. That is, their algorithm finds a rooted tree and places the mule at one of its nodes, so that the sum of tours corresponding to their tree is bounded by a constant times the sum corresponding to the optimal solution. In this paper, we present an alternative, more efficient, constant-factor approximation algorithm for the (two-dimensional version of the) problem. The running time of our algorithm is $O(n \log n)$.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

In general, some research has been done on various problems related to sensor networks that are augmented with mobile elements, see, e.g., [2–4]. See [5] for more details on related work.

2 Tree Construction

In this section we describe how to construct a spanning tree of the UDG induced by the set of sensors S. The constructed tree will have some desirable properties, mentioned below. In the subsequent section we will choose one of the nodes to serve as the tree's root (and as the mule's base); this will determine the direction of the edges of the tree.

We begin by laying a regular grid of edge length $\frac{1}{\sqrt{2}}$ over the input scene. (Notice that any two nodes within the same cell are at distance at most 1 from each other and are therefore connected by an edge in the underlying UDG.) Next, in each non-empty grid cell, we pick an arbitrary node to be the cell's *central node* (CN) and connect all other nodes in the cell to the CN. At this point the set of central nodes is a dominating set (DS) of UDG.

We now add some edges to connect between adjacent stars. More precisely, for any pair of stars, if the distance between them is at most one, i.e., if there exist a node u in one and v in the other such that $d(u, v) \leq 1$, then add an edge between the closest such pair of nodes. We do so by running the following algorithm.

---

1. For each cell X:
  Construct the Voronoi diagram of the nodes in X and preprocess it for point location queries.
2. For each cell X:
  For each of the 24 cells Y surrounding X (i.e., for each cell in the first or second circle around X):
  a. For each node $u \in Y$:
    Find the node v of X which is closest to u.
  b. Let (u, v) be the closest pair that was found.
  c. If $d(u, v) \leq 1$
    $E \leftarrow E \cup \{(u, v)\}$
3. Eliminate the cycles from the current graph by running DFS.

Let T be the tree that was obtained and notice that the set of its internal nodes is a connected dominating set (CDS) of UDG. We call an internal node of T a *backbone* node, and denote the set of internal nodes of T by BBN_T. Thus BBN_T is a CDS of UDG. Actually, BBN_T is an area constrained CDS (ACCDS) of UDG, where a CDS is an ACCDS if the number of nodes of the set in any disk of constant area A is $O(A)$. This is because the number of backbone nodes in any cell is bounded by 25 (the cell's CN plus at most 24 backbone nodes that are created by the code fragment above).

Finally, it is easy to see that the total time required to construct T is $O(n \log n)$. The total time spent on building the Voronoi diagrams and their corresponding search structures is $O(n \log n)$, and, for each node we perform a constant number of point location queries, so the total time spent on querying the diagrams is $O(n \log n)$.

3 Fixing the root and placing the mule

In [5], it was shown that the optimal solution with the additional constraint that the mule must be placed at the root is a 2-approximation of the unconstrained optimal solution. We therefore restrict our attention to the constrained version and present a constant-factor approximation for it, which in turn is a constant-factor approximation for the unconstrained version.

We focus on the more interesting and difficult case, where the area of the union of the unit disks around the sensors is greater than some constant. When this area is small, the number of backbone nodes (which is a linear function of the area) is small and the problem becomes much easier.

3.1 Placing the mule

Let T be the tree that was constructed in the previous section, and let $BBN_T = \{u_1, \ldots, u_m\}$ be the set of its internal nodes (i.e., backbone nodes). In this section, we describe how to determine the node of T in which the mule will be placed (and that will serve as the root of T). For a node u of T, let $w(u) = \sum_{i=1}^{m} d(u, u_i)$, that is, $w(u)$ is the sum of distances from u to the internal nodes of T. We would like to place the mule at the node of T for which this value is minimum, however we cannot afford to compute all these values. Instead, we choose a node v' whose value $w(v')$ is a good approximation of the minimum value.

We use the data structure of Bose et al. [1], which is built over a set of points in the plane (given some $\varepsilon > 0$), and which supports sum-of-distances queries. More precisely, we construct the data structure over the set BBN_T, in $O(\frac{1}{\epsilon^2} n \log n)$ time and $O(\frac{n}{\varepsilon^2})$ space, and perform n queries in it, one per each node in T, in total $O(\frac{1}{\epsilon^2} n \log n)$ time. The answer to a query with node u is a value $w_\varepsilon(u)$, such that $(1 - \varepsilon\sqrt{2})w(u) \leq w_\varepsilon(u) \leq (1 + \varepsilon\sqrt{2})w(u)$, and let v' be the node whose returned value (i.e., $w_\varepsilon(v')$) is minimum.

Denote by OPT_T the sum of tours corresponding to the optimal location in tree T, and denote by OPT_{T^*} the sum of tours corresponding to the optimal location in the (unknown) optimal tree T^*. We prove below that the sum of tours corresponding to v' is bounded by some constant times OPT_{T^*}.

▶ **Theorem 3.1.** *Choosing v' as the location for the mule yields a constant-factor approximation of OPT_{T^*}, for sufficiently large m. Moreover, the total running time of our algorithm is $O(n \log n)$.*

Proof. The proof is based on the following three claims, which correspond to Lemmas 3.2–3.5 below.

1. There exists a constant c' such that $OPT_T \leq c' \cdot OPT_{T^*}$.
2. Placing the mule at the node v such that $w(v) = min\{w(u) : u \in BBN_T\}$ yields a 2-approximation of OPT_T, for sufficiently large m.
3. The sum of tours corresponding to v' is a 4-approximation of the sum of tours corresponding to v, for sufficiently large m.

From these claims it follows that placing the mule at v' yields a $c = 8c'$-approximation of OPT_{T^*}, for sufficiently large m. As for the running time, the tree T is constructed in $O(n \log n)$ time, after which the node v' is found in $O(n \log n)$ time. ◀

We now prove the 3 lemmas mentioned in the proof of Theorem 3.1.

▶ **Lemma 3.2.** *There exists a constant c' such that $OPT_T \leq c' \cdot OPT_{T^*}$.*

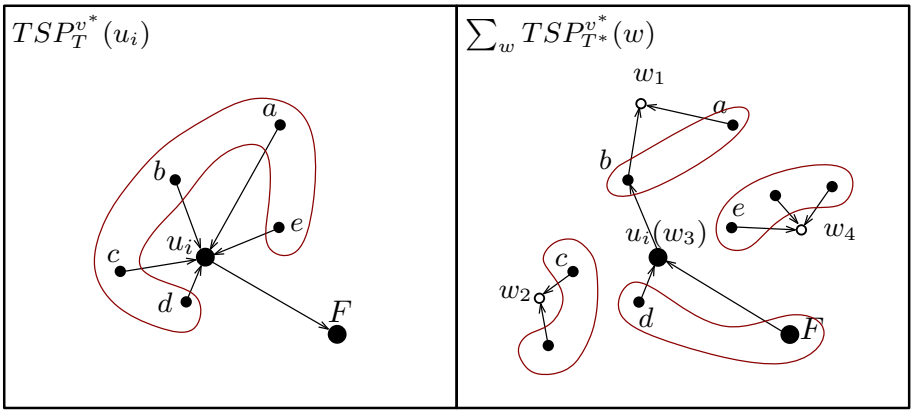


Figure 1 Proof of Lemma 3.2. Left: $C_T(u_i) = \{a, b, c, d, e\}$. Right: The nodes that T's mule will visit if u_i is faulty.

Proof. Let v^* be the optimal location in T^*. We will show that the sum of tours corresponding to v^* in T is already bounded by some constant c' times the sum of tours corresponding to v^* in T^*. But the former sum is at least OPT_T and the latter sum is equal to OPT_{T^*}, so we may conclude that $OPT_T \leq c' \cdot OPT_{T^*}$.

Let $C_T(u_i)$ be the set of u_i's children in T. Instead of visiting the nodes in $C_T(u_i)$ if u_i is faulty, T's mule will do the following. For each node w, which in T^* is a parent of a node in $C_T(u_i)$, T's mule will visit the nodes in $C_{T^*}(w)$. In other words, we replace $TSP_T^{v^*}(u_i)$ by $\sum_w TSP_{T^*}^{v^*}(w)$, where the sum is over all nodes w which in T^* have a child in $C_T(u_i)$; see Figure 1. Clearly, by doing so, T's mule will still visit the nodes in $C_T(u_i)$, but it may also visit other nodes, and in either way, the total distance traveled by T's mule can only increase.

Observe, however, that each of the nodes w in the latter sum is at distance at most 2 from u_i. So, since the internal nodes of T constitute an ACCDS, the number of internal nodes of T that lie within distance at most 2 from w is bounded by some constant c', and therefore, the number of times that T's mule will need to visit the nodes in $C_{T^*}(w)$ is bounded by some constant c'. We conclude that the total distance traveled by T's mule (after replacing the terms $TSP_T^{v^*}(u_i)$ by the corresponding sums) is bounded by $c' \sum_{w \in BBN_{T^*}} TSP_{T^*}^{v^*}(w) = c' \cdot OPT_{T^*}$, which implies that $\sum_{i=1}^m TSP_T^{v^*}(u_i) \leq c' \cdot OPT_{T^*}$. ◀

From now on, we are dealing only with the tree T, so we write TSP^v instead of TSP_T^v and BBN instead of BBN_T. Our proof of the next two lemmas relies on the following observation, which follows immediately from the way T was constructed.

▶ **Observation 3.3.** *Let v_1, v_2 be two nodes and let u be a backbone node. Then, there exists at most one node that is a child of u, when the mule is at v_1, but is not a child of u, when the mule is at v_2.*

▶ **Lemma 3.4.** *Placing the mule at the node v such that $w(v) = min\{w(u) : u \in BBN\}$ is a 2-approximation of OPT_T, for sufficiently large m.*

Proof. Let v^* be the node in which the mule is placed in the optimal solution for T, i.e., $\sum_{i=1}^m TSP^{v^*}(u_i) = OPT_T$. We first show that $\sum_{i=1}^m TSP^v(u_i) \leq \sum_{i=1}^m (TSP^{v^*}(u_i) + 6)$. Denote by s_i and t_i the first and last nodes that are visited in the tour taken by the mule based in v^* when u_i fails, and denote by $\pi(s_i, \ldots, t_i)$ the length of the portion of this tour

beginning at s_i and ending at t_i. Then, $TSP^{v^*}(u_i) = d(v^*, s_i) + \pi(s_i, \ldots, t_i) + d(t_i, v^*)$, and, by Observation 3.3, $TSP^{v}(u_i) \leq d(v, u_i) + d(u_i, s_i) + \pi(s_i, \ldots, t_i) + d(t_i, u_i) + d(u_i, v) + 2$, where 2 is an upper bound on the total length of the back-and-forth trips from u_i to visit the at most one child of v that is not also v^*'s child. So,

$$\begin{aligned}
\sum_{i=1}^{m} TSP^{v}(u_i) &\leq \sum_{i=1}^{m} (d(v, u_i) + d(u_i, s_i) + \pi(s_i, \ldots, t_i) + d(t_i, u_i) + d(u_i, v) + 2) \\
&\leq^{(i)} \sum_{i=1}^{m} (d(v^*, u_i) + d(u_i, s_i) + \pi(s_i, \ldots, t_i) + d(t_i, u_i) + d(u_i, v^*) + 2) \\
&\leq^{(ii)} \sum_{i=1}^{m} (d(v^*, s_i) + 1 + d(u_i, s_i) + \pi(s_i, \ldots, t_i) + d(t_i, u_i) + 1 + d(t_i, v^*) + 2) \\
&\leq \sum_{i=1}^{m} (TSP^{v^*}(u_i) + 6) \ ,
\end{aligned}$$

where inequality (i) is true since $w(v) \leq w(v^*)$ and inequality (ii) is true since $d(v^*, u_i) \leq d(v^*, s_i) + d(s_i, u_i) \leq d(v^*, s_i) + 1$, and similarly, $d(u_i, v^*) \leq d(u_i, t_i) + d(t_i, v^*) \leq 1 + d(t_i, v^*)$.

Now, since BBN is an ACCDS, if we assume that m is sufficiently large, then the average tour length from v^* (i.e., $(\sum_{i=1}^{m} TSP^{v^*}(u_i))/m$) is greater than 6, and therefore $\sum_{i=1}^{m}(TSP^{v^*}(u_i) + 6) \leq 2\sum_{i=1}^{m} TSP^{v^*}(u_i)$. Thus, for sufficiently large m, we get $\sum_{i=1}^{m} TSP^{v}(u_i) \leq 2\sum_{i=1}^{m} TSP^{v^*}(u_i) = 2OPT_T$. ◀

▶ **Lemma 3.5.** *$\sum_{i=1}^{m} TSP^{v'}(u_i) \leq 4\sum_{i=1}^{m} TSP^{v}(u_i)$, for sufficiently large m.*

Proof. Since $w_\varepsilon(v') \leq w_\varepsilon(v)$,

$$(1 - \varepsilon\sqrt{2})w(v') \leq w_\varepsilon(v') \leq w_\varepsilon(v) \leq (1 + \varepsilon\sqrt{2})w(v) \ ,$$

or

$$\sum_{i=1}^{m} d(v', u_i) \leq \frac{1 + \varepsilon\sqrt{2}}{1 - \varepsilon\sqrt{2}} \sum_{i=1}^{m} d(v, u_i) \ .$$

Now, as in the proof of Lemma 3.4, we write $TSP^{v}(u_i) = d(v, s_i) + \pi(s_i, \ldots, t_i) + d(t_i, v)$, where s_i and t_i are the first and last nodes visited by the mule based at v when u_i fails, and $\pi(s_i, \ldots, t_i)$ is the portion of $TSP^{v}(u_i)$ beginning at s_i and ending at t_i. Then,

$$\sum_{i=1}^{m} d(v', s_i) \leq \sum_{i=1}^{m} (d(v', u_i) + d(u_i, s_i)) \leq \frac{1 + \varepsilon\sqrt{2}}{1 - \varepsilon\sqrt{2}} \sum_{i=1}^{m} (d(v, u_i) + d(u_i, s_i)) \ ,$$

and, similarly,

$$\sum_{i=1}^{m} d(v', t_i) \leq \frac{1 + \varepsilon\sqrt{2}}{1 - \varepsilon\sqrt{2}} \sum_{i=1}^{m} (d(v, u_i) + d(u_i, t_i)) \ .$$

Again, as in the proof of Lemma 3.4,

$$\sum_{i=1}^{m} TSP^{v'}(u_i) \leq \sum_{i=1}^{m} (d(v', s_i) + \pi(s_i, \ldots, t_i) + d(t_i, v') + 2) \ ,$$

so

$$\begin{aligned}
\sum_{i=1}^{m} TSP^{v'}(u_i) &\leq \frac{1 + \varepsilon\sqrt{2}}{1 - \varepsilon\sqrt{2}} \sum_{i=1}^{m} (d(v, u_i) + d(u_i, s_i) + \pi(s_i, \ldots, t_i) + d(v, u_i) + d(u_i, t_i) + 2) \\
&\leq \frac{1 + \varepsilon\sqrt{2}}{1 - \varepsilon\sqrt{2}} \sum_{i=1}^{m} (TSP^{v}(u_i) + 6) \ .
\end{aligned}$$

Now, since BBN is an ACCDS, if we assume that m is sufficiently large, then the average tour length from v (i.e., $(\sum_{i=1}^{m} TSP^v(u_i))/m$) is greater than 6, and therefore $\sum_{i=1}^{m}(TSP^v(u_i)+6) \leq 2\sum_{i=1}^{m} TSP^v(u_i)$. Thus, for sufficiently large m, we get $\sum_{i=1}^{m} TSP^{v'}(u_i) \leq 4\sum_{i=1}^{m} TSP^v(u_i)$, by choosing $\varepsilon < 1/(3\sqrt{2})$. ◀

Acknowledgements

The author wishes to thank Matya Katz for his help during all stages of this work.

References

1 Prosenjit Bose, Anil Maheshwari, and Pat Morin. Fast approximations for sums of distances, clustering and the fermat-weber problem. *Comput. Geom.*, 24(3):135–146, 2003.

2 Jon Crowcroft, Liron Levin, and Michael Segal. Using data mules for sensor network data recovery. *Ad Hoc Networks*, 40:26–36, 2016.

3 M. Di Francesco, S. K. Das, and A. Giuseppe. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Transactions on Sensor Networks (TOSN)*, 8.1:7–38, 2011.

4 O. Tedas, V. Isler, J. h. Lim, and A. Terzis. Using mobile robots to harvest data from sensor fields. *IEEE Wireless Communications*, 16.1:22, 2009.

5 Harel Yedidsion, Aritra Banik, Paz Carmi, Matthew J. Katz, and Michael Segal. Efficient data retrieval in faulty sensor networks using a mobile mule. In *15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WiOpt 2017, Paris, France, May 15-19, 2017*, pages 1–8, 2017.

On Convex Polygons in Cartesian Products*

Jean-Lou De Carufel[1], Adrian Dumitrescu[2], Wouter Meulemans[3], Tim Ophelders[3], Claire Pennarun[4], Csaba D. Tóth[5], and Sander Verdonschot[6]

1 **University of Ottawa, Canada**
jdecaruf@uottawa.ca
2 **University of Wisconsin-Milwaukee, USA**
dumitres@uwm.edu
3 **TU Eindhoven, The Netherlands**
{w.meulemans|t.a.e.ophelders}@tue.nl
4 **LIRMM, CNRS & Université de Montpellier, France**
claire.pennarun@lirmm.fr
5 **California State University Northridge, Los Angeles, CA, USA**
csaba.toth@csun.edu
6 **Carleton University, Ottawa, Canada**
sander@cg.scs.carleton.ca

Abstract

We study several problems concerning convex polygons whose vertices lie on a grid defined by the Cartesian product of two sets of n real numbers, using each coordinate at most once. First, we prove that all such grids contain a convex polygon with $\Omega(\log n)$ vertices and that this bound is asymptotically tight. Second, we present two polynomial-time algorithms that find the largest convex polygon of a restricted type. These algorithms give an approximation of the unrestricted case. It is unknown whether the unrestricted problem can be solved in polynomial time.

1 Introduction

A fast way to generate a random convex polygon, based on a proof by Pavel Valtr [7], first generates two random sets of n integer coordinates before significantly transforming the y-coordinates to produce a convex n-gon with the original x-coordinates. What happens if we do not transform the y-coordinates, and instead ask for a convex polygon with the original x- and y-coordinates?

Formally, we say that two sets, X and Y, each containing n real numbers, form a *grid* $X \times Y$. A grid *supports* a convex polygon P if for every vertex of P, its x-coordinate is in X and its y-coordinate is in Y, and no two vertices of P share an x- or y-coordinate.

It turns out that not every $n \times n$ grid supports a convex n-gon. In fact, this is true already for $n = 5$ (see Figure 1). This raises several interesting questions. Can we quickly decide whether a grid supports a convex n-gon? Or can we find the largest k such that it supports a convex k-gon? And what is the largest k such that *any* $n \times n$ grid supports a convex k-gon? We initiate the study of these questions.

There is a rich history of problems involving convex subsets, including the famous Happy Ending Problem: that any set of five points in the plane in general position contain four

* This work was initiated at the 2017 Fields Workshop on Discrete and Computational Geometry. Meulemans is partially supported by the Netherlands eScience Center (NLeSC, grant 027.015.G02). Ophelders is supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208. Research by Tóth was partially supported by the NSF awards CCF-1422311 and CCF-1423615.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

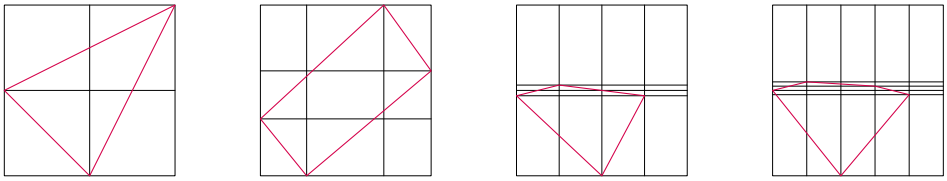


Figure 1 Maximum-size supported convex polygons of respective sizes 3, 4, 4, and 5 in $n \times n$ grids, where n is between 3 and 6.

points in convex position. Generalizing this result, Erdős and Szekeres conjectured that every set of $2^{n-2}+1$ points in general position contains n points in convex position and that this is tight [2]. While this conjecture has been proven only for $n \leq 6$ [6], and the current best upper bound is $2^{n+o(n)}$ [5], the asymptotics are known to be correct for the lower bound: a set of n points in general position always contains $\Omega(\log n)$ points in convex position [2].

Algorithmically, the problem of finding the largest convex subset of a set of n points in the plane in general position can be solved in $O(n^3)$ time [1]. While this approach likely generalizes to finding the largest convex subset in a grid, it is not clear how to include the restriction that each coordinate is used at most once. On the negative side, it was recently shown that the problem of finding the largest convex subset in a point set in $\mathbb{R}^d$ for dimensions $d \geq 3$ is NP-hard [3].

The remainder of the paper is structured as follows. First, in Section 2, we give an asymptotically tight lower bound on the maximum size convex polygon supported by an $n \times n$ grid. Then, in Section 3, we provide algorithms to find, for a given grid, the largest supported convex polygon of two special types. These algorithms give a constant-factor approximation of the size of the largest supported convex polygon.

2 General bounds

We consider two special types of convex polygons. We classify a convex polygon P with vertices $((x_1, y_1), \ldots, (x_k, y_k))$ (in clockwise order), as follows:

- *Convex caps* come in four types $\{\curvearrowright, \curvearrowleft, \curvearrowbelow, \curvearrowabove\}$. We have
  - $P \in \curvearrowright$ if and only if $(x_i)_{i=1}^k$ is increasing;
  - $P \in \curvearrowleft$ if and only if $(y_i)_{i=1}^k$ is increasing;
  - $P \in \curvearrowbelow$ if and only if $(x_i)_{i=1}^k$ is decreasing;
  - $P \in \curvearrowabove$ if and only if $(y_i)_{i=1}^k$ is decreasing.
- *Convex chains* come in four types $\{\nearrow, \searrow, \swarrow, \nwarrow\}$. We have
  $\nearrow = \curvearrowleft \cap \curvearrowright$, $\quad \searrow = \curvearrowright \cap \curvearrowabove$, $\quad \nwarrow = \curvearrowbelow \cap \curvearrowleft$, $\quad \swarrow = \curvearrowabove \cap \curvearrowbelow$.

Figure 1 illustrates some maximum-size supported convex polygons for various grids. For $n \times n$ grids with $n \leq 4$, the largest supported convex polygon always has size n. For $n > 4$, this size can be less than n (as for $n = 5$ in Figure 1). Interestingly, for $n = 6$, there always exists a supported convex polygon of size at least 5.

▶ **Lemma 2.1.** *Every 6×6 grid $X \times Y$ supports a convex polygon of size at least 5.*

Proof. Let $X' = X \setminus \{\min(X), \max(X)\}$ and $Y' = Y \setminus \{\min(Y), \max(Y)\}$. The 4×4 grid $X' \times Y'$ supports a convex chain P' of size 3 between two opposite corners of $X' \times Y'$. Then one x-coordinate $x' \in X'$ and one y-coordinate $y' \in Y'$ are not used by P'. Without loss of generality, assume that $P' \in \nearrow$. Then the convex polygon containing the points of P' and $(x', \min(Y))$ and $(\max(X), y')$ is a supported convex polygon of size 5 on $X \times Y$. ◀

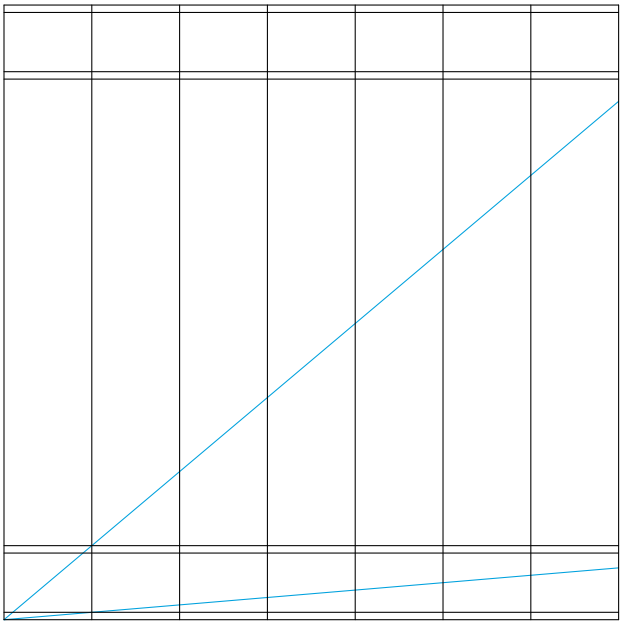


Figure 2 An 8×8 grid without convex chains of size greater than $4 = \log_2 8 + 1$. $X = \{1, \ldots, 8\}$, $Y = \{0, 7, 63, 70, 511, 518, 574, 581\}$. Two lines through pairs of grid points are drawn in blue.

More generally, by Lemma 2.2, every grid supports a convex chain of size $\Omega(\log n)$. We show in Lemma 2.3 that this bound is asymptotically tight: for each n, there exists a grid for which the maximum convex chain has size $O(\log n)$. Since every convex cap consists of two convex chains (some of which may be empty), and each convex polygon is composed of two convex caps, the same asymptotic bounds hold for maximum convex caps and polygons.

▶ **Lemma 2.2.** *Every $n \times n$ grid $X \times Y$ supports a convex polygon of size $\Omega(\log n)$.*

Proof. By Payne and Wood [4], every set of k points with at most ℓ collinear contains a set of $\Omega(\sqrt{k/\log \ell})$ points in general position. Here, we have n^2 points with at most n collinear, so there is a set of $\Omega(\sqrt{n^2/\log n}) = \Omega(n/\sqrt{\log n})$ points in general position. By Suk [5], every set of $2^{k+o(k)}$ points in general position contains a set of k points in convex position. Hence, we can find a subset of $\Omega\big(\log(n/\sqrt{\log n})\big) = \Omega(\log n)$ points in convex position. Eliminating points with the same x- or same y-coordinate reduces the size by at most 75%, so this asymptotic bound also holds when coordinates in X and Y may be used at most once. ◀

For the upper bound, we construct a family of grids without any large convex chain. For $n = 8$, this grid is depicted in Figure 2.

▶ **Lemma 2.3.** *For every $n \in \mathbb{N}$, there exists an $n \times n$ grid $X \times Y$ that does not support any convex chain of size greater than $\lceil \log n \rceil + 1$.*

Proof. Let $g(n)$ be the maximum value such that for all X and Y of size n, the grid $X \times Y$ supports a convex polygon of size $g(n)$; clearly $g(n)$ is nondecreasing. Let k be the minimum integer such that $n \leq 2^k$. We show that $g(2^k) \leq k + 1$ to establish that $g(n) \leq g(2^k) \leq k + 1$.

Without loss of generality, assume that $n = 2^k$, and let $X = \{1, \ldots, n\}$. For a k-bit integer m, let m_i be the bit at its i-th position, such that $m = \sum_{i=0}^{k-1} m_i 2^i$. Let $Y = \{\sum_{i=0}^{k-1} m_i(n^{i+1} - 1) \mid 0 \leq m \leq n-1\}$. Both X and Y are symmetric: $X = \{\max(X) + 1 - x \mid$

$x \in X\}$ and $Y = \{\max(Y) - y \mid y \in Y\}$. Thus, it suffices to show that no $P \in \nearrow$ of size greater than $k+1$ exists.

Consider $p = (x, y)$ and $p' = (x', y') \in X \times Y$ with $y = \sum_{i=0}^{k-1} m_i(n^{i+1} - 1)$ and $y' = \sum_{i=0}^{k-1} m'_i(n^{i+1} - 1)$. The slope of the line between p and p' is $\text{slope}(p, p') = \sum_{i=0}^{k-1}(m'_i - m_i)(n^{i+1} - 1)/(x' - x)$. Let j be the largest index such that $m_j \neq m'_j$. Assume that $x < x'$ and $y < y'$, then $1 \leq x' - x \leq n - 1$ and we bound the slope as follows:

$$\frac{n^{j+1} - 1}{n - 1} \leq \frac{n^{j+1} - 1 + \sum_{i=0}^{j-1}(m'_i - m_i)(n^{i+1} - 1)}{x' - x} = \text{slope}(p, p')$$

$$= \frac{\sum_{i=0}^{j}(m'_i - m_i)(n^{i+1} - 1)}{x' - x} \leq \frac{\sum_{i=0}^{j} n^{i+1} - 1}{1} = n\frac{n^{j+1} - 1}{n - 1} - 1 < n\frac{n^{j+1} - 1}{n - 1}.$$

Hence, $\text{slope}(p, p') \in I_j = [\frac{n^{j+1}-1}{n-1}, n\frac{n^{j+1}-1}{n-1}]$. Consider the family of intervals $I_0, I_1, \ldots, I_{k-1}$ defined analogously. For $n > 1$, we have $\max(I_j) < \min(I_{j+1})$. Suppose for a contradiction that some $P \in \nearrow$ is of size greater than $k+1$. Then, since the slopes of the first $k+1$ edges of P decrease monotonically, there must be three consecutive vertices $p = (x, y)$, $p' = (x', y')$, and $p'' = (x'', y'')$ of P such that both $\text{slope}(p, p') \in I_j$ and $\text{slope}(p', p'') \in I_j$. Let $y = \sum_{i=0}^{k-1} m_i(n^{i+1} - 1)$ and $y' = \sum_{i=0}^{k-1} m'_i(n^{i+1} - 1)$ and $y'' = \sum_{i=0}^{k-1} m''_i(n^{i+1} - 1)$. Then j is the largest index such that $m_j \neq m'_j$, and also the largest index such that $m'_j \neq m''_j$. Because $m < m' < m''$, we have $m_j < m'_j < m''_j$, which is impossible since each of m_j, m'_j and m''_j is either 0 or 1. Hence, there are no convex chains of size greater than $k+1$. ◀

3 Algorithms

In this section, we describe polynomial time algorithms for finding convex chains and caps of maximum size, as well as polynomial time approximation algorithms for finding the maximum size of a convex polygon. We make use of the following general observation:

▶ **Observation 3.1.** *If a supported convex polygon P is in a set of $\nearrow$, $\searrow$, $\swarrow$, $\nwarrow$, $\curvearrowright$, $\curvearrowleft$, $\cup$, or $\cap$, then any subsequence of P also lies in that set.*

Convex chains. Given a grid $X \times Y$, we provide an algorithm to compute a supported convex chain $P \in \searrow$ of maximum size. For this, we use a dynamic program to compute for each edge $(p_1, p_2) \in E = (X \times Y)^2$, the maximum size $R(p_1, p_2)$ of a chain of $\searrow$ with p_1 and p_2 as first two vertices, or (p_1) if $p_1 = p_2$ (in which case $R(p_1, p_2) = 1$). By Observation 3.1, removing the first vertex from a chain of $\searrow$ again yields a chain of $\searrow$.

▶ **Observation 3.2.** *If $A = (a_1, \ldots, a_k) \in \searrow$ and $B = (b_1, \ldots, b_\ell) \in \searrow$ with $k \geq 2$, $\ell \geq 2$ and $a_{k-1} = b_1$ and $a_k = b_2$, then $(a_1, \ldots, a_{k-2}, b_1, \ldots, b_\ell)$ also lies in $\searrow$ and has size $k + \ell - 2$.*

Conversely, by Observation 3.2, for a chain P in $\searrow$, adding a vertex v at the front yields a chain of $\searrow$ if $(v, p_1, p_2) \in \searrow$: the x- (resp., y-) coordinate of v is less (resp., greater) than those of vertices of P, so distinctness is maintained. Therefore we can find the maximum size of a chain starting with p_1 and p_2 based on chains without p_1 as follows:

$$R(p_1, p_2) = \begin{cases} -\infty & \text{if } p_1 \neq p_2 \text{ and } (p_1, p_2) \notin \searrow \\ 1 & \text{if } p_1 = p_2 \\ \max_{(p_1, p_2, v) \in \searrow \text{ or } v = p_2} R(p_2, v) + 1 & \text{otherwise.} \end{cases}$$

Since the x-coordinate of the first vertex of a $\searrow$ chain is less than those of subsequent vertices, this formula is well defined. For sequences of constant size, membership in $\searrow$

can be checked in constant time. So the running time to compute $R(e)$ for all edges is $O(|E| \cdot |X \times Y|) = O(n^6)$, and the space complexity is $O(|E|) = O(n^4)$. This algorithm can easily be adapted to find the maximum size convex chains in $\{\nearrow, \searrow, \swarrow, \nwarrow\}$ with the same time and space complexity. We thus conclude the following:

▶ **Lemma 3.3.** *For a given $n \times n$ grid, we can compute a maximum size convex chain in $O(n^6)$ time and $O(n^4)$ space.*

Convex caps. To compute the maximum size of a convex cap in $\frown$, we compute the maximum size of two convex chains that use distinct y-coordinates. Specifically, for two edges $l = (l_1, l_2)$ and $r = (r_1, r_2)$, we compute the maximum total size $C(l, r)$ of a pair of chains $A \in \nearrow$ and $B \in \searrow$ such that their vertices use distinct y-coordinates and such that A ends with vertices l_1 and l_2 (or $A = (l_1)$ if $l_1 = l_2$), and B starts with vertices r_1 and r_2 (or $B = (r_1)$ if $r_1 = r_2$). To compute $C(l, r)$, we reuse the algorithm of Lemma 3.3 to compute $L(p_1, p_2)$ (resp., $R(p_1, p_2)$), the size of a largest convex chain P in $\nearrow$ (resp., $\searrow$), ending (resp., starting) in vertices p_1 and p_2, or $P = (p_1)$ if $p_1 = p_2$.

The desired quantity $C(l, r)$ can now be computed using a dynamic program. The main idea is that we can always safely eliminate the highest vertex of the two chains, to find a smaller subproblem, as this vertex cannot be (implicitly) part of the optimal solution to a subproblem. In particular, if l is a single vertex and it is highest, we can simply use the value of $R(r_1, r_2)$, incrementing it by one for the one vertex of l. Analogously, we handle the case if r is or both l and r are a single vertex. The interesting case is when both chains end in an edge. Here, we observe that we can easily check whether l and r use unique coordinates. If not, then this subproblem is invalid; otherwise, we may find a smaller subproblem by eliminating the highest vertex and checking all possible subchains that could lead to it.

With the reasoning above, we obtain the recurrence below. The first case eliminates invalid edges and combinations that use a coordinate more than once or that do not give a cap. After the first, it holds that $l \in \nearrow$, $r \in \searrow$ and that l and r use unique coordinates.

$$C(l,r) = \begin{cases} -\infty & \text{if } l_1 \neq l_2 \text{ and } l \notin \nearrow, \text{ or} \\ & \quad r_1 \neq r_2 \text{ and } r \notin \searrow, \text{ or} \\ & \quad \{l_1.y, l_2.y\} \cap \{r_1.y, r_2.y\} \neq \emptyset \\ 2 & \text{otherwise, if } l_1 = l_2 \text{ and } r_1 = r_2 \\ L(l_1, l_2) + 1 & \text{otherwise, if } r_1 = r_2 \text{ and } l_2.y < r_1.y \\ R(r_1, r_2) + 1 & \text{otherwise, if } l_1 = l_2 \text{ and } l_2.y > r_1.y \\ \max_{(v,l_1,l_2) \in \nearrow \text{ or } v = l_1} C((v, l_1), r) + 1 & \text{otherwise, if } l_2.y > r_1.y \\ \max_{(r_1,r_2,v) \in \searrow \text{ or } v = r_1} C(l, (r_2, v)) + 1 & \text{otherwise, } l_2.y < r_1.y. \end{cases}$$

We can compute $C(l, r)$ for all l and r in $O(|E|^2|X \times Y|) = O(n^{10})$ time and $O(|E|^2) = O(n^8)$ space. With $C(l, r)$, we can easily find the size of a maximum size cap P in $\frown$, using the observation below, and analogous observations for the special case $k = 1$ and/or $\ell = 1$.

▶ **Observation 3.4.** *If $A = (a_1, \ldots, a_k) \in \nearrow$ and $B = (b_1, \ldots, b_\ell) \in \searrow$ with $k \geq 2$, $\ell \geq 2$ and $(a_{k-1}, a_k, b_1, b_2) \in \frown$ and A and B use distinct y-coordinates, then $(a_1, \ldots, a_{k-2}, b_1, \ldots, b_\ell)$ lies in $\frown$ and has size $k + \ell$.*

▶ **Lemma 3.5.** *For a given $n \times n$ grid, we can compute a maximum size convex cap in $O(n^{10})$ time and $O(n^8)$ space.*

Convex n-chains and n-caps. If we are solely interested in deciding whether a convex chain or cap exists that has $|X| = |Y| = n$ vertices, we can improve upon the previous algorithms

considerably. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ with $x_i < x_{i+1}$ and $y_i < y_{i+1}$. To test whether $\nearrow$ supports a chain of size n, it suffices to test the chain $((x_1, y_1), \ldots, (x_n, y_n))$, which can be done in linear time.

To test whether $\curvearrowright$ supports a convex cap of size n, we adapt the algorithm of Lemma 3.5. Suppose P is a cap of $\curvearrowright$ of size n. For $k < n$, let $A_k \in \nearrow$ and $B_k \in \searrow$ be the subchains of A_P and B_P obtained after discarding vertices with y-coordinates greater than y_k. Let (l_1, l_2) be the last edge of A_k and let (r_1, r_2) be the first edge of B_k. Then h, i and j exist such that $i < j < k$ and $l_1.x = x_h$, $l_2.x = x_{h+1}$, $r_1.x = x_{n-k+h+2}$, $r_2.x = x_{n-k+h+3}$ and $\{l_1.y, l_2.y, r_1.y, r_2.y\} = \{y_i, y_j, y_{k-1}, y_k\}$. Suppose we adapt the formula for $C(l, r)$ to consider only entries of this form, and adapt the formulas for L and R to consider only entries of the form $((x_{i-1}, y_{i-1}), (x_i, y_i))$ and $((x_{n-i}, y_{i-1}), (x_{n-i+1}, y_i))$, respectively. We then obtain $O(|Y|^3|X|)$ possible values for (l, r), and the corresponding values can be computed in $O(|Y|^4|X|) = O(n^5)$ time and $O(n^4)$ space. Testing whether $\curvearrowright$ supports a cap of size n can be done within the same time and space bounds.

Approximation. Although an efficient algorithm for computing the maximum size of a supported convex polygon is left as an open problem, the algorithms above provide constant-factor approximations. A convex cap $P \in \curvearrowright$ is composed of two convex chains ($A_P \in \nearrow$ and $B_P \in \searrow$ as defined before), which are themselves caps in $\curvearrowright$, and one of which has at least half the size of P. Hence, the algorithms to compute the maximum size of an x- and y-monotone chain provide a factor $\frac{1}{2}$-approximation on the size of the largest cap. Similarly, a convex polygon P is composed of four x- and y-monotone convex chains, one of which contains at least a quarter of the vertices of P. Furthermore, P is composed of a convex cap and a convex cup, one of which contains at least half of the vertices of P. Thus, the algorithms in Lemma 3.3 and Lemma 3.5, respectively, yield factor $\frac{1}{4}$- and $\frac{1}{2}$-approximations for the maximum size of a convex polygon supported by $X \times Y$.

References

1 V. Chvátal and G. Klincsek. Finding largest convex subsets. In *Proc. 11th SE Conf. on Combin., Graph Theory and Comp*, volume 29, pages 453–460, 1980.

2 P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio mathematica*, 2:463–470, 1935.

3 Panos Giannopoulos, Christian Knauer, and Daniel Werner. On the computational complexity of Erdős-Szekeres and related problems in $\mathbb{R}^3$. In *European Symposium on Algorithms*, pages 541–552. Springer, 2013.

4 Michael S. Payne and David R. Wood. On the general position subset selection problem. *SIAM Journal on Discrete Mathematics*, 27(4):1727–1733, 2013.

5 Andrew Suk. On the Erdős-Szekeres convex polygon problem. *Journal of the American Mathematical Society*, 30:1047–1053, 2017.

6 George Szekeres and Lindsay Peters. Computer solution to the 17-point Erdős-Szekeres problem. *The ANZIAM Journal*, 48(2):151–164, 2006.

7 Pavel Valtr. Probability that n random points are in convex position. *Discrete & Computational Geometry*, 13(3-4):637–643, 1995.

Rollercoasters: Long Sequences without Short Runs

Therese Biedl[1], Ahmad Biniaz[1], Robert Cummings[1], Anna Lubiw[1], Florin Manea[2], Dirk Nowotka[2], and Jeffrey Shallit[1]

1 **Cheriton School of Computer Science, University of Waterloo, Canada.**
2 **Department of Computer Science, Kiel University, Germany.**

Abstract

A *rollercoaster* is a sequence of real numbers for which every maximal contiguous subsequence, that is increasing or decreasing, has length at least three. By translating this sequence to a set of points in the plane, a rollercoaster can be defined as a polygonal path for which every maximal sub-path, with positive- or negative-slope edges, has at least three points. Given a sequence of distinct real numbers, the rollercoaster problem asks for a maximum-length (not necessarily contiguous) subsequence that is a rollercoaster. It was conjectured that every sequence of n distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$ for $n > 7$, while the best known lower bound is $\Omega(n/\log n)$. In this paper we prove this conjecture. Our proof is constructive and implies a linear-time algorithm for computing a rollercoaster of this length. Extending the $O(n \log n)$-time algorithm for computing a longest increasing subsequence, we show how to compute a maximum-length rollercoaster within the same time bound. A maximum-length rollercoaster in a permutation of $\{1, \ldots, n\}$ can be computed in $O(n \log \log n)$ time.

The search for rollercoasters was motivated by orthogeodesic point-set embedding of caterpillars. A *caterpillar* is a tree such that deleting the leaves gives a path, called the *spine*. A *top-view caterpillar* is one of degree 4 such that the two leaves adjacent to each vertex lie on opposite sides of the spine. As an application of our result on rollercoasters, we are able to find a planar drawing of every n-node top-view caterpillar on every set of $\frac{25}{3}n$ points in the plane, such that each edge is an orthogonal path with one bend. This improves the previous best known upper bound on the number of required points, which is $O(n \log n)$. We also show that such a drawing can be obtained in linear time, provided that the points are given in sorted order.

1 Introduction

A *run* in a sequence of real numbers is a maximal contiguous subsequence that is increasing (an "ascent") or decreasing (a "descent"). A *rollercoaster* is a sequence of real numbers such that every run has length at least three. For example the sequence $(8, 5, 1, 3, 4, 7, 6, 2)$ is a rollercoaster with runs $(8, 5, 1)$, $(1, 3, 4, 7)$, $(7, 6, 2)$, which have lengths 3, 4, 3, respectively. The sequence $(8, 5, 1, 7, 6, 2, 3, 4)$ is not a rollercoaster because its run $(1, 7)$ has length 2. Given a sequence $S = (s_1, s_2, \ldots, s_n)$ of n distinct real numbers, the rollercoaster problem is to find a maximum-size set of indices $i_1 < i_2 < \cdots < i_k$ such that $(s_{i_1}, s_{i_2}, \ldots, s_{i_k})$ is a rollercoaster. In other words, this problem asks for a longest rollercoaster in S, i.e., a longest subsequence of S that is a rollercoaster.

One can interpret S as a set P of points in the plane by translating each number $s_i \in S$ to a point $p_i = (i, s_i)$. With this translation, a rollercoaster in S translates to a "rollercoaster" in P, which is a polygonal path whose vertices are points of P and such that every maximal sub-path, with positive- or negative-slope edges, has at least three points. See Figure 1(a). Conversely, for any point set in the plane, the y-coordinates of the points, ordered by their x-coordinates, forms a sequence of numbers. Therefore, any rollercoaster in P translates to a rollercoaster of the same length in S.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

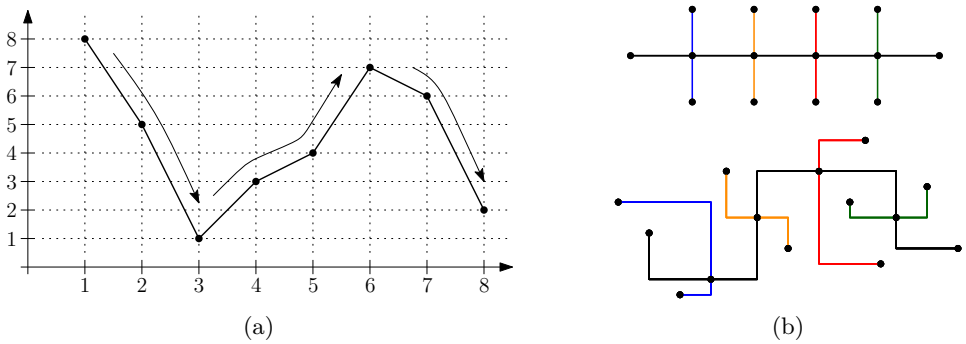


Figure 1 (a) Translating the sequence $(8, 5, 1, 3, 4, 7, 6, 2)$ to a set of points. (b) A planar L-shaped drawing of a top-view caterpillar.

The best known lower bound on the length of a longest rollercoaster is $\Omega(n/\log n)$ due to Biedl et al. [2]. They conjectured that

▶ Conjecture 1.1. Every sequence of $n > 7$ distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$.

Conjecture 1.1 can be viewed as a statement about patterns in permutations, a topic with a long history, and the subject of much current research. For example, the Eulerian polynomials, introduced by Euler in 1749, are the generating function for the number of descents in permutations. For surveys of recent work, see, for example, Linton et al. [7] and Kitaev [6]. Specifically, Conjecture 1.1 is related to the following seminal result of Erdős and Szekeres [3] in the sense that they prove the existence of an increasing or a decreasing subsequence of length at least $\sqrt{n}+1$ for $n = ab+1$, which is essentially a rollercoaster with one run.

▶ **Theorem 1.2** (Erdős and Szekeres, 1935)**.** *Every sequence of $ab+1$ distinct real numbers contains an increasing subsequence of length at least $a+1$ or a decreasing subsequence of length at least $b+1$.*

Hammersley [5] gave an elegant proof of the Erdős-Szekeres theorem that is short, simple, and based on the pigeonhole principle. The Erdős-Szekeres theorem also follows from the well-known decomposition of Dilworth (see [9]). The following is a restatement of Dilworth's decomposition for sequences of numbers.

▶ **Theorem 1.3** (Dilworth, 1950)**.** *Any finite sequence S of distinct real numbers can be partitioned into k ascending sequences where k is the maximum length of a descending sequence in S.*

Besides its inherent interest, the study of rollercoasters is motivated by point-set embedding of caterpillars [2]. A *caterpillar* is a tree such that deleting the leaves gives a path, called the *spine*. An *ordered caterpillar* is a caterpillar in which the cyclic order of edges incident to each vertex is specified. A *top-view caterpillar* is an ordered caterpillar where all vertices have degree 4 or 1 such that the two leaves adjacent to each vertex lie on opposite sides of the spine. Planar orthogonal drawings of trees on a fixed set of points in the plane have been explored recently, see e.g., [2, 4, 8]; in these drawings every edge is drawn as an orthogonal path between two points, and the edges are non-intersecting. A *planar L-shaped drawing* is a simple type of planar orthogonal drawing in which every edge is an orthogonal path of exactly two segments. Such a path is called an *L-shaped edge*. For example see the

top-view caterpillar in Figure 1(b) together with a planar L-shaped drawing on a given point set. Biedl et al. [2] proved that every top-view caterpillar on n vertices has a planar L-shaped drawing on every set of $O(n \log n)$ points in the plane that is in *general orthogonal position*, meaning that no two points have the same x- or y-coordinate.

Due to space restrictions we cannot give all the proofs. We refer the interested reader to the full version [1].

2 Rollercoasters

Our main result is to show that Conjecture 1.1 holds. In fact we prove something stronger: every sequence of n distinct numbers contains two rollercoasters of total length n. Our proof is constructive and yields a linear-time algorithm for computing such rollercoasters. The length 4 sequence $(3, 4, 1, 2)$ has no rollercoaster, and it can be shown that for $n = 5, 6, 7$ the longest rollercoaster has length 3. Therefore, we only consider $n \geq 8$.

▶ **Theorem 2.1.** *Every sequence of $n \geqslant 8$ distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$; such a rollercoaster can be computed in linear time. The lower bound of $\lceil n/2 \rceil$ is tight in the worst case.*

Proof. Consider a sequence with $n \geqslant 8$ distinct real numbers, and let P be its point-set translation with points $p_1, \ldots, p_n$ that are ordered from left to right. We define a *pseudo-rollercoaster* as a sequence in which every run is a 3-ascent (an ascent of length at least 3) or a 3-descent, except possibly the first run. We present an algorithm that computes two pseudo-rollercoasters R_1 and R_2 in P such that $|R_1| + |R_2| \geqslant n$; the length of the longer one is at least $\lceil n/2 \rceil$. Then with a more involved proof we show how to extend this longer pseudo-rollercoaster to obtain a rollercoaster of length at least $\lceil n/2 \rceil$; this will prove the lower bound.

First we provide a high-level description of our algorithm as depicted in Figure 2. Our algorithm is iterative, and proceeds by sweeping the plane by a vertical line ℓ from left to right. We maintain the following invariant: At the beginning of every iteration we have two pseudo-rollercoasters whose union is the set of all points to the left of ℓ and such that the last run of one of them is an ascent and the last run of the other one is a descent. Furthermore, these two last runs have a point in common.

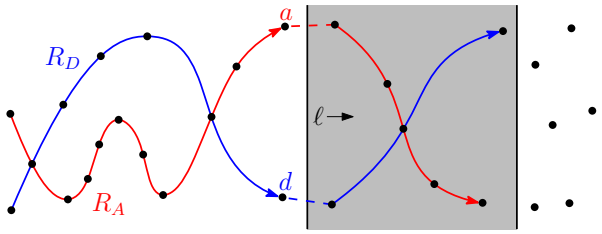


Figure 2 One iteration of algorithm: Constructing two pseudo-rollercoasters.

During every iteration we move ℓ forward and try to extend the current pseudo-rollercoasters. If this is not immediately possible with the next point, then we move ℓ farther and stop as soon as we are able to split all the new points into two chains that can be appended to the current pseudo-rollercoasters to obtain two new pseudo-rollercoasters that satisfy the invariant. See Figure 2. Now we present our iterative algorithm in detail.

The First Iteration: We take the leftmost point p_1, and initialize each of the two pseudo-rollercoasters by p_1 alone. We may consider one of the pseudo-rollercoasters to end in an

ascent and the other pseudo-rollercoaster to end in a descent. The two runs have a point in common.

An Intermediate Iteration: By the above invariant we have two pseudo-rollercoasters R_A and R_D whose union is the set of all points to the left of ℓ and such that the last run of one of them, say R_A, is an ascent and the last run of R_D is a descent. Furthermore, the last run of R_A and the last run of R_D have a point in common. During the current iteration we make sure that every swept point will be added to R_A or R_D or both. We also make sure that at the end of this iteration the invariant will hold for the next iteration. Let a and d denote the rightmost points of R_A and R_D, respectively; see Figure 2. Let p_i be the first point to the right of ℓ. If p_i is above a, we add p_i to R_A to complete this iteration. Similarly, if p_i is below d, we add p_i to R_D to complete this iteration. In either case we get two pseudo-rollercoasters that satisfy the invariant for the next iteration. Thus we may assume that p_i lies below a and above d. In particular, this means that a lies above d.

Consider the next point p_{i+1}. (If there is no such point, go to the last iteration.) Suppose without loss of generality that p_{i+1} lies above p_i as depicted in Figure 3. Then d, p_i, p_{i+1} forms a 3-ascent. Continue considering points $p_{i+2}, \ldots, p_k$ until for the first time, there is a 3-descent in $a, p_i, \ldots, p_k$. In other words, k is the smallest index for which $a, p_i, \ldots, p_k$ contains a descending chain of length 3. (If we run out of points before finding a 3-descent, then go to the last iteration.)

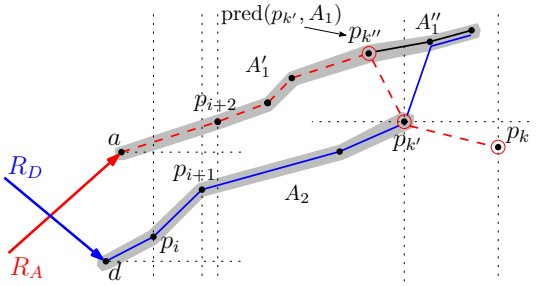


Figure 3 Illustration of an intermediate iteration of the algorithm.

Without p_k there is no descending chain of length 3. Thus the longest descending chain has two points, and by Theorem 1.3, the sequence $P' = a, p_i, p_{i+1}, \ldots, p_{k-1}$ is the union of two ascending chains. We give an algorithm to find two such chains A_1 and A_2 with A_1 starting at a and A_2 starting at p_i. The algorithm also finds the 3-descent ending with p_k. For every point $q \in A_2$ we define its A_1-*predecessor* to be the rightmost point of A_1 that is to the left of q. We denote the A_1-predecessor of q by $\text{pred}(q, A_1)$.

The algorithm is as follows: While moving ℓ forward, we denote by r_1 and r_2 the rightmost points of A_1 and A_2, respectively; at the beginning $r_1 = a$, $r_2 = p_i$, and $\text{pred}(p_i, A_1) = a$. Let p be the next point to be considered. If p is above r_1 then we add p to A_1. If p is below r_1 and above r_2, then we add p to A_2 and set $\text{pred}(p, A_1) = r_1$. If p is below r_2, then we find our desired first 3-descent formed by (in backwards order) $p_k = p$, $p_{k'} = r_2$, and $p_{k''} = \text{pred}(r_2, A_1)$. See Figure 3. This algorithm runs in time $O(k-i)$, which is proportional to the number of swept points.

We add point d to the start of chain A_2. The resulting chains A_1 and A_2 are shaded in Figure 3. Observe that A_2 ends at $p_{k'}$. Also, all points of P' that are to the right of $p_{k'}$ (if there are any) belong to A_1, and lie to the right of $p_{k''}$, and form an ascending chain. Let A_1'' be this ascending chain. Let A_1' be the sub-chain of A_1 up to $p_{k''}$; see Figure 3. Now we form one pseudo-rollercoaster (shown in red/dashed) consisting of R_A followed by A_1' and

then by the descending chain $p_{k''}, p_{k'}, p_k$. We form another pseudo-rollercoaster (shown in blue/solid) consisting of R_D followed by A_2 and then by A_1''. We need to verify that the ascending chain added after d has length at least 3. This chain contains d, p_i and $p_{k'}$. This gives a chain of length at least 3 unless $k' = i$, but in this case $p_{k''} = a$, so p_{i+1} is part of A_1'' and consequently part of this ascending chain. Thus we have constructed two longer pseudo-rollercoasters whose union is the set of all points up to point p_k, one ending with a 3-ascent and one with a 3-descent and such that the last two runs share the point $p_{k'}$. Figure 4(a) shows an intermediate iteration.

The Last Iteration: If there are no points left, then we terminate the algorithm. Otherwise, let p_i be the first point to the right of ℓ. Let a and d be the endpoints of the two pseudo-rollercoasters obtained so far, such that a is the endpoint of an ascent and d is the endpoint of a descent. Notice that p_i is below a and above d, because otherwise this iteration would be an intermediate one. For the same reason, the remaining points $p_i, \ldots, p_n$ do not contain a 3-ascent together with a 3-descent. If p_i is the last point, i.e., $i = n$, then we discard this point and terminate this iteration. Assume that $i \neq n$, and suppose without loss of generality that the next point p_{i+1} lies above p_i. In this setting, by Theorem 1.3 and as described in an intermediate iteration, with the remaining points, we can get two ascending chains A_1 and A_2 such that A_2 contains at least two points. By connecting A_1 to a and A_2 to d we get two pseudo-rollercoasters whose union is all the points (in this iteration we do not need to maintain the invariant).

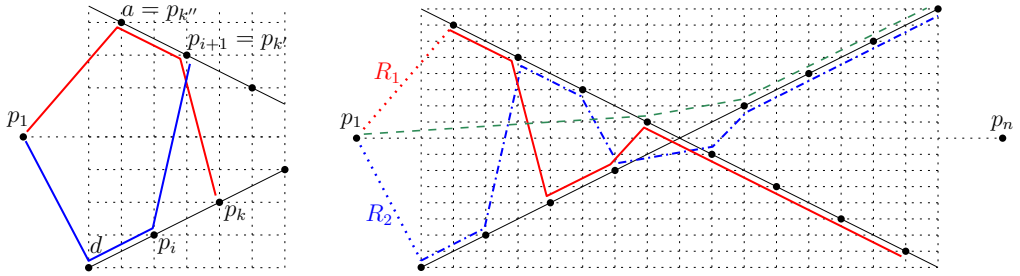


Figure 4 (a) An intermediate iteration. (b) A point set for which any rollercoaster of length at least $n/4 + 3$ does not contain p_1 and p_n. The green (dashed) rollercoaster, which contains p_1, has length $n/4 + 2$. The red (solid) and blue (dash-dotted) chains are the two rollercoasters returned by our algorithm.

Final Refinement: At the end of the algorithm, we obtain two pseudo-rollercoasters R_1 and R_2 that share p_1, and their union contains all points of P, except possibly p_n. Thus, $|R_1| + |R_2| \geqslant n$, and the length of the longer one is at least $\lceil \frac{n}{2} \rceil$.

This ends the presentation of our algorithm. It is not hard to see that the algorithm runs in $O(n)$ time.

To obtain rollercoasters (not just pseudo-rollercoasters), we remove p_1 from R_1 and/or R_2 if the first run only contains two points. This gives two rollercoasters $\mathcal{R}_1$ and $\mathcal{R}_2$ whose union contains all points, except possibly p_1 and p_n. The length of the longer one is at least $\lceil \frac{n-2}{2} \rceil$. We can improve this bound to $\lceil \frac{n}{2} \rceil$ by revisiting the first and last iterations of our algorithm with some case analysis.

We note that there are point sets, with n points, for which every rollercoaster of length at least $n/4 + 3$ does not contain any of p_1 and p_n; see e.g., the point set in Figure 4(b). To verify the tightness of the $\lceil n/2 \rceil$ lower bound, consider a set of n points in the plane where

$\lceil n/2 \rceil$ of which lie on a positive-slope line segment in the $(-,+)$-quadrant and the other $\lfloor n/2 \rfloor$ points lie on a positive-slope lines segment in the $(+,-)$-quadrant. ◀

3 Further Results

Our result can be extended to k-rollercoasters, i.e., sequences of real numbers in which every run is either a k-ascent or a k-descent. Namely, for $k \geqslant 4$, every sequence of $n \geqslant (k-1)^2+1$ distinct real numbers contains a k-rollercoaster of length at least $\frac{n}{2(k-1)} - \frac{3k}{2}$.

The algorithm presented in the proof of Theorem 2.1 does not necessarily compute the longest rollercoaster in a sequence. This can be done in $O(n \log n)$-time by an algorithm extending the classical algorithm for computing a longest increasing subsequence. This algorithm can be implemented in $O(n \log \log n)$ time if each number in the input sequence is an integer that fits in a constant number of memory words. Connected to this last result, we give an estimate on the number of permutations of $\{1, \ldots, n\}$ that are rollercoasters. Namely, let $r(n)$ be the number of permutations of $\{1, 2, \ldots, n\}$ that are rollercoasters. We show that $r(n) \sim c' \cdot n! \cdot \lambda^{n-3}$ where c' is a constant, approximately 0.204.

Finally, we study the problem of drawing a top-view caterpillar, with L-shaped edges, on a set of points in the plane that is in general orthogonal position. Recall that a top-view caterpillar is an ordered caterpillar of degree 4 such that the two leaves adjacent to each vertex lie on opposite sides of the spine; see Figure 1(b) for an example. The best known upper bound on the number of required points for a planar L-shaped drawing of every n-vertex top-view caterpillar is $O(n \log n)$; this bound is due to Biedl et al. [2]. We use Theorem 2.1 and improve this bound to $\frac{25}{3}n+O(1)$.

▶ **Theorem 3.1.** *Any top-view caterpillar of n vertices has a planar L-shaped drawing on any set of $\frac{25}{3}n + O(1)$ points in the plane that is in general orthogonal position.*

References

1 Therese Biedl, Ahmad Biniaz, Robert Cummings, Anna Lubiw, Florin Manea, Dirk Nowotka, and Jeffrey Shallit. Rollercoasters and caterpillars. *CoRR*, abs/1801.08565, 2018.

2 Therese Biedl, Timothy M. Chan, Martin Derka, Kshitij Jain, and Anna Lubiw. Improved bounds for drawing trees on fixed points with L-shaped edges. *CoRR*, abs/1709.01456, 2017. Also to appear in *Proceedings of GD, 2017*.

3 Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

4 Emilio Di Giacomo, Fabrizio Frati, Radoslav Fulek, Luca Grilli, and Marcus Krug. Orthogeodesic point-set embedding of trees. *Computational Geometry: Theory and Applications*, 46(8):929–944, 2013.

5 John M. Hammersley. A few seedlings of research. In *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394. University of California Press, 1972.

6 Sergey Kitaev. *Patterns in Permutations and Words*. Springer, 2011.

7 S. Linton, N. Ruškuc, and V. Vatter, editors. *Permutation Patterns*. London Mathematical Society Lecture Note Series, vol. 376, Cambridge, 2010.

8 Manfred Scheucher. Orthogeodesic point set embeddings of outerplanar graphs. Master's thesis, Graz University of Technology, 2015.

9 J. Michael Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In David Aldous, Persi Diaconis, Joel Spencer, and J. Michael Steele, editors, *Discrete Probability and Algorithms*, pages 111–131. Springer New York, 1995.

Altitude Terrain Guarding and Guarding Uni-Monotone Polygons

Stephan Friedrichs[1,2], Valentin Polishchuk[3], and Christiane Schmidt[3]

1 **Max Planck Institute for Informatics, Saarbrücken, Germany**
sfriedri@mpi-inf.mpg.de
2 **Saarbrücken Graduate School of Computer Science, Saarbrücken, Germany**
3 **Communications and Transport Systems, ITN, Linköping University, Sweden,**
{valentin.polishchuk,christiane.schmidt}@liu.se

Abstract

We show that the problem of guarding an x-monotone terrain from an altitude line and the problem of guarding a uni-monotone polygon are equivalent. We present a polynomial time algorithm for both problems, and show that the cardinality of a minimum guard set and the cardinality of a maximum witness set coincide. Thus, uni-monotone polygons are perfect.

1 Introduction

Both the Art Gallery Problem (AGP) and the 1.5D Terrain Guarding Problem (TGP) are well known problems in Computational Geometry. We are given a polygon P (AGP) or an x-monotone chain T of line segments in $\mathbb{R}^2$ (1.5D TGP) and need to place a minimum number of point-shaped guards in P or on T, such that they cover all of P or T, respectively. Both problems have been shown to be NP-hard: Krohn and Nilsson [3] proved the AGP to be hard even for monotone polygons, and King and Krohn [2] established the NP-hardness of both the discrete and the continuous TGP (with guards restricted to the terrain vertices or guards located anywhere on the terrain).

The problem of guarding a uni-monotone polygon (an x-monotone polygon with a single horizontal segment as one of its two chains) and the problem of guarding a terrain with guards placed on a horizontal line above the terrain appear to be problems somewhere between the 1.5D TGP and the AGP in monotone polygons. We show that, surprisingly, both problems allow for a polynomial time algorithm: a simple sweep.

Moreover, we are able to construct a maximum witness set of the same cardinality as the minimum guard set for uni-monotone polygons. Hence, we establish the first non-trivial class of perfect polygons (earlier only proven for "rectilinear"[5] and "staircase" visibility [4]).

One application of guarding a terrain with guards placed on a horizontal line above the terrain, the Altitude Terrain Guarding Problem (ATGP), comes from the idea of using drones to surveil a complete geographical area. Usually, these drones will not be able to fly arbitrarily high, which motivates to cap the allowed height for guards (and without this restriction a single sufficiently high guard above the terrain will be enough). Of course, eventually we are interested in working in two dimensions and a height, the 2.5D ATGP—one dimension and height is a natural starting point for this.

2 Notation and Preliminaries

A polygon P is *x-monotone* if any line orthogonal to the x-axis has a simply connected intersection with P. Its leftmost and rightmost vertex split the boundary in two x-monotone polygonal chains. A *uni-monotone polygon* P is an x-monotone polygon, such that one of

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

its two chains is a single horizontal segment. W.l.o.g. we will assume the single horizontal segment to be the upper chain for the remainder of this paper; we denote this segment by $\mathcal{H}$. The lower chain of P, $LC(P)$, is defined by its *vertices* $V(P) = \{v_1, \ldots, v_n\}$ and has *edges* $E(P) = \{e_1, \ldots, e_{n-1}\}$ with $e_i = \overline{v_i v_{i+1}}$. Due to uni-monotonicity the vertices of P are totally ordered w.r.t. their x-coordinates. A point $p \in P$ *sees* or *covers* $q \in P$ if and only if $\overline{pq}$ is fully contained in P. $\mathcal{V}_P(p)$ is the *visibility polygon* (VP) of p in P with $\mathcal{V}_P(p) := \{q \in P \mid p \text{ sees } q\}$. For $G \subset P$ we abbreviate $\mathcal{V}_P(G) := \bigcup_{g \in G} \mathcal{V}_P(g)$.

A *terrain* T is an x-monotone chain of line segments in $\mathbb{R}^2$ defined by its *vertices* $V(T) = \{v_1, \ldots, v_n\}$ that has *edges* $E(T) = \{e_1, \ldots, e_{n-1}\}$ with $e_i = \overline{v_i v_{i+1}}$; and $\text{int}(e_i) := e_i \setminus \{v_i, v_{i+1}\}$ is e_i's *interior*. Due to monotonicity the points on T are totally ordered w.r.t. their x-coordinates. For $p, q \in T$, we write $p \leq q$ ($p < q$) if p is (strictly) left of q.

An *altitude line* $\mathcal{A}$ at height a_h for a terrain T is a horizontal line located a_h above the lowest (y-)coordinate of all vertices of T, with the leftmost point vertically above v_1 and the rightmost point vertically above v_n. For this abstract we consider only the case where the altitude line lies completely above T. The points on $\mathcal{A}$ are totally ordered as well w.r.t. their x-coordinates, and we adapt the same notation as for two points on T. A point $p \in \mathcal{A}$ *sees* or *covers* $q \in T$ if and only if $\overline{pq}$ is nowhere below T (i.e. $\overline{pq}$ lies on or above T). $\mathcal{V}_T(p)$ is the *visibility region* of p with $\mathcal{V}_T(p) := \{q \in T \mid p \text{ sees } q\}$. For $G \subseteq \mathcal{A}$ we abbreviate $\mathcal{V}_T(G) := \bigcup_{g \in G} \mathcal{V}_T(g)$. We also define the *visibility region* for $p \in T$: $\mathcal{V}_T(p) := \{q \in \mathcal{A} \mid p \text{ sees } q\}$.

For an edge $e \in P$ or $e \in T$ the *strong VP* (*weak VP*) is the set of points that see all of e (at least one point of e): $\mathcal{V}^s_P(e) := \{p \in P : \forall q \in e \; p \text{ sees } q\}$ and $\mathcal{V}^s_T(e) := \{p \in \mathcal{A} : \forall q \in e \; p \text{ sees } q\}$ ($\mathcal{V}^w_P(e) := \{p \in P : \exists q \in e \; p \text{ sees } q\}$ and $\mathcal{V}^w_T(e) := \{p \in \mathcal{A} : \exists q \in e \; p \text{ sees } q\}$).

▶ **Definition 2.1** (Altitude Terrain Guarding Problem)**.** In the *Altitude Terrain Guarding Problem (ATGP)*, abbreviated $\text{ATGP}(T, \mathcal{A})$, we are given a terrain T and an altitude line $\mathcal{A}$. A guard set $G \subset \mathcal{A}$ is optimal w.r.t. $\text{ATGP}(T, \mathcal{A})$ if G is feasible, that is, $T \subseteq \mathcal{V}_T(G)$, and $|G| = \text{OPT}(T, \mathcal{A}) := \min\{|C| \mid C \subseteq \mathcal{A} \text{ is feasible w.r.t. } \text{ATGP}(T, \mathcal{A})\}$.

▶ **Definition 2.2** (Art Gallery Problem)**.** In the *Art Gallery Problem (AGP)*, abbreviated $\text{AGP}(G, W)$, we are given a polygon P and sets of guard candidates and points to cover $G, W \subseteq P$. A guard set $C \subseteq G$ is optimal w.r.t. $\text{AGP}(G, W)$ if C is feasible, that is, $W \subseteq \mathcal{V}_P(C)$, and $|C| = \text{OPT}(G, W) := \min\{|C| \mid C \subseteq G \text{ is feasible w.r.t. } \text{AGP}(G, W)\}$. In general, we want to solve the AGP for $G = P$ and $W = P$, that is, $\text{AGP}(P, P)$.

A set $W \subset P$ ($W \subset T$) is a *witness set* if $\forall\, w_i \neq w_j \in W$ we have $\mathcal{V}_P(w_i) \cap \mathcal{V}_P(w_j) = \emptyset$. A polygon class $\mathcal{P}$ is *perfect* if the cardinality of an optimum guard set and the cardinality of a maximum witness set coincide for all polygons $P \in \mathcal{P}$.

▶ **Lemma 2.3.** *Let P be a uni-monotone polygon, with guard set G. Then there exists a guard set $G^{\mathcal{H}}$ with $|G| = |G^{\mathcal{H}}|$ and $g \in \mathcal{H}\ \forall g \in G^{\mathcal{H}}$. That is, if we want to solve the AGP for a uni-monotone polygon, w.l.o.g. we can restrict our guards to be located on $\mathcal{H}$.*

Proof. Let G be an optimal guard set. Consider a point $p \in P$, there exists a guard $g \in G$ that covers p. Let $g^{\mathcal{H}}$ be the point located vertically above g on $\mathcal{H}$. Because of P being uni-monotone the triangle $\Delta(g, p, g^{\mathcal{H}})$ must be empty, hence, also $g^{\mathcal{H}}$ sees p. ◀

An analogous proof shows that we can always place guards on the altitude line $\mathcal{A}$ even if we would be allowed to place them anywhere between the terrain T and $\mathcal{A}$.

▶ **Lemma 2.4.** *Let P be a uni-monotone polygon, G a guard set with $g \in \mathcal{H}\ \forall g \in G$ that covers $LC(P)$, that is, $LC(P) \subset \mathcal{V}_P(G)$. Then G covers all of P, that is, $P \subseteq \mathcal{V}_P(G)$.*

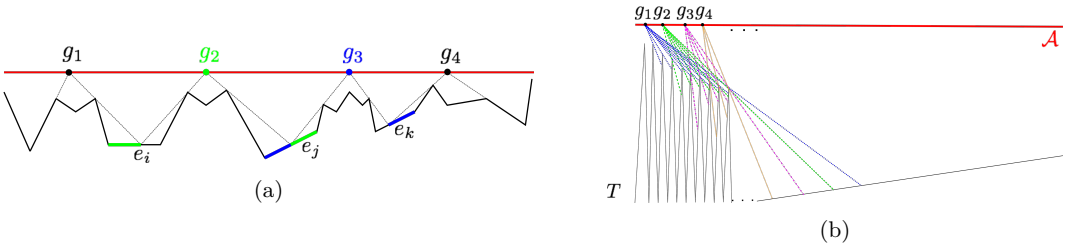


Figure 1 (a) Terrain T and altitude line $\mathcal{A}$ is shown in black and red, resp.. $g_1, \ldots, g_4$ are an optimal guard cover. g_2 and g_3 both cover a critical edge both to their left and to their right. (b) Example: each of the $O(n)$ guards needs to shoot $O(n)$ (colored) rays.

Proof. Assume there is a point $p \in P$, $p \notin LC(P)$ with $p \notin \mathcal{V}_P(G)$. Consider the point p^{LC}, which is located vertically below p on $LC(P)$. Let $g \in G$ be a guard that sees p^{LC}. $LC(P)$ does not intersect the line $\overline{p^{LC}g}$, and because P is uni-monotone the triangle $\Delta(g, p, p^{LC})$ is empty, hence, g sees p; a contradiction. ◀

Consequently, the ATGP and the AGP for uni-monotone polygons are equivalent; we will only refer to the ATGP in the remainder of this paper, with the understanding that all our results can be applied directly to the AGP for uni-monotone polygons.

▶ **Lemma 2.5.** *Let $g \in \mathcal{A}, p \in T, g < p$. If $p \notin \mathcal{V}_T(g)$ then $\forall g' < g, g' \in \mathcal{A} : p \notin \mathcal{V}_T(g')$.*

Before we present our algorithm, we observe that the ATGP is intrinsically different from TGP. We repeat (and extend) a definition from [1]: For a feasible guard cover C of T ($C \subset T$ for TGP and $C \subset \mathcal{A}$ for ATGP), an edge $e \in E$ is *critical w.r.t.* $g \in C$ if $C \setminus \{g\}$ covers some part of, but not all of $\text{int}(e)$. If e is critical w.r.t. some $g \in C$, we call e *critical edge* (e is critical iff more than one guard is responsible for covering its interior). $g \in C$ is a *left-guard (right-guard)* of $e_i \in E$ if $g < v_i$ ($v_{i+1} < g$) and e_i is critical w.r.t. g. We call g a *left-guard (right-guard)* if it is a left-guard (right-guard) of some $e \in E$.

▶ Observation 2.6. For the TGP we have: Let C be finite and cover T, then no $g \in C \setminus V(T)$ is both a left- and a right-guard, that is, no guard that is not on a vertex is responsible to cover critical edges to its left and right, see Friedrichs et al. [1]. However, for the ATGP, any guard g on $\mathcal{A}$ may be responsible to cover critical edges both to its left and to its right, that is, guards may be both a left- and a right-guard, see Figure 1(a).

3 Sweep Algorithm

Our algorithm is a sweep, and informally it can be described as follows (the pseudocode for our algorithm, using definitions from Section 3.1, is presented in Algorithm 1):

- We start with an empty set of guards, $G = \emptyset$, and at the leftmost point of $\mathcal{A}$; all edges $E(T)$ are completely unseen.
- We sweep along $\mathcal{A}$ from left to right and place a guard g_i whenever we could no longer see all of an edge e' if we would move more to the right.
- We compute the visibility polygon of g_i, $\mathcal{V}_T(g_i)$, and for each edge $e = \{v, w\}$ partially seen by g_i, we split the edge, and only keep the open interval that is not yet guarded.
- Thus, whenever we insert a new guard g_i we have a new set of "edges" $E_i(T)$ that are still completely unseen, and $\forall f \in E_i(T)$ we have $f \subseteq e \in E(T)$.
- We continue placing new guards until $T \subseteq \mathcal{V}_T(G)$.
- As we can define a witness set of $|G|$ our guard set is optimal: we place a point witness on e' at the point p we would lose coverage of, if we had not placed guard g_i.

3.1 How to Split the Partly Seen Edges

For each edge in the initial set of edges, $e \in E(T)$, we need to determine the point p_e^c that closes the interval on $\mathcal{A}$ from which all of e is visible. We denote the set of points $p_e^c \; \forall e \in E(T)$ as the set of closing points $\mathcal{C}$, that is, $\mathcal{C} = \cup_{e \in E(T)} \{p_e^c \in \mathcal{A} : (e \subseteq \mathcal{V}_T(p_e^c)) \wedge (e \nsubseteq \mathcal{V}_T(p) \; \forall p > p_e^c, \; p \in \mathcal{A})\}$. The points in $\mathcal{C}$ are the rightmost points on $\mathcal{A}$ in the strong visibility polygon of the edge e, for all edges. Analogously, we define the set of opening points $\mathcal{O}$: $\mathcal{O} = \cup_{e \in E(T)} \{p_e^o \in \mathcal{A} : (e \subseteq \mathcal{V}_T(p_e^o)) \wedge (e \nsubseteq \mathcal{V}_T(p) \; \forall p < p_e^o, \; p \in \mathcal{A})\}$. For each edge e the point in $\mathcal{O}$ is the leftmost point on $\mathcal{A}$ in the strong visibility polygon of e.

Moreover, whenever we place a new guard, we need to split partly seen edges to obtain the new, completely unseen, possibly open, interval, and determine the point on $\mathcal{A}$ where we would lose coverage of this edge (interval). That is, whenever we split an edge we need to add the appropriate point to $\mathcal{C}$.

To be able to easily identify whether an edge e of the terrain needs to be split due to a new guard g, we define the set of "soft openings" $\mathcal{S}$: the leftmost point on $\mathcal{A}$ in the weak visibility polygon of e (if g is to the right of this point—and to the left of the closing point—it can see at least parts of e). We define $\mathcal{S} = \cup_{e \in E(T)} \{p_e^s \in \mathcal{A} : (\exists q \in e, q \in \mathcal{V}_T(p_e^s)) \wedge (\nexists q \in e, \; q \in \mathcal{V}_T(p) \; \forall p < p_e^s, p \in \mathcal{A})\}$.

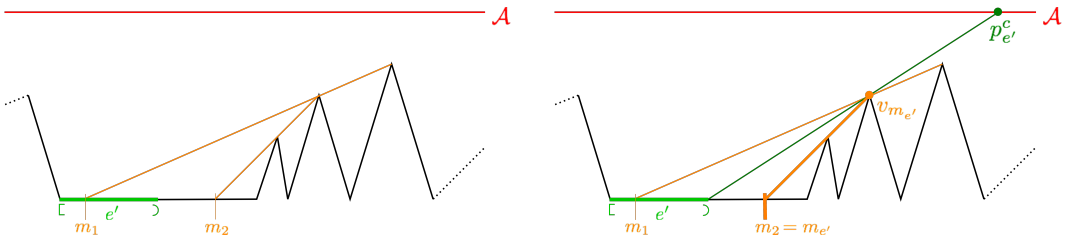
So, how do we preprocess our terrain such that we can easily identify the point on $\mathcal{A}$ that we need to add to $\mathcal{C}$ when we split an edge? We make an initial sweep from the rightmost to the leftmost vertex; for each vertex we shoot a ray to all other vertices to its left and mark the points, *mark points*, where these rays hit the edges of the terrain. This leaves us with $O(n^2)$ preprocessed intervals. For each mark point m we store the rightmost of the two terrain vertices that defined the ray hitting the terrain at m, let this vertex be denoted by v_m. For each edge $e_j = \{v_j, v_{j+1}\}$ with v_{j+1} convex, this includes v_{j+1} as a mark point.

Whenever the placement of a guard g splits an edge e such that the open interval $e' \subset e$ is not yet guarded, see for example Figure 2(a), we identify the mark, $m_{e'}$ to the right of e' and shoot a ray r from the right endpoint of e' through $v_{m_{e'}}$ (the one we stored with $m_{e'}$). The intersection point of r and $\mathcal{A}$ defines our new closing point $p_{e'}^c$, see Figure 2(b).

3.2 Minimum Guard Set and Perfect Polygons

▶ **Lemma 3.1.** *The set G output by Algorithm 1 is feasible, that is, $T \subseteq \mathcal{V}_T(G)$.*

Proof. Assume there is a point $p \in T$ with $p \notin \mathcal{V}_T(G)$. $p \in e$ for some edge $e \in E(T)$. As p is not covered, there exists no guard in G in the interval $[p_e^o, p_e^c]$ on $\mathcal{A}$. Thus, p_e^c is never the



(a) (b)

Figure 2 Terrain T and altitude line $\mathcal{A}$ is shown in black and red, resp.. The orange lines show the rays from the preprocessing step, their intersection points with the terrain define the mark points. Assume the open interval e', shown in light green, is still unseen. To identify the closing point for e' we identify the mark to the right of e', $m_{e'}$, and shoot a ray, shown in dark green, from the right end point of e' through $v_{m_{e'}}$. The intersection point of r and $\mathcal{A}$ defines our new closing point $p_{e'}^c$.

INPUT : Terrain T, altitude line $\mathcal{A}$, its leftmost point a, sets $\mathcal{C}, \mathcal{O}, \mathcal{S}$ of closing, opening, and soft opening points for all edges in T, all ordered from left to right.
OUTPUT: An optimal guard set G.

1 $E_g = E(T)$ // set of edges that still need to be guarded
2 $i := 1$
3 $g_0 := a$ // the point on $\mathcal{A}$ before the first guard is a
4 **while** $E_g \neq \emptyset$ // as long as there are still unseen edges
5 **do**
6 1. Sweep right from g_{i-1} along $\mathcal{A}$ until the first closing point $c \in \mathcal{C}$ is hit
7 2. Place g_i on c, $G = G \cup \{g_i\}$, $i := i + 1$
8 3. **for** *all* $e \in E_g$ // $g_i \not> p_e^c$ by construction
9 **do**
10 **if** $p_e^o \leq g_i \leq p_e^c$ **then**
11 $E_g = E_g \setminus \{e\}$ // if all of e is seen, delete it from E_g
12 $\mathcal{C} = \mathcal{C} \setminus \{p_e^c\}$ // and delete the closing point from the event queue
13 **else if** $g_i < p_e^o$ **then**
14 **if** $p_e^s \leq g_i$ // if g_i can see the right point of e
15 **then**
16 Shoot a visibility ray from g_i onto e, let the intersection point be r_e // all points on e to the right of r_e (incl. r_e) are seen
17 Identify the mark m_e immediately to the right of r_e on e
18 Shoot a ray r from r_e through v_{m_e}
19 Let $p_{e'}^c$ be the intersection point of r and $\mathcal{A}$ // $p_{e'}^c$ is the closing point for the still unseen interval $e' \subset e$
20 $\mathcal{C} = \mathcal{C} \cup \{p_{e'}^c\} \setminus \{p_e^c\}$
21 Sort $\mathcal{C}$
22 $E_g = E_g \cup \{e'\} \setminus \{e\}$

Algorithm 1: Optimal Guard Set for ATGP

event point that defines the placement of a guard in lines 6,7 of Algorithm 1. Moreover, as $\nexists g_i : p_e^o \leq g_i \leq p_e^c$, e is never completely deleted from E_g in lines 10–12. Consequently, for some i we have $g_i < p_e^o$ and $p_e^s \leq g_i$ (lines 14–22). As $p \notin \mathcal{V}_T(G)$, we have $p \in e' \subset e$.

Again, because $p \notin \mathcal{V}_T(G)$, $\nexists g_j \in [p_e^o, p_{e'}^c] \subset \mathcal{A}$, $j \geq i$. Due to line 6 no guard may be placed to the left of $p_{e'}^c$, hence, there is no guard placed in $[p_e^o, b]$ (b being the right end point of $\mathcal{A}$). So, e' is never deleted from E_g, a contradiction to G being the output of Alg. 1. ◀

▶ **Theorem 3.2.** *The set G output by Algorithm 1 is optimal.*

Proof. To show that G is optimal, we need to show that G is feasible and that G is minimum. Feasibility follows from Lemma 3.1, it remains to show that G is minimum. Given a witness set W and a guard set G, $|W| \leq |G|$ holds. Thus, if we can find a witness set W with $|W| = |G|$, we can show that G is minimum. We will place a witness for each guard Algorithm 1 places. First, we need an auxiliary lemma (and omit the proof):

▶ **Lemma 3.3.** *Let $c \in \mathcal{C}$ be the closing point for a complete edge (and not just an edge interval) in line 6 of Algorithm 1 that enforces the placement of a guard g_i. Then there exists an edge $e_j = \{v_j, v_{j+1}\} \in E(T)$ for which c is the closing point, such that v_{j+1} is a reflex vertex, and v_j is a convex vertex.*

Now we can define our witness set: we consider the edges or edge intervals, which define the closing point $c \in \mathcal{C}$ that leads to a placement of guard g_i in lines 6,7 of Algorithm 1.

If c is defined by some complete edge $e_j \in E(T)$, let $E_c \subseteq E_g$ be the set of edges for which c is the closing point. We pick the rightmost edge $e_j \in E_c$ such that v_j is a convex vertex and v_{j+1} is a reflex vertex, which exists by Lemma 3.3, and choose $w_i = v_j$.

Otherwise, that is, if c is only defined by edge intervals, we pick the rightmost such edge interval $e' \subset e_j$. Then $e' = [v_j, q)$ for some point $q \in e_j, q \neq v_{j+1}$, and we place a witness at q^ε, a point ε_i to the left of q on T: $w_i = q^\varepsilon$. We define $W = \cup_{i=1}^{|G|} w_i$. By definition $|W| = |G|$, and we still need to show that W is indeed a witness set.

Let S_i be the strip of all points with x-coordinates between $x(g_{i-1}) + \varepsilon$ and $x(g_i)$. Let p_T and $p_{\mathcal{A}}$ be the vertical projection of a point p onto T and $\mathcal{A}$, respectively. $S_i = \{p \in \mathbb{R}^2 : (x(g_{i-1}) + \varepsilon \leq x(p) \leq x(g_i)) \wedge (y(p_T) \leq y(p) \leq y(p_{\mathcal{A}}))\}$. We show that $\mathcal{V}_T(w_i) \subseteq S_i \forall i$, hence, $\mathcal{V}_T(w_k) \cap \mathcal{V}_T(w_\ell) = \emptyset \; \forall w_k \neq w_\ell \in W$, which shows that W is a witness set.

If $w_i = v_j$ for an edge $e_j \in E(T)$, $\mathcal{V}_T(w_i)$ contains the guard g_i, but no other guard: If g_{i-1} could see v_j, we would have $\angle(g_{i-1}, v_j, v_j + 1) \leq 180°$ because v_j is a convex vertex, thus, g_{i-1} could see all of e_j, a contradiction to $e_j \in E_g$. Moreover, assume w_i could see some point p with $x(p) \leq x(g_{i-1})$. The terrain does not intersect the line $\overline{w_i p}$, and because the terrain is monotone the triangle $\Delta(w_i, p, g_{i-1})$ would be empty, a contradiction to g_{i-1} not seeing w_i. Hence, no guard $g_j, j < i$ sees w_i; a similar argument can be given for $g_j, j > i$.

If $w_i = q^{\varepsilon}$ for $e' = [v_j, q)$, $\mathcal{V}_T(w_i)$ contains the guard g_i, but no other guard: If g_{i-1} could see w_i, q would not be the endpoint of the edge interval, a contradiction. Moreover, assume w_i could see some point p with $x(p) \leq x(g_{i-1})$. T does not intersect the line $\overline{w_i p}$, and because T is monotone the triangle $\Delta(w_i, p, g_{i-1})$ would be empty, a contradiction. Thus, again, no guard $g_j, j < i$ sees p (and the case $j > i$ can be shown similarly). ◀

We showed that there exists a maximum witness set $W \subset T$ and a minimum guard set $G \subset \mathcal{A}$ with $|W| = |G|$. By Lemma 2.3 and 2.4 the ATGP and the AGP for uni-monotone polygons are equivalent. Thus, for a uni-monotone polygon P we can find a maximum witness set $W \subset LC(P) \subset P$ and a minimum guard set $G \subset \mathcal{H} \subset P$ with $|W| = |G|$:

▶ **Theorem 3.4.** *Uni-monotone polygons are perfect.*

3.3 Algorithm Runtime

The preprocessing step to compute the mark points costs $O(n^2)$, based on these we can compute the closing points for all edges of the terrain. Similarly, we compute the mark points from the left to compute the opening points (using the left vertex of an edge to shoot the ray) and the soft opening points (using the right vertex of an edge to shoot the ray). Then, whenever we insert a guard (of which we might add $O(n)$), we need to shoot $O(n)$ rays, see Figure 1(b), which altogether costs $O(n^2 \log n)$. Similarly, for each of the intersection points r_e, we need to shoot a ray through v_{m_e}. This gives a total runtime of $O(n^2 \log n)$. In fact, when we stepwise build the convex hull of the terrain vertices from the right and only shoot rays through vertices of this CH, we can reduce the preprocessing step to $O(n)$.

References

1 S. Friedrichs, M. Hemmer, J. King, and C. Schmidt. The continuous 1.5D terrain guarding problem: Discretization, optimal solutions, and PTAS. *JoCG*, 7(1):256–284, 2016.

2 J. King and E. Krohn. Terrain guarding is NP-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.

3 E. Krohn and B. J. Nilsson. The complexity of guarding monotone polygons. In *Proc. of the 24th Canadian Conference on Comp. Geometry*, pages 167–172, 2012.

4 R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *J. Comput. Syst. Sci.*, 40(1):19–48, 1990.

5 C. Worman and J. M. Keil. Polygon decomposition and the orthogonal art gallery problem. *Int. J. Comput. Geometry Appl.*, 17(2):105–138, 2007.

On Merging Straight Skeletons

Franz Aurenhammer[1] and Michael Steinkogler[2]

1 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria**
auren@igi.tugraz.at

2 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria**
steinkogler@igi.tugraz.at

Abstract

The search for efficient algorithms to compute the straight skeleton of a simple polygon has resulted in a variety of algorithms. We present a new approach that applies the divide-and-conquer paradigm with the divide step based on the motorcycle graph. A practical randomized algorithm is obtained that derives the straight skeleton from the motorcycle graph, with an expected running time of $\mathcal{O}(d\,n \log n)$, where d is the decomposition depth of the motorcycle graph.

1 Introduction

The straight skeleton of a simple polygon was introduced to computational geometry about two decades ago in [1]. It is defined as the trace of the vertices as the polygon shrinks by moving its edges in a self-parallel manner towards the interior of the polygon. During this offsetting process edges disappear (so-called edge events, see the dotted offset in Figure 1 where the edge e disappears), and reflex vertices may run into other edges (so-called split events, see the dashed offset in Figure 1 where the vertex r runs into some edge and splits it). For more detailed information see, for example, the section on straight skeletons in [2].

Initially, a simple priority queue algorithm with a running time of $\mathcal{O}(n^2 \log n)$ has been proposed, simulating the shrinking process by computing all edge and split events. A theoretical breakthrough was the first sub-quadratic algorithm, by Eppstein and Erickson in [5] who also introduce the motorcycle graph. This graph consists of straight traces of 'motorcycles' that start at reflex vertices, with the speed and direction of the reflex vertex during the shrinking process; see Figure 2 for an example. A motorcycle's trace stops when it hits either the (already existing) trace of another motorcycle or the boundary of the polygon (as in [3] we assume that no two motorcycles collide). It seems that the motorcycle graph encodes essential information needed for the construction of the straight skeleton; several of the more recent algorithms for the straight skeleton build upon it. However, even the

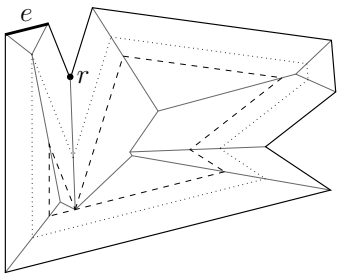


Figure 1 Straight skeleton offsetting process with edge and split events.

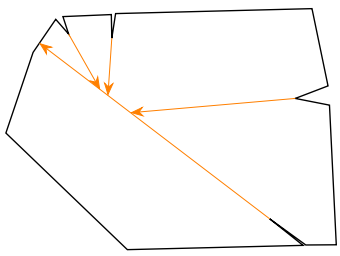


Figure 2 The motorcycle graph partitions a nonconvex polygon into convex cells.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

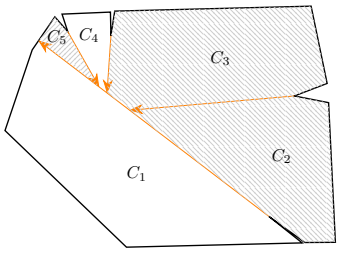


Figure 3 Motorcycle regions C_5 (motorcycle cell) and $C_2 \cup C_3$ (union of two regions).

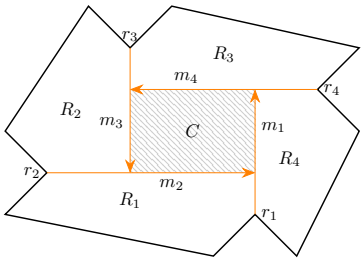


Figure 4 An inner motorcycle cell C, bounded by a cycle of dominant motorcycles.

derivation of the straight skeleton from the motorcycle graph has remained a complicated task being hard to implement. Also, the computation of the motorcycle graph itself is a challenging problem (the best known algorithm has a running time of $\mathcal{O}(n^{4/3+\epsilon})$ [6]).

The global nature of split events complicates the design of efficient *divide-and-conquer* algorithms for the straight skeleton. It took almost two decades (and several failing attempts) until the first correct and efficient algorithm of this type was published, in Cheng et al. [3]. Interestingly, this algorithm shows the currently best theoretical running time, $\mathcal{O}(n \log n \log r + r^{4/3+\varepsilon})$, for a polygon with n edges and r reflex vertices.

Here and later on, let $\mathcal{P}$ denote a simple n-vertex polygon, $\mathcal{M}(\mathcal{P})$ its motorcycle graph, and $\mathcal{S}(\mathcal{P})$ its straight skeleton (or just skeleton for brevity). The line segments forming $\mathcal{S}(\mathcal{P})$ will be called arcs.

In this note, we present a very simple divide-and-conquer algorithm that computes $\mathcal{S}(\mathcal{P})$ once $\mathcal{M}(\mathcal{P})$ is given. The idea is to divide $\mathcal{P}$ into cells according to $\mathcal{M}(\mathcal{P})$, to compute the skeletons of the motorcycle cells separately, and then merge them into $\mathcal{S}(\mathcal{P})$. We start in Section 2 by defining motorcycle regions and related concepts. The skeletons of regions are the topic of Section 3, in particular the relationship between the skeleton of a region and the skeletons of its subregions. The results from Section 3 are put to use in Section 4 for a divide-and-conquer algorithm that computes $\mathcal{S}(\mathcal{P})$. The divide step is trivial provided $\mathcal{M}(\mathcal{P})$ is available. The conquer step is as simple as Chew's [4] incremental method for the medial axis of a convex polygon. We first describe the algorithm for motorcycle graphs that are free of cycles. General motorcycle graphs are handled in Section 5, along with runtime considerations depending on the structure of these graphs. If the structure of $\mathcal{M}(\mathcal{P})$ is nice enough, the expected running time is $\mathcal{O}(n \log^2 n)$, while in the general case it is $\mathcal{O}(n^2 \log n)$.

2 Motorcycle regions

The motorcycle graph $\mathcal{M}(\mathcal{P})$ partitions $\mathcal{P}$ into polygons called *cells*, which are convex because a motorcycle egde emanates from each reflex vertex of $\mathcal{P}$, bisecting its interior angle. We first assume that each cell is supported by edges of $\mathcal{P}$. (Inner motorcycle cells, which are caused by cycles in $\mathcal{M}(\mathcal{P})$, will be handled later.) To combine motorcycle cells we introduce motorcycle regions. A *motorcycle region*, R, of $\mathcal{M}(\mathcal{P})$ is either a motorcycle cell defined by $\mathcal{M}(\mathcal{P})$, or $R = R_1 \cup R_2$ where R_1 and R_2 are regions sharing some motorcycle edge. See Figure 3 for an illustration of both kinds of regions.

From now on let $R = R_1 \cup R_2$ be a region of $\mathcal{M}(\mathcal{P})$, let m be the common motorcycle edge of R_1 and R_2, and denote with r the reflex vertex that defines m. To associate R with a skeleton suitable for future merging steps, a so-called *region polygon* for R (which is a superset of R and actually generates the desired skeleton) is needed. Following the recursive

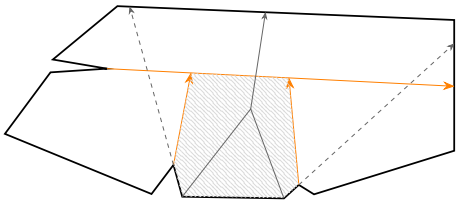


Figure 5 Motorcycle cell polygon (dashed, open) using only the cell's polygon edges.

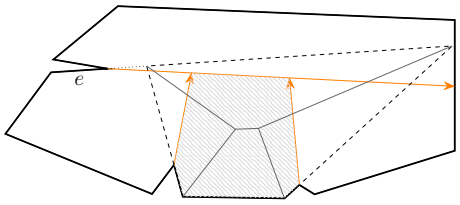


Figure 6 Motorcycle cell polygon (dashed) with the dominating edge e.

definition of motorcycle regions, we will define the region polygon for a motorcycle cell first, and then show how the region polygons for R_1 and R_2 are combined to obtain the region polygon for R.

Consider some motorcycle cell C. Each edge of C is either supported by an edge of $\mathcal{P}$ or by an edge of $\mathcal{M}(\mathcal{P})$. The former type of edges could be used to construct a (possibly unbounded) convex polygon by extending them until adjacent edges intersect; see Figure 5 for an example. However, merging two such polygons can create a polygon whose skeleton contains an arc *longer* than its motorcycle edge in $\mathcal{M}(\mathcal{P})$. This leads to complications during the merging process.

The solution is to also add edges of $\mathcal{P}$ that define motorcycle edges supporting C. Let m be a motorcycle edge supporting C, and let r be its reflex vertex in $\mathcal{P}$. The edge e of $\mathcal{P}$ incident to r and on the same side of m as C is called a *dominating edge* of C. Combining all dominating edges of C with all edges of $\mathcal{P}$ that support C still results in a convex polygon, the *motorcycle cell polygon*; see Figure 6 for an example. Note that the combined size of all motorcycle cell polygons is $\mathcal{O}(n)$.

In the following, let $\mathcal{R}$, $\mathcal{R}_1$ and $\mathcal{R}_2$ be the region polygons of R, R_1 and R_2, respectively. To obtain $\mathcal{R}$, the polygons $\mathcal{R}_1$ and $\mathcal{R}_2$ need to be merged at both endpoints, say r and v, of m. At the reflex vertex r this is done by simply truncating the respective edges of $\mathcal{R}_1$ and $\mathcal{R}_2$. Concerning v, this endpoint either lies on a polygon edge (which, therefore, is already part of the subregions' polygons), or on another motorcycle edge m' that dominates m, and thus the corresponding dominating edge is already part of the subregions' polygons. As a consequence, the merging of $\mathcal{R}_1$ and $\mathcal{R}_2$ at v is simple as well. Note that r is the only new reflex vertex that gets introduced by the merge.

3 Straight skeletons of motorcycle regions

The skeleton of a region R of $\mathcal{M}(\mathcal{P})$ is defined as the part of the skeleton of R's region polygon that lies within R. That is, $\mathcal{S}(R) = \mathcal{S}(\mathcal{R}) \cap R$.

As an important property of $\mathcal{S}(\mathcal{R})$, the skeleton arcs of reflex vertices are contained in R. Therefore, such skeleton arcs cannot cross region boundaries during the merge process.

▶ **Lemma 3.1.** *Let u be the arc of a reflex vertex r in $\mathcal{S}(\mathcal{R})$. Then $u \subset R$.*

Proof. Vertex r is part of R, therefore its motorcycle edge m is completely contained in R. If m hits the boundary of $\mathcal{P}$, then the hit edge of $\mathcal{P}$ is also part of R, and thus the arc u is contained in R. Assume now that m is blocked by another motorcycle edge m'. The motorcycle cells that have parts of both m and m' on their boundary are part of R. Therefore the associated dominating edge of m' with respect to these cells contributes to the boundary of $\mathcal{R}$. Thus u cannot cross m', and $u \subset R$ again. ◀

Let us now look in more detail at the straight skeleton within a single motorcycle cell C. We say that an edge e of $\mathcal{P}$ is *relevant* for C, if the unique face $f_{\mathcal{P}}(e)$ that e defines in $\mathcal{S}(\mathcal{P})$ has a nonempty intersection with C.

▶ **Theorem 3.2.** *The relevant edges for C (extended if necessary) form a convex polygon in the cyclic order given by $\mathcal{P}$.*

Proof. Clearly, the edges of $\mathcal{P}$ that support C are all relevant for C, and they form a convex polygon. All other edges relevant for C must cross a motorcycle edge during the shrinking process to have a part of C in their face in $\mathcal{S}(\mathcal{P})$. Therefore, they must form convex angles with the adjacent edges in the polygon formed by all relevant edges; the arcs of reflex vertices are shorter than their corresponding motorcycle edges. ◀

▶ **Corollary 3.3.** *Let e be an edge of $\mathcal{R}_2$ that is not an edge of $\mathcal{R}_1$, and such that its face $f_R(e)$ in the straight skeleton of $R = R_1 \cup R_2$ has a nonempty intersection with R_1. Then e forms convex angles when inserted into $\mathcal{R}_1$.*

Proof. For each motorcycle cell $C \subseteq R_1$ with $f_R(e) \cap C \neq \emptyset$, we know from Theorem 3.2 that e is in *convex position* with C's motorcycle cell polygon, that is, e cuts off a single convex vertex from this polygon. Therefore e forms convex angles with its adjacent edges in $\mathcal{R}_1$. ◀

The straight skeleton behaves nicely when inserting an edge in convex position. During the shrinking process, the new edge is always ahead of the parts of the adjacent edges that were cut off. All other edges remain unchanged. Thus the skeleton faces of old edges can shrink but never expand.

For the merge of $\mathcal{S}(\mathcal{R}_1)$ and $\mathcal{S}(\mathcal{R}_2)$ we need to find the edges from one region that can influence the skeleton of the other region within $\mathcal{S}(\mathcal{R})$. The following lemma gives a necessary condition for such edges.

▶ **Lemma 3.4.** *Only edges of $\mathcal{R}_2$ whose faces in $\mathcal{S}(R_2)$ are bounded by the common motorcycle edge m can change $\mathcal{S}(\mathcal{R}_1)$ within $\mathcal{S}(R)$, the merged skeleton.*

Proof. Let e be an edge of $\mathcal{R}_2$ whose face $f_R(e)$ in $\mathcal{S}(R)$ has a nonempty intersection with R_1. Then $f_R(e)$ must be intersected by m, since e is on the opposite side of m. Edges of R_1 with faces that have nonempty intersection with R_2 in $\mathcal{S}(R)$ are in convex position with respect to $\mathcal{R}_2$, by Theorem 3.3. Consequently, e's face in $\mathcal{S}(R_2)$ can only shrink; more precisely, $f_R(e) \cap R_2 \subseteq f_{R_2}(e)$. But this implies that $f_{R_2}(e)$ is bounded by m. ◀

The following related lemma is stated without proof, due to lack of space.

▶ **Lemma 3.5.** *Let E be the (cyclically ordered) set of edges of $\mathcal{R}_2$ whose faces in $\mathcal{S}(R_2)$ are bounded by m, excluding the edge adjacent to m's reflex vertex r in R_2. Then E forms a convex chain, and its edges form convex angles when inserted into $\mathcal{R}_1$.*

4 Divide-and-conquer

We are now ready to describe our divide-and-conquer algorithm. In the divide part, we use the motorcycle graph $\mathcal{M}(\mathcal{P})$ to recursively divide $\mathcal{P}$ along motorcycle edges. Since we assumed $\mathcal{M}(\mathcal{P})$ to have no cycles, there exists a motorcycle edge m that hits the boundary of $\mathcal{P}$. This divides $\mathcal{P}$ into two polygons, P_1 and P_2. This dividing step can be repeated, using motorcycle edges m_1 and m_2 that split P_1 and P_2 into two parts, respectively, and is

iterated until all motorcycle edges have been used and the final polygons are the motorcycle cells of $\mathcal{M}(\mathcal{P})$. The division process can be represented in a tree, which inherits the future merge plan and is therefore called the *merge tree*.

In the conquer part of our algorithm, the results from the previous section are put to use. Suppose $\mathcal{S}(\mathcal{R}_1)$ and $\mathcal{S}(\mathcal{R}_2)$ have already been computed, and need to be merged into $\mathcal{S}(\mathcal{R})$. For $\mathcal{S}(R_1)$, all edges with faces bounded by m are computed. By Lemma 3.5, these edges form a convex chain E_1 that forms convex angles when inserted into $\mathcal{R}_2$. Inserting E_1 into $\mathcal{S}(\mathcal{R}_2)$ results in $\mathcal{S}(\mathcal{R}_2 \cup E_1)$. The same is done with the roles of R_1 and R_2 exchanged, to obtain $\mathcal{S}(\mathcal{R}_1 \cup E_2)$. Note that both skeletons coincide along m, and only need to be glued together along m to obtain $\mathcal{S}(\mathcal{R})$.

Still missing is the part of the algorithm that updates a straight skeleton during the insertion of a convex chain E. We adapt the approach from Chew [4] that computes the medial axis of a convex polygon (which equals its straight skeleton). We insert the edges of E in random order, with each update taking time proportional to the number of arcs of the inserted edge's face as its boundary is traced out.

Suppose an edge e in convex position is inserted into a region polygon and its skeleton needs to be updated. The expected number of arcs defined by e and other (already inserted) edges from E is still constant (here Chew's analysis still applies), but the expected number of arcs defined by e and edges from the original polygon (so-called *mixed arcs*) is not constant. Let l be the size of the original polygon, k the size of E, and $n = l+k$. Suppose that $i-1$ edges have already been inserted and the skeleton of the resulting polygon computed. Inserting the i^{th} edge may cause the construction of new mixed arcs. The maximal number of mixed arcs after the insertion is $l+i-1$, and since the insertion order is randomized, the expected number of mixed arcs supported by the new edge is $(l+i-1)/i$. Summing over all insertions gives the expected total number of mixed arcs when E is inserted: $\sum_{i=1}^{k}(l+i-1)/i = \mathcal{O}(n \log n)$.

As a conseqence, the expected running time for all merges on a single level of the merge tree is $\mathcal{O}(n \log n)$. The running time of the complete algorithm depends on the depth of the merge tree and we get the following theorem:

▶ **Theorem 4.1.** *Let d be the height of the merge tree. Then the straight skeleton of $\mathcal{P}$ can be computed in $\mathcal{O}(d\ n \log n)$ expected time.*

5 General motorcycle graphs and runtime considerations

Now we lift the restriction that the motorcycle graph must be free of cycles. A motorcycle graph cycle is created by a cycle of dominating motorcycles bounding an inner region (see Figure 4). Merges cannot be performed as above, since no two regions share a complete motorcycle edge. However, it is possible to first compute the skeleton of the inner cell, and use the known merge procedure for the skeletons of the outer regions.

Let C be an inner motorcycle cell, bounded by the cycle of dominating motorcycle edges $m_1, \ldots, m_k$. Let R_i denote the region bounded by m_i and m_{i+1} (indices modulo k).

Now consider an edge e of $\mathcal{R}_i$, such that its face $f_{\mathcal{P}}(e)$ in $\mathcal{S}(\mathcal{P})$ has a nonempty intersection with C. Then the face of e in $\mathcal{S}(R_i)$ must be bounded by m_{i+1}, giving a necessary condition for an edge to have a face in $\mathcal{S}(\mathcal{P})$ intersecting with C. It can be shown that all such edges form a convex polygon, enabling the computation of $\mathcal{S}(C)$ in linear time.

Having computed $\mathcal{S}(C)$, it is now possible to detect the edges that sweep over C during the shrinking process, and thus need to be included in the skeletons of the outer regions adjacent to C. Merging all these edges into the skeletons of the appropriate regions R_i can be done in overall $\mathcal{O}(l \log l)$ expected time, with l being the number of edges involved.

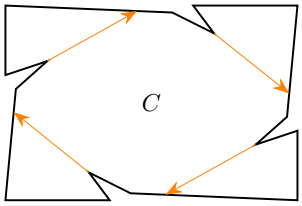


Figure 7 All ears are merged into C.

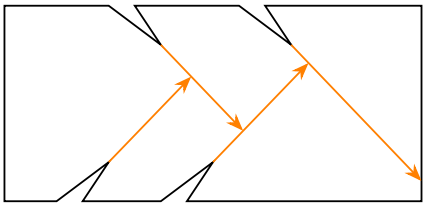


Figure 8 The merge tree degenerates to a path.

Finally, the updated skeletons of the outer regions are combined. First, these skeletons can be restricted to their region (remember that the skeleton of C does not change). Then they can be merged using the original merge procedure along a common motorcycle edge. Balanced binary merges can be used to get an overall expected running time of $\mathcal{O}(n \log^2 n)$.

Whereas motorcycle graphs with cycles can be integrated into the divide step without causing the running time to increase, the dependency on the height of the merge tree (Theorem 4.1) can cause an expected running time of $\mathcal{O}(n^2 \log n)$. There are two kinds of motorcycle graph structures producing merge trees with linear height. The first one results from a 'central' motorcycle cell C with linearly many adjacent cells that need to be merged with C as they have no common motorcycle edge with another cell (see Figure 7 for an example). The second kind occurs when for linearly many motorcycles $m_1, \ldots, m_k$, the motorcycle m_i crashes into the trace of m_{i+1}, and m_k hits the polygon boundary as in Figure 8. For both structures, there are efficient solutions once they have been identified in the motorcycle graph. However, detecting these structures efficiently is an open problem.

6 Conclusions

We have presented a simple and easy-to-implement divide-and-conquer algorithm that derives the straight skeleton of a polygon from its motorcycle graph. The running time depends on the structure of the motorcycle graph, and is expected $\mathcal{O}(n \log^2 n)$ if the motorcycle graph is reasonably 'balanced', competing with the best known algorithm [3]. Still, it seems to be a long way till such a running time may be achieved for constructing the straight skeleton from scratch, as precomputing the motorcycle graph is believed to be the hardest part.

References

1 Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A novel type of skeleton for polygons. In *J.UCS The Journal of Universal Computer Science*, pages 752–761. Springer, 1996.

2 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Co., Inc., 2013.

3 Siu-Wing Cheng, Liam Mencel, and Antoine Vigneron. A faster algorithm for computing straight skeletons. In *European Symposium on Algorithms*, pages 272–283. Springer, 2014.

4 L. Paul Chew. Building voronoi diagrams for convex polygons in linear expected time. Technical report, 1990.

5 David Eppstein and Jeff Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete & Computational Geometry*, 22(4):569–592, 1999.

6 Antoine Vigneron and Lie Yan. A faster algorithm for computing motorcycle graphs. *Discrete & Computational Geometry*, 52(3):492–514, 2014.

The k-Fréchet distance of polygonal curves

Maike Buchin[1] **and Leonie Ryvkin**[2]

1 **Faculty of Computer Science, TU Dortmund,** maike.buchin@tu-dortmund.de

2 **Department of Mathematics, Ruhr University Bochum,** leonie.ryvkin@rub.de

Abstract

We introduce a new distance measure for comparing polygonal chains: the k-Fréchet distance. As the name implies it is closely related to the well-studied Fréchet distance but allows to find similarities between curves that resemble each other only piecewise. As we will explain it provides a nice transition between (weak) Fréchet distance and Hausdorff distance. We prove NP-completeness for the k-Fréchet distance of polygonal curves in different variants and furthermore APX-completeness for the optimization version of our problem, discuss algorithmic approaches and present open questions.

1 Introduction

During the past decades several methods for comparing geometrical shapes have been studied in a variety of applications, for example analysing geographic data, such as trajectories, or comparing chemical structures, e.g. protein chains or human DNA.

The Fréchet distance has been well-studied in the past decades since it has proven to be very helpful in applications such as the above mentioned geographic data analysis or computer aided design. The Hausdorff distance, another similarity measure, has also proven to be useful in applications and can be computed more efficiently than the Fréchet distance. However, it provides us with less information by taking only the overall shape of curves into consideration, not how they are traversed.

We introduce the k-Fréchet distance as a distance measure in between Hausdorff and (weak) Fréchet distance. The k-Fréchet distance allows to compare shapes consisting of several parts by cutting a curve into a number of subcurves where the subcurves resemble each other in terms of the (weak) Fréchet distance. Therefore it allows to find similarities between objects of rearranged pieces such as chemical structures or handwritten characters and symbols.

Characterizing these distance measures in the free space shows that the k-Fréchet distance bridges between the (weak) Fréchet distance and Hausdorff distance (see below for details): the weak Fréchet distance can be characterized by one component in the free space projecting surjectively onto both parameter spaces, and the Hausdorff distance can be characterized by all components of the free space projecting surjectively onto both parameters. For the k-Fréchet distance we ask that k components of the free space project surjectively onto both parameter spaces.

This paper is organized as follows: first we recall some necessary definitions and formally define the k-Fréchet distance. In Section 3 we prove NP- and APX-completeness of the k-Fréchet problem. In Section 4 we present an approximation algorithm of factor 2 as well as an exact algorithm which runs in polynomial time for fixed k. We conclude the paper with open questions.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2 Definitions

Recall the Fréchet distance [1], a well-known measure for curves, which is defined as follows: For curves $P, Q\colon [0,1] \to [0,1]$ the Fréchet distance is given by

$$\delta_F(P,Q) = \inf_{\sigma} \max_{t\in[0,1]} \|P(t) - Q(\sigma(t))\|,$$

where the reparametrisations $\sigma\colon [0,1] \to [0,1]$ range over all orientation-preserving homeomorphisms. A variant is the weak Fréchet distance δ_{wF}, where both curves are reparameterised by σ and τ, respectively, which range over all continuous surjective functions.

A well-known characterisation which is key to efficient algorithms for computing both weak and (strong) Fréchet distance [1] uses the free space diagram. First we recall the free space F_ε:

$$F_\varepsilon(P,Q) = \{(t_1,t_2) \in [0,1]^2\colon \|P(t_1) - Q(t_2)\| \leqslant \varepsilon\}.$$

The free space diagram puts this information into a $(n \times m)$-grid, where n and m are the number of segments in P and Q, respectively.

The Fréchet distance of two curves is at most a given value ε if there exists a monotone path through the free space connecting the bottom left to the top right corner. For the weak Fréchet distance to equal at most ε such a path need not be monotone. The Hausdorff distance δ_H can be characterised as the free space projecting surjectively onto both parameter spaces.

We define the k-Fréchet distance δ_{kF} as in between Hausdorff and weak Fréchet distance using only a constant number k of components in the free space to project onto both parameter spaces:

$$\delta_{kF}(P,Q) = \inf_{\sigma,\tau} \max_{t\in[0,1]} \|P(\sigma(t)) - Q(\tau(t))\|,$$

where now $\sigma, \tau\colon [0,1] \to [0,1]$ range over all surjective functions which are piecewise defined, allowing at most $k-1$ jump discontinouities, such that the images of the continuous parts partition the curve. That is, we cut the curves P and Q into at most k pieces or subcurves such that two resembling subcurves have small weak Fréchet distance. For the decision version of the problem, we ask whether the weak Fréchet distance between pieces can be bounded a given value ε (note that k is a fixed upper bound of cuts here). Naturally, for a fixed distance ε, we would like to cut the curves into as few subcurves as possible (optimization version).

We will also consider the variant where we use (strong) Fréchet distance instead of weak Fréchet distance for the subcurves, that is the reparameterizations σ, τ have to be piecewise homeomorphisms. However, as the weak Fréchet distance (arguably) results in the more natural definition for the k-Fréchet distance we use it as the standard variant.

By definition k-Fréchet distance lies in between Hausdorff and (weak) Fréchet distance

$$\delta_H(P,Q) \leq \delta_{kF}(P,Q) \leq \delta_F(P,Q).$$

Also, the k-Fréchet distance decreases as k increases, and for $k = 1$ it equals weak Fréchet distance, whereas for $k \geq n^2$ it equals Hausdorff distance.

3 NP-Completeness

▶ **Theorem 3.1.** *Deciding wether $\delta_{kF}(P,Q) \leq \varepsilon$ for fixed ε and k is NP-complete. Finding the minimum number of cuts (or components) k is NP-complete as well.*

Proof. First note that we can easily verify a given solution to our problem: we compute the free space diagram of P and Q and identify the free space components resembling mappings of the respective subcurves. Next we need to check whether the union of these components projects surjectively onto both parameter spaces. This can be done in polynomial time, therefore we have k-Fréchet $\in$ NP.

To show NP-hardness of our problem we reduce from the Minimum Common String Partition (MCSP)-Problem. Let us first recall the MCSP-Problem [7]: Given two strings A and B we want to partition them into substrings such that $A = A_1A_2 \dots A_n$ and $B = B_1B_2 \dots B_n$ where for all $i = 1, \dots, n$ holds that $A_i = B_j$ for some $j \in \{1, \dots, n\}$.

There are several variants of MCSP, such as k-MCSP where each letter occurs at most k times in each string (which was proven to be an NP-hard optimization problem in [5]) or MCSP^c where there are at most c elements in the alphabet of the strings (an NP-hard problem even for the decision version, presented in [4]). Both 2-MCSP and MCSP^2 have been proven to be NP-hard (therefore MCSP is NP-hard as well), the first variant is even APX-hard ([5], [7]).

Now, for an instance of MCSP we construct curves P and Q as follows: first we subdivide the unit interval into c intervals of equal size, where c denotes the number of different letters in both strings (which is bounded since we consider finite strings). We then identify every letter of the alphabet Σ with a number in $\{1, \dots, c\}$. Next we transform string A into a curve P and string B into curve Q: for every letter l in a string we form a polygonal chain consisting of five segments: the first connecting $(\frac{1}{2}, 0)$ to $(\frac{1}{2}, \frac{l}{c})$ vertically, a second connecting $(\frac{1}{2}, \frac{l}{c})$ to $(0, \frac{l}{c})$ horizontally, followed by $(0, \frac{l}{c})$ to $(1, \frac{l}{c})$, back to $(\frac{1}{2}, \frac{l}{c})$ and finally back to $(0, \frac{l}{c})$. The result is a T-shape, as shown in Figure 1.

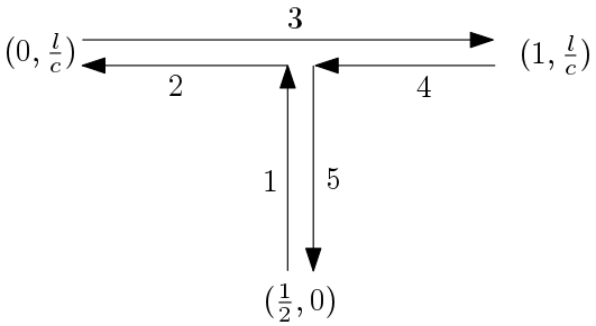


Figure 1 A T-shaped curve fragment resembling the letter l in a string

Since beginning and ending of each T-shaped curve fragment is the same, the fragments can be concatenated. In this manner we produce curves consisting of as many T-shapes as there are letters in the strings, both curves lying on top of each other. We can choose ε to be any number smaller than $1/c$, e.g. $1/2c$.

The T-shapes only differ by length of their vertical line segments, and each length corresponds to a certain letter of the respective string. By choosing $\varepsilon < 1/c$, T-shapes resembling different letters cannot be matched. Hence our construction leads to the following result for all pairs of subcurves:

$$\forall P_i, Q_j \colon P_i = Q_j \Leftrightarrow \delta_{wF}(P_i, Q_i) \leqslant \varepsilon.$$

In fact we even have $\delta_{wF}(P_i, Q_i) = 0 \leqslant \varepsilon$, since we place Q on top of P and construct identical concatenations of T-shaped subcurves for equal substrings. Because all letters start

in $(\frac{1}{2}, 0)$, we get that two substrings are equal iff the respective subcurves have weak Fréchet distance less than ε.

In the free space the following picture arises: we get components in the free space of P and Q resembling equal substrings of A and B. Hence, the longer identical substrings, the larger their resembling component in the free space. Therefore finding a minimum collection of k components projecting surjectively onto the parameter spaces equals finding a minimum number of blocks where each block of A equals a block in B. See Figure 2 for an example.

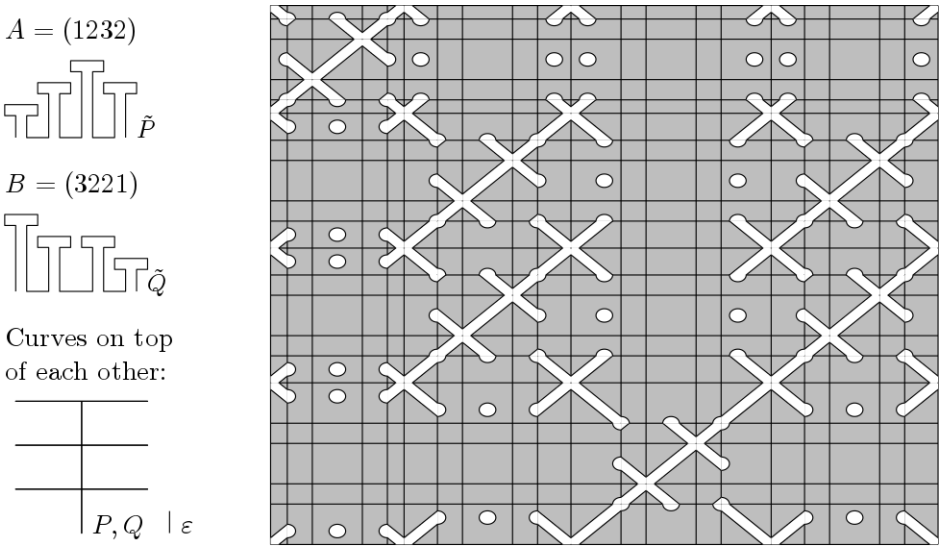


Figure 2 The curves and resulting free space diagram corresponding to strings A and B.

Note that the antenna-like symbol in the bottom left corner consists of the actual curves P and Q that resemble the input strings A and B. To make the curves more visible we added illustrations $\tilde{P}$, $\tilde{Q}$ above.

For the decision problem we need to identify a collection of components of size at most k, which is a constant and part of the input in this case, that projects surjectively onto the parameter spaces. As discussed this corresponds to the problem of finding a common string partition consisting of at most k substrings per string. The latter has been proven to be NP-hard [7] and the result transfers to our problem.

When we use (strong) Fréchet distance instead of weak Fréchet distance for the subcurves, our reduction still works. In fact, even a simpler construction suffices for this variant: we construct the curves as before but omit the horizontal line segments, i.e., we only get (one-dimensional) "I"-shaped subcurves. Again, we get that two letters are equal iff the corresponding subcurves have Fréchet distance 0. Observe that this simplified reduction does not work for the weak Fréchet distance which allows to backtrack. Therefore we use the T-shape for the weak Fréchet, which makes backtracking ineffective, in the sense that it is impossible to map a subcurve of P representing a letter of A to a subcurve of Q representing a letter of B with reversed orientation. ◀

▶ Remark. Our reduction from MCSP also works for the variants MCSP^c and k-MCSP, which translate to curves with at most c different heights of T-shapes (so basically the same curves as above) and curves with no more than k identical copies of T-shapes of each height.

▶ **Theorem 3.2.** *The optimization version of the k-Fréchet-problem is APX-complete.*

Proof. The reduction above provides a one-to-one relationship between MCSP and k-Fréchet, as any selection of components can be directly translated into common substrings. Therefore a selection of components (say, of quality $k + c$ with c being a constant and k the size of an optimal solution) corresponds to a division of the input strings into the exact same number of substrings, thus proving that our reduction is a strict AP-reduction. Since k-MCSP is APX-hard (in the optimization version), so is the k-Fréchet-Problem. APX-completeness follows from the existence of a 2-approximation algorithm which we present in the Section 4. ◀

▶ **Lemma 3.3.** *For curves in one dimension, that is $P, Q\colon [0,1] \to \mathbb{R}$, k-Fréchet distance equals Hausdorff and weak Fréchet distance, whereas strong k-Fréchet distance remains NP-complete.*

Proof. Equality of k-Fréchet distance, Hausdorff and weak Fréchet distance can be shown using the Mountain Climbing Theorem, which states that it is possible for two climbers to proceed from foot to top of a mountain while always remaining on equal height. Of course, the climbers have to start at the same time but on different sides of the same mountain [2], [6]. For the strong Fréchet distance as underlying distance measure the simplified reduction mentioned above (I-shapes) shows NP-hardness for one-dimensional curves. ◀

4 Algorithmic approaches

First, we observe that a brute force approach results in a runtime exponential in k, and then present an efficient 2-approximation algorithm (regarding the size of the found selection of components) using a greedy approach.

▶ Remark. The k-Fréchet distance can be computed in $\mathcal{O}(k \cdot n^{2k})$ time.

The brute force approach simply checks for all selections of k components of the free space whether their joint projections cover both parameter spaces. That means we have to check at most $\binom{n^2}{k}$ possible combinations of components, which results in a runtime of $\mathcal{O}(k \cdot n^{2k})$ for fixed k which is only feasible for very small k.

To approximately decide the k-Fréchet distance, we greedily choose a set of components that cover both parameter spaces.

For this we compute the free space F_ε, project its components onto the two parameter spaces and interpret the projections as intervals. We store the intervals in sorted lists. For each parameter space we then use a scan to greedily select the smallest number of intervals that cover it. The worst case that might result is the following: the intervals we select correspond to different components in the free space for the two parameter spaces, so that the union of our selections is of size $s_1 + s_2$ where s_1 and s_2 are the number of selected components for the respective parameter spaces. A different selection of size $s = \max(s_1, s_2)$ might cover both parameter spaces but is not detected by the greedy scan. So the size of our selection indicates whether $\delta_{kF} \leqslant \varepsilon$: if it is smaller or equal to k the answer is positive, i.e. $\delta_{kF} \leqslant \varepsilon$, and if it is larger than $2k$ the answer is definitely negative. For any number of selected components in between k and $2k$ we cannot rule out that there might be a smaller selection of at most k components that covers the parameter spaces.

Computing the free space takes quadratic time. Sorting the lists adds another logarithmic factor while the scan takes linear time in the number of intervals. All in all we get a runtime of $\mathcal{O}(n^2 \log n)$.

▶ **Theorem 4.1.** *The algorithm described above runs in $\mathcal{O}(n^2 \log n)$ time and finds a selection of components that covers both parameter spaces iff one exists. A found selection contains at worst twice the minimum number of components needed.*

We conjecture that the approximation factor 2 is probably not tight, as we were not able to construct an example where it is. Showing that it is or that a better approximation factor holds remains open.

5 Conclusion and further work

We introduced a new distance measure, the k-Fréchet distance, which lies in between Hausdorff and (weak) Fréchet distance. We showed NP-hardness of deciding two variants of this distance between polygonal curves. Finding the minimum number of subcurves is even APX-hard. Furthermore, we presented a polynomial time algorithm that works for small fixed k as well as an efficient 2-approximation algorithm. We close this section with some open questions.

Since the MCSP-Problem has been studied intensively, we hope to transfer more of the interesting results to our k-Fréchet-Problem: It was shown in [3] that MCSP is fixed-parameter tractable, therefore we plan to investigate whether our problem is as well. We defined the k-Fréchet distance as subdivision of curves where we match subcurves in terms of the weak Fréchet distance, but it would be interesting to just reparameterize the curves allowing jump discontinuities. As a result it is possible to match a subcurve of one to several subcurves of the other original curve. Our reduction does not work in that case and we believe it is an even harder problem. Furthermore it would be interesting to see if there are special cases for our problem to which we can find algorithms with polynomial runtimes since the curves we construct are neither monotone, nor κ-straight, κ-bounded or c-packed. Finally it remains open to improve the approximation factor achieved by our greedy approach.

Acknowledgments. We are grateful to Simon Pflips for proofreading and interesting, helpful discussions.

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1-2):75–91, 1995.

2 Kevin Buchin, Maike Buchin, Christian Knauer, Günther Rote, and Carola Wenk. How difficult is it to walk the dog? In *Proc. 23rd European Workshop on Computational Geometry*, pages 170–173, 2007.

3 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 102–121. ACM, New York, 2014.

4 Peter Damaschke. Minimum common string partition parameterized. In *Algorithms in bioinformatics*, volume 5251 of *Lecture Notes in Comput. Sci.*, pages 87–98. Springer, Berlin, 2008.

5 Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: hardness and approximations. *Electron. J. Combin.*, 12:Research Paper 50, 18, 2005.

6 Jacob E. Goodman, János Pach, and Chee-K. Yap. Mountain climbing, ladder moving, and the ring-width of a polygon. *Amer. Math. Monthly*, 96(6):494–510, 1989.

7 Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *J. Comb. Optim.*, 23(4):519–527, 2012.

8 Leonie Lange. Algorithms for comparing monotone curves and terrains. Master's thesis, Ruhr-Universität Bochum, Faculty of Mathematics, 2017.

Reconstructing a convex polygon from its ω-cloud *

Prosenjit Bose[1], Jean-Lou De Carufel[2], Elena Khramtcova[3], and Sander Verdonschot[4]

1 **Carleton University, Ottawa, Canada**
jit@scs.carleton.ca
2 **University of Ottawa, Ottawa, Canada**
jdecaruf@uottawa.ca
3 **Université libre de Bruxelles (ULB), Brussels, Belgium**
elena.khramtsova@gmail.com
4 **Carleton University, Ottawa, Canada**
sander@cg.scs.carleton.ca

Abstract

An ω-wedge is the set of all points contained between two rays emanating from a single point (the apex) and separated by an angle $\omega < \pi$. Given a convex polygon P, we place the ω-wedge so that it contains P and its both rays are tangent to P. The ω-cloud of P is the curve traced by the apex of the ω-wedge as it rotates around P while maintaining tangency in both rays.

We investigate reconstructing a polygon P from its ω-cloud. Previous work on reconstructing P from probes with the ω-wedge required knowledge of the points of tangency between P and the two rays of the ω-wedge. Here we show that if ω is known, the ω-cloud alone uniquely determines P, and we give a linear-time reconstruction algorithm. Furthermore, even if we only know that $\omega < \pi/2$, we can still reconstruct P, albeit in cubic time in the number of vertices. This reduces to quadratic time if in addition we are given the location of one of the vertices of P.

1 Introduction

"Geometric probing considers problems of determining a geometric structure or some aspect of that structure from the results of a mathematical or physical measuring device, a probe." [6, Page 1] Many probing tools have been studied in the literature such as finger probes, hyperplane (or line) probes, diameter probes [5], x-ray probes, histogram (or parallel x-ray) probes, half-plane probes and composite probes to name a few. See the review of Skiena [6] and for more recent results, see Bose et al. [1] and references therein.

Closely related to a geometric probing problem is a *reconstruction* problem: Can one reconstruct an object given a set of probes? Surprisingly, for diameter probes this is not the case [5]. An ω-wedge, introduced by Bose et al. [1], is a probing device that is the (closed) set of all points contained between two rays emanating from a single point called the *apex* of the wedge. The angle ω formed by the two rays is such that $0 < \omega < \pi$. A probe of a convex n-gon P is *valid* when P is inside the wedge and both rays of the wedge are tangent to P, see Fig. 1a. A valid probe returns the coordinates of the apex and of the two points of contact between the rays and the polygon. A convex n-gon can be reconstructed using between $2n-3$ and $2n+5$ such probes [1], depending on the value of ω and the number of *narrow vertices* (vertices whose internal angle is at most ω) in P. As the ω-wedge rotates around P, the locus of the apex of the ω-wedge describes a curve called an *ω-cloud* (see Fig. 1c).

The ω-cloud is a generalization of the diameter function of Rao and Goldberg [5]. A *diameter probe* consists of two parallel calipers turning around a convex object P in the plane. The *diameter function* returns the distance between the calipers as they turn around P. As

* P. Bose, J.-L. De Carufel, and S. Verdonschot were partially supported by NSERC, E. Khramtcova was partially supported by SNF Early Postdoc Mobility grant P2TIP2-168563 and F.R.S.- FNRS.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

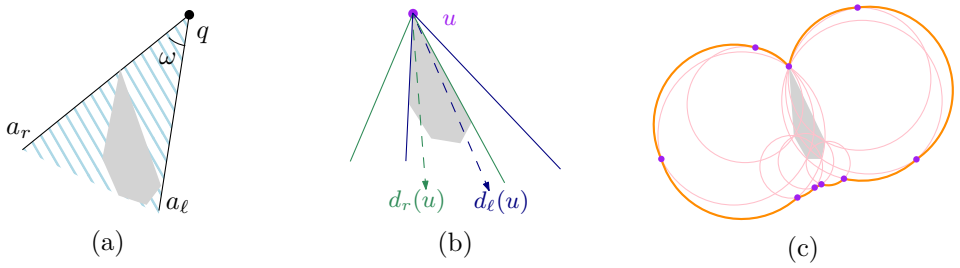


Figure 1 A convex polygon P (shaded area), and: (a) A minimal ω-wedge of P (tiling pattern); (b) A narrow vertex u of P, wedges $W_\ell(u)$ and $W_r(u)$ (bounded by, resp., blue and green solid lines) and their directions $d_\ell(u)$ and $d_r(r)$ (dashed lines); (c) The ω-cloud Ω of P: the arcs (orange lines), pivots (purple disk marks), and all the supporting circles (light-pink lines).

two different convex polygons can have the same diameter function [5], recovering a convex n-gon given only its diameter function is not always possible. An ω-wedge can be seen as two non-parallel calipers turning around P. Here we show that the ω-cloud function is free from the above drawback, and thus is a more advantageous than the diameter function.

In this paper, we analyze the structure of ω-cloud, resulting in many interesting properties, including the uniqueness of the polygon for a given ω-cloud (see Sec. 2). Further, we show, that if the value of ω is known, P can be reconstructed from its ω-cloud in $O(n)$ time and $O(k)$ space, where k is the number of narrow vertices; the required space is constant for any fixed value of ω (see Sec. 3.1). If the value of ω is not known, we can still recover P, as long as $\omega < \pi/2$. In this case, we give an $O(n^3)$ time and $O(n^2)$ space reconstruction algorithm. The time complexity reduces to $O(n^2)$ if, in addition, we know a vertex of P and no three vertices of P are on one supporting circle of an arc of the ω-cloud (see Sec. 3.2). Due to space constraints, many proofs are omitted; they can be found in the full version of this paper [2].

2 Properties of the ω-cloud

In this section we introduce the necessary definitions and notation, and then we list the properties of the ω-cloud (Lemmas 2.2-2.6), which lead to the uniqueness of the polygon for a given ω-cloud (Thm. 2.8) and are the basis for our reconstruction algorithms (see Sec. 3).

Let P be an n-vertex convex polygon in $\mathbb{R}^2$. For any vertex v of P, let $\alpha(v)$ be the internal angle of P at v. Let ω be an angle with $0 < \omega < \pi$. Consider an *ω-wedge* W; recall that it is the set of points contained between two rays emanating from the same point q (the apex of W) such that the angle between the two rays is ω. We call the ray a_ℓ (resp., a_r) that bounds W from the left (resp., right) as seen from q, the *left* (resp., *right*) *arm* of W. See Fig. 1a. We say that an ω-wedge W is *minimal* for P if P is contained in W and the arms of W are tangent to P. The *direction* of W is given by the bisector ray of the two arms of W. For each direction, there is a unique minimal ω-wedge.

▶ **Definition 2.1.** The *ω-cloud* of P is the locus of the apexes of all minimal ω-wedges for P.

The ω-cloud Ω of P is a circular sequence of circular arcs, where each two consecutive arcs share an endpoint. An *arc* Γ of the ω-cloud is a maximal contiguous portion of Ω that corresponds to apexes of combinatorially same ω-wedges (i.e., the arms of all the wedges touch the same pair of vertices in P). Each two consecutive arcs share an endpoint called *pivot* of Ω. If $\omega \geq \pi/2$, two consecutive arcs of the ω-cloud can have same supporting circle. We call the pivot connecting such arcs a *hidden pivot*. There are between n and $2n$ pivots [3].

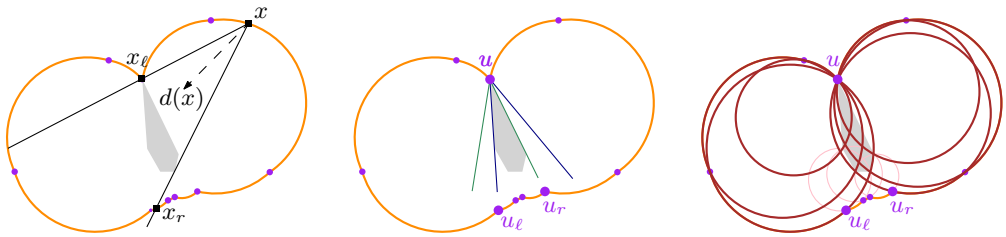
A vertex v of P is *narrow* if $\alpha(v) \leq \omega$. A pivot of Ω coincides with a vertex of P if and only if that vertex is narrow; such a pivot is also *narrow*. If $\alpha(v) < \omega$, we call v (the vertex or the pivot) *strictly narrow*. As the *portion of* Ω *between two points* $s, t \in \Omega$, denoted by Ω_{st}, we refer to the open portion of Ω encountered when traversing Ω from s to t clockwise. The *angular measure* of an arc Γ is the angle spanned by Γ, measured from the center of its supporting circle. For two points s, t on Ω, the *total angular measure* of Ω from s to t, denoted by $D_\Omega(s, t)$, is the sum of the angular measures of all arcs in Ω_{st}.

Each point x in the interior of an arc corresponds to a unique minimal ω-wedge $W(x)$ with direction $d(x)$. Let u be a pivot of Ω. If u is not strictly narrow, u also corresponds to a unique minimal ω-wedge $W(u)$ with direction $d(u)$. Otherwise, u corresponds to a closed interval of directions $[d_\ell(u), d_r(u)]$, where the angle between $d_\ell(u)$ and $d_r(u)$ equals $\omega - \alpha(u)$. See Fig. 1b. Let $W_\ell(u)$ and $W_r(u)$ denote the minimal ω-wedges with apex at u and directions resp. $d_\ell(u)$ and $d_r(u)$. For points x on Ω that are not strictly narrow pivots, we define $d_r(x)$ and $d_\ell(x)$ both to be equal to $d(x)$, and both $W_\ell(x), W_r(x)$ equal to $W(x)$.

The following is a crucial property of the ω-cloud, lying in the basis of the other properties.

▶ **Lemma 2.2.** *Let s and t be two points on Ω such that there are no narrow pivots between s and t. Then the angle β between $d_r(s)$ and $d_\ell(t)$ is $D_\Omega(s, t)/2$.*

Proof (sketch). If Ω_{st} is a single arc, angle β equals the angle between the left arms of the two minimal ω-wedges corresponding to $d_r(s)$ and $d_\ell(t)$. This angle by elementary geometry equals $D_\Omega(s, t)/2$. If Ω_{st} consists of several arcs, since none of the pivots between s and t narrow, angle β is the total sum of the corresponding angles for all the arcs.



(a) (b) (c)

Figure 2 (a) Point x in the interior of an arc of Ω, wedge $W(x)$, direction $d(x)$, and points x_ℓ and x_r. (b) Narrow pivot u, wedges $W_\ell(u)$ and $W_r(u)$, points u_ℓ and u_r. (c) Narrow pivot u, the points $v = u_\ell$ and $w = u_r$, and the supporting circles of all the arcs between them (bold brown lines).

▶ **Corollary 2.3.** *For any arc Γ of Ω, $|\Gamma| \leq 2(\pi - \omega)$.*

Let x be a point on Ω. The open ray of the right arm of $W_\ell(x)$ intersects Ω at least once. Among the points of this intersection, let x_ℓ be the one closest to x. Define the point x_r analogously for the left arm of $W_r(x)$. See Figs. 2a,b.

▶ **Lemma 2.4.** *(a) Both $\Omega_{x_\ell x}$, $\Omega_{x x_r}$ contain no narrow pivots. (b) If x is a narrow pivot, then $D_\Omega(x_\ell, x) = D_\Omega(x, x_r) = 2(\pi - \omega)$. (c) If x is not narrow, then either $D_\Omega(x, x_r) = 2(\pi - \omega)$, or x_r is the clockwise first narrow pivot after x. A symmetric statement holds for x_ℓ.*

▶ **Lemma 2.5.** *Let u be a pivot of Ω, and let v and w be the points on Ω such that $D_\Omega(v, u) = D_\Omega(u, w) = 2(\pi - \omega)$. (a) If pivot u is narrow, then the supporting circles of all the arcs of Ω_{vw} pass through u. See Fig. 2c. (b) If Ω_{vu} consists of a single arc, or there is an arc Γ of Ω_{vu} that is not incident to u, such that the supporting circle of Γ contains u, then u is narrow. A symmetric statement holds for Ω_{uw}.*

With Lemma 2.5 we can identify all narrow pivots on Ω that are not hidden, so we now turn our attention to the properties of hidden pivots.

▶ **Lemma 2.6.** *Let u be a hidden pivot of Ω, let Γ_ℓ and Γ_r be the two arcs of Ω incident to u, and let v and w be the other endpoints of Γ_ℓ and Γ_r, respectively. Then v, u, and w are all narrow and each of the arcs Γ_ℓ, Γ_r has angular measure $2(\pi - \omega)$.*

▶ **Corollary 2.7.** *If all arcs of the ω-cloud of P have the same supporting circle C, then $k = \pi/(\pi - \omega)$ is an integer and P is a regular k-gon inscribed in C.*

Suppose Ω is the ω-cloud of a convex polygon P. Lemmas 2.5 and 2.6 uniquely identify the narrow pivots of Ω, which are the narrow vertices of P (including hidden narrow pivots). By Lemma 2.4a, the portion of Ω between any two narrow pivots has total angular measure at least $2(\pi - \omega)$. Thus the components of P as defined by excluding all narrow vertices are uniquely determined by Ω: For each such component Lemma 2.4b,c gives the minimal ω-wedge W with the apex at some point x in that component. Wedge W intersects the supporting circle of an arc Γ, incident to or containing x, in the two vertices of P tangent to the arms of the minimal ω-wedge as its apex traverses Γ. This proves the following.

▶ **Theorem 2.8.** *Given an angle ω, and a circular sequence Ω of at least two circular arcs, there is at most one convex polygon P such that Ω is the ω-cloud of P.*

3 Reconstructing P from its ω-cloud

Let Ω be a circular sequence of circular arcs. Our goal is to reconstruct the convex polygon P for which Ω is the ω-cloud, or determine that no such polygon exists. Sec. 3.1 and 3.2 consider respectively ω to be given or not. As opposed to the above sections, here we consider arcs of Ω to be *maximal* portions of the same circle, that is, no two neighboring arcs have the same supporting circle. This is natural for the reconstruction task, since as an input we are given a locus of the apexes of all the minimal ω-wedges and no additional information.

If Ω is a single (maximal) arc, i.e., it is a circle C, then P is not unique: By Cor. 2.7, it is a regular $\pi/(\pi - \omega)$-gon inscribed in C; but the position of its vertices on C is impossible to identify given only Ω and ω. Thus we assume that Ω has at least two arcs.

3.1 An ω-aware reconstruction algorithm

We are given an angle ω, $0 < \omega < \pi$, and a circular sequence of at least two circular arcs Ω. We want to check if Ω is the ω-cloud of some convex polygon P, and to return P if this is the case. Our algorithm performs two passes through Ω. The first pass computes a list S of all strictly narrow vertices of P that are not hidden pivots. With each such vertex u, we store the supporting lines of the two edges of P incident to u. The second pass reconstructs separately the portion of P for each connected component of Ω induced by the vertices in S. After giving the procedure for the latter task in Lemma 3.1, we present the two passes.

For two points u and v on Ω, let P_{uv} be the union of the edges and vertices of P touched by the arms of the minimal ω-wedge as its apex traverses Ω_{uv}. Note that P_{uv} consists of at most two connected portions of P; it is possible that one of the portions is a single vertex.

▶ **Lemma 3.1.** *Given a portion Ω_{uv} with no strictly narrow pivots and direction $d_r(u)$, the portion P_{uv} can be reconstructed in time linear in the number of arcs in Ω_{uv} and $O(1)$ space.*

Proof. Let $\Gamma = uu'$ be the arc of Ω_{uv} incident to u, and let C be the supporting circle of Γ. See Fig. 3a. The values of ω and $d_r(u)$ determine the wedge $W_r(u)$. The intersection

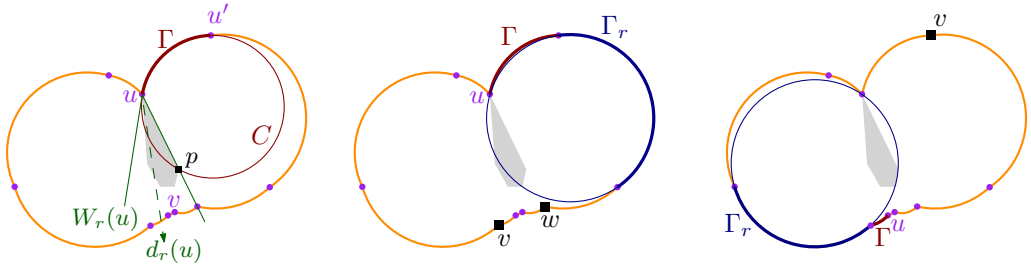


Figure 3 (a) Illustration for the proof of Lemma 3.1. First pass of the ω-aware algorithm: (b) u is a strictly narrow pivot, and (c) u is not a narrow pivot.

between $W_r(u)$ and C gives the two vertices of P touched by the minimal ω-wedges with apex on Γ. See the points u, p in Fig. 3a. Direction $d_\ell(u')$ equals $d_r(u) + D_\Omega(u, u')/2$ due to Lemma 2.2. If u' is inside Ω_{uv}, then u' is not a strictly narrow vertex, and thus the minimal ω-wedge with apex at u' is unique. This way we find the pair of vertices of P corresponding to each arc of Ω_{uv}. By visiting the pivots of Ω_{uv} in order, we find the vertices of each of the two chains of P_{uv} ordered clockwise. To avoid double-reporting vertices of P_{uv}, we keep the startpoints of the two chains, and if one chain reaches the startpoint of the other one, we stop reporting the points of the former one. This procedure visits each pivot of Ω_{uv} once, performing $O(1)$ operations at each pivot. Only $O(1)$ storage is required. ◀

First pass. We iterate through the pivots of Ω. For the currently processed pivot u, we maintain the point v on Ω such that $D_\Omega(v, u) = 2(\pi - \omega)$. If pivot u is narrow, we jump to the point on Ω at the distance $2(\pi - \omega)$ from u. Moreover, if u is strictly narrow, we add u to the list S. If u is not narrow, we process the next pivot of Ω. We now give the details.

Let Γ be the arc of Ω incident to u and following it. Let Γ_r be the arc following Γ, and C_r be the supporting circle of Γ_r. We consider cases depending on the angular measure $|\Gamma|$ of Γ:

(a) $|\Gamma| < 2(\pi - \omega)$. See Fig. 3b,c.
  (i) Circle C_r passes through u (see Fig. 3b). Then u is narrow by Lemma 2.5b. By tracing Ω, find the point w on it with $D_\Omega(u, w) = 2(\pi - \omega)$. Add u to the list S with the lines through vu and uw, if u is strictly narrow ($\angle vuw < \omega$). Set $v := u$, and $u := w$ (regardless the later condition).
  (ii) Circle C_r does not pass through u (see Figure 3b). Then u is not narrow by Lemma 2.5a. Set u to be the other endpoint of Γ, and update v accordingly.

(b) $|\Gamma| = 2(\pi - \omega)$. Then u is narrow by Lemma 2.5b. Let w be the other endpoint of Γ. Update S, v, and u as in item a(i).

(c) $|\Gamma| = 2t(\pi - \omega)$ for some integer $t > 1$. Then Γ is in fact multiple arcs separated by hidden pivots, see Lemma 2.6 and Corollary 2.3. Let p be the other endpoint of Γ. Let w and w' be the points on Γ such that $D_\Omega(u, w) = 2(\pi - \omega)$ and $D_\Omega(w', p) = 2(\pi - \omega)$. Update S, v, and u as in item a(i).

(d) Otherwise, stop and report that Ω is not an ω-cloud of any polygon.

Second pass. If list S is empty, we apply the procedure of Lemma 3.1 to the whole Ω. In particular, as both the start and the endpoint, we take the point x with which we completed the first pass of the algorithm; the point x' such that $D_\Omega(x', x) = 2(\pi - \omega)$ is already known from the first pass. Then $d_r(x) = d(x)$ is the direction of the minimal ω-wedge with the apex at x and the right arm passing through x'.

Suppose now the list S contains k vertices. They subdivide Ω into k connected portions that are free from strictly narrow non-hidden pivots. Each portion is treated as follows. If it is a single maximal arc of measure $2t(\pi - \omega)$, we separate it by $t - 1$ equidistant points, and those points are exactly the vertices of the saught portion of P, see Lemma 2.6. Otherwise, it is a portion free from any strictly narrow pivots. We process it as in Lemma 3.1.

▶ **Theorem 3.2.** *Given an angle ω such that $0 < \omega < \pi$, and a circular sequence of circular arcs Ω which is not a single circle, there is an algorithm to check if Ω is the sequence of the maximal arcs, corresponding to the ω-cloud of some n-vertex convex polygon P, and to return P if this is the case. The algorithm works in $O(n)$ time, making two passes through the input, and it requires $O(k)$ storage, where k is the number of strictly narrow vertices of P. In particular, the required storage is constant for any fixed value of ω.*

3.2 An ω-oblivious reconstruction algorithm

▶ **Theorem 3.3.** *Given a circular sequence of circular arcs Ω, there is an algorithm that finds the convex polygon P such that Ω is the ω-cloud of P for some angle ω with $0 < \omega < \pi/2$, if such a polygon exists. Otherwise, it reports that such a polygon does not exist.*

(i) If no additional information is given, the algorithm works in $O(n^3)$ time and $O(n^2)$ space.

(ii) If a vertex v of P is given, and each supporting circle of an arc of Ω is guaranteed to pass through exactly two vertices of P, the algorithm works in $O(n^2)$ time and $O(n^2)$ space.

Proof. Our algorithm, summarized below, is based on the following property: if $0 < \omega < \pi/2$, each vertex of P lies on at least two distinct supporting circles of arcs of the ω-cloud of P.

In both cases (i) and (ii), we first construct the arrangement $\mathcal{A}$ of all the supporting circles of the arcs of Ω. This can be done in $O(n^2)$ time and $O(n^2)$ space [4].

(i) For each pair of vertices u, v of $\mathcal{A}$ incident to the same circle C, we do the following. Construct a wedge W passing through u and v, such that the apex x of W lies in the interior of the unique arc Γ of Ω supported by C, such that the corresponding angle ω at x is less than $\pi/2$. Run the algorithm of Thm. 3.2 for Ω, angle ω, and the direction $d(x)$ of W.

We process $O(n^2)$ pairs of vertices in total, processing one pair takes $O(n)$ time, thus the total time spent on the reconstruction of P is $O(n^3)$.

(ii) If v is not a vertex of $\mathcal{A}$, stop and return a negative answer. Otherwise, choose a circle C containing v. Run the above procedure for v and each vertex u of $\mathcal{A}$ with $u \in C, u \neq v$.

Since each circle of $\mathcal{A}$ incident to v is incident to only one other vertex of P, the minimal ω-wedge corresponding to Γ must pass through v. Thus we just consider one circle C incident to v. Since there are $O(n)$ vertices of $\mathcal{A}$ on the circle C, the algorithm runs in $O(n^2)$ time. ◀

References

1 P. Bose, J.-L. De Carufel, A. Shaikhet, and M. Smid. Probing convex polygons with a wedge. *Comput. Geom.*, 58:34–59, 2016. doi:10.1016/j.comgeo.2016.06.001.

2 P. Bose, J.-L. De Carufel, E. Khramtcova, and S. Verdonshot. Reconstructing a convex polygon from its ω-cloud. *ArXiv e-prints*, 2018. arXiv:1801.02162.

3 P. Bose, M. Mora, C. Seara, and S. Sethia. On computing enclosing isosceles triangles and related problems. *Int. J. of Comput. Geom. & Appl.*, 21(01):25–45, 2011.

4 H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Computing*, 15(2):341–363, 1986.

5 A. S. Rao and K. Y. Goldberg. Shape from diameter: Recognizing polygonal parts with a parallel-jaw gripper. *I. J. Robotic Res.*, 13(1):16–37, 1994.

6 S. Skiena. Problems in geometric probing. *Algorithmica*, 4(4):599–605, 1989.

Computing optimal shortcuts for networks*

Delia Garijo[1], Alberto Márquez[1], Natalia Rodríguez[2], and Rodrigo I. Silveira[3]

1 **Departamento de Matemática Aplicada I, Universidad de Sevilla, Spain**
dgarijo@us.es,almar@us.es
2 **Departamento de Computación, Universidad de Buenos Aires, Argentina**
nrodriguez@dc.uba.ar
3 **Departament de Matemàtiques, Universitat Politècnica de Catalunya, Spain**
rodrigo.silveira@upc.edu

Abstract

We augment a plane Euclidean network with a segment or *shortcut* to minimize the largest distance between any two points along the edges of the resulting network. In this continuous setting, the problem of computing distances and placing a shortcut is much harder as all points on the network, instead of only the vertices, must be taken into account. Our main result for general networks states that it is always possible to determine in polynomial time whether the network has an optimal shortcut and compute one in case of existence. We also improve this general method for networks that are paths, restricted to using two types of shortcuts: those of any fixed direction and shortcuts that intersect the path only on its endpoints.

1 Introduction

A *geometric network* of points in the plane is an undirected graph whose vertices are points in $\mathbb{R}^2$ and whose edges are straight-line segments connecting pairs of points. A *Euclidean network* is an edge-weighted geometric network: edges are assigned lengths equal to the Euclidean distance between their endpoints. When in addition there are no crossings between edges, the Euclidean network is said to be *plane*. In the following, we shall simply say network, it being understood as plane Euclidean network.

In this work we study a variant of the Diameter-Optimal-k-Augmentation problem that deals with inserting k additional segments into a network, while minimizing the largest distance in the resulting network (see the survey [7] for more on augmentation problems over plane geometric graphs). Concretely, we consider a continuous version of the problem for $k = 1$: the endpoints of the inserted segment, called *shortcut*, are allowed to be any two points on the network (instead of only vertices), and we seek to minimize the largest distance between any two points on the edges of the augmented network. The complexity of the problem, which lies in the fact that all points must be considered in computing distances and placing the shortcut, motivates that there are very few results on this continuous version.

Yang [8] designed three different approximation algorithms to compute for certain types of paths an *optimal* shortcut which, informally, is a segment that attains the minimum of that largest distance. De Carufel et al. [3] gave an algorithm to determine in linear time optimal shortcuts for paths and optimal pairs of shortcuts ($k = 2$) for convex cycles. We want to stress that their definition of shortcut is simpler, as they do not consider the intersections between the shortcut and the network as points of the augmented network. The

* D.G. and R.S. were supported by project MTM2015-63791-R. R. S. was also supported by Gen. Cat. 2017SGR1640 and MINECO through the Ramón y Cajal program. A.M. was supported by project BFU2016-74975-P.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

same definition of shortcut was used in [4] to develop a study for trees, which includes the computation of an optimal shortcut for a tree of size n in $O(n \log n)$ time.

The first approach for general networks was presented in [2] where the authors compute shortcuts (i.e., segments whose insertion improve the diameter) in polynomial time, but they do not obtain optimal shortcuts. In Section 2, we do decide existence and compute such optimal shortcuts in polynomial time. Section 3 focuses on paths: we first analyze how distances change by the insertion of a shortcut, and compute the largest distance between any two points on the augmented network in $\Theta(n)$ time. We also improve the method of Section 2 for shortcuts of any fixed direction and shortcuts that intersect the path only at its endpoints. Due to space limitations, proofs are briefly sketched.

1.1 Preliminaries

The *locus* of a network $\mathcal{N} = (V(\mathcal{N}), E(\mathcal{N}))$, denoted by $\mathcal{N}_\ell$, is the set of all points of the Euclidean plane that are on $\mathcal{N}$. Thus, $\mathcal{N}_\ell$ is treated indistinctly as a network or as a closed point set. We write $a \in \mathcal{N}_\ell$ for a point a on $\mathcal{N}_\ell$, and $V(\mathcal{N}) \subset \mathcal{N}_\ell$. We will use $\mathcal{P}_\ell$ instead of $\mathcal{N}_\ell$ when $\mathcal{N}_\ell$ is a path. A *path* P connecting two points a, b on $\mathcal{N}_\ell$ is a sequence $au_1 \dots u_k b$ such that $u_1u_2, \dots, u_{k-1}u_k \in E(\mathcal{N})$, a is a point on an edge ($\neq u_1u_2$) incident to u_1, and b is a point on an edge ($\neq u_{k-1}u_k$) incident to u_k. We use $|P|$ to denote P's *length*, i.e., the sum of the lengths of all edges u_iu_{i+1} plus the lengths of the segments au_1 and bu_k The length of a shortest path P from a to b is the *distance* between a and b on N_ℓ. This distance is written as $d_{N_\ell}(a, b)$ or $d(a, b)$ when the network is understood, and whenever $ab \notin E(N_\ell)$, it is different from the Euclidean distance between the points, denoted by $|ab|$.

The *eccentricity* of a point $a \in \mathcal{N}_\ell$ is $\mathrm{ecc}(a) = \max_{b \in \mathcal{N}_\ell} d(a, b)$, and the *diameter* of $\mathcal{N}_\ell$ is $\mathrm{diam}(\mathcal{N}_\ell) = \max_{a \in \mathcal{N}_\ell} \mathrm{ecc}(a)$. Two points $a, b \in \mathcal{N}_\ell$ are *diametral* whenever $d(a, b) = \mathrm{diam}(\mathcal{N}_\ell)$, and a shortest path between them is then called *diametral path*.

A *shortcut* for $\mathcal{N}_\ell$ is a segment s with endpoints on $\mathcal{N}_\ell$ such that $\mathrm{diam}(\mathcal{N}_\ell \cup s) < \mathrm{diam}(\mathcal{N}_\ell)$. We say that shortcut s is *simple* if its two endpoints are the only intersection points with $\mathcal{N}_\ell$, and s is *maximal* if it is the intersection of a line and $(\mathcal{N}_\ell \cup s)$, i.e., $s = (\mathcal{N}_\ell \cup s) \cap \ell$, for some line ℓ. A shortcut is *optimal* if it minimizes $\mathrm{diam}(\mathcal{N}_\ell \cup s)$ among all shortcuts s for $\mathcal{N}_\ell$.

2 General networks

The main result in [2] states that one can always determine in polynomial time whether a network $\mathcal{N}_\ell$ has a shortcut, and compute one in case of existence. In this section, we first state the analogous result but for optimal shortcuts. Our proof mainly uses the ideas in [2], but some additional information is needed to capture the property of being optimal.

By Lemma 3.2 of [2], $\mathrm{diam}(\mathcal{N}_\ell)$ can be computed in polynomial time, and the diametral pairs of points on $\mathcal{N}_\ell$ are either two vertices, or two points on distinct non-pendant edges, or a pendant vertex and a point on a non-pendant edge (an edge is pendant if one of its vertices has degree one). Thus, with some abuse of notation, we say that a diametral pair may be vertex-vertex, edge-edge, or vertex-edge.

Let $\alpha, \beta \in V(\mathcal{N}) \cup E(\mathcal{N})$, and let $e = uv$ and $e' = u'v'$ be two edges of $\mathcal{N}$. When α is an edge, we use $\mathrm{ecc}(u, \alpha)$ to indicate the maximum distance from u to the points on α (analogous for β and the remaining endpoints of e and e'); if α is a vertex, $\mathrm{ecc}(u, \alpha) = d(u, \alpha)$. In general, $\mathrm{ecc}(\alpha, \beta) = \max_{t \in \alpha, z \in \beta} d(t, z)$.

▶ **Lemma 2.1.** *Let $y = ax+b$ be a line intersecting edges $e = uv$ and $e' = u'v'$ on, respectively, points p and q, and let $\alpha, \beta \in V(\mathcal{N}) \cup E(\mathcal{N})$. For each pair (w, z) with $w \in \{u, v\}$ and*

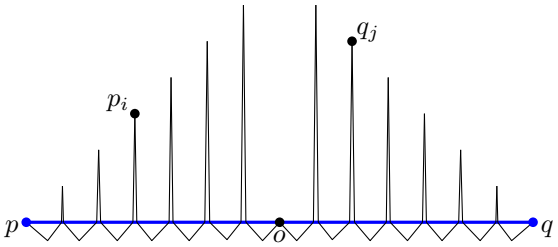


Figure 1 $\Theta(n)$ spikes placed symmetrically with respect to the midpoint o of a shortcut pq. The spikes are spaced by one unit each, while their heights are set such that the distance from o to the top of the spike is always the same, namely $|pq|/2$. Thus, the distance between the top of one spike on the left of o and one on its right, like p_i and q_j, is $|pq|$, and equals the diameter of $\mathcal{P}_\ell \cup pq$.

$z \in \{u', v'\}$, *function* $f^{w,z}_{\alpha,\beta}(a,b) = \text{ecc}(w,\alpha) + d(w,p) + |pq| + d(q,z) + \text{ecc}(z,\beta)$ *is linear in* b.

Given a line m that crosses two fixed edges $e, e' \in E(\mathcal{N})$, let $\mathcal{P}_{e,e'}(m)$ be the set of equivalent lines to m that intersect edges e and e'.[1] Consider a line $r \equiv y = ax+b$ in $\mathcal{P}_{e,e'}(m)$ and the set $I = \{e = e_0, e_1, \ldots, e_k, e_{k+1} = e'\}$ of edges that it intersects in between e and e'; let $e_i = u_iv_i$ and $p_i = r \cap e_i$. For a fixed diametral pair $\alpha, \beta \in V(\mathcal{N}) \cup E(\mathcal{N})$, a function of the type $f^{w,z}_{\alpha,\beta}(a,b)$ with $w \in \{u_i, v_i\}$ and $z \in \{u_j, v_j\}$, $0 \le i \ne j \le k+1$, computes $\text{ecc}(\alpha,\beta)$ when using a path that passes through vertices w, z and contains segment p_ip_j. For a fixed value of a, each function $f^{w,z}_{\alpha,\beta}(a,b)$ becomes a linear function on b. Thus, geometrically, an optimal shortcut for $\mathcal{N}_\ell$ in $\mathcal{P}_{e,e'}(m)$ is given by the minimum of the upper envelope of the set of lines $f^{w,z}_{\alpha,\beta}(a,b)$. Note that any shortcut s satisfies that $\text{ecc}(t) < \text{diam}(\mathcal{N}_\ell)$ for every $t \in s$, and so all segments p_ip_j must be included in the set of diametral pairs α, β. Applying the same argument to the $O(n^2)$ regions $\mathcal{P}_{e,e'}(m)$, we obtain the following theorem.

▶ **Theorem 2.2.** *It is always possible to determine in polynomial time whether a network $\mathcal{N}_\ell$ admits an optimal shortcut, and compute one in case of existence.*

It would be interesting to characterize the networks $\mathcal{N}_\ell$ that have an optimal shortcut, even restricted to simple shortcuts. The following proposition is a first approach to this question. Note that one must distinguish between an *optimal simple shortcut* and a *simple optimal shortcut*. The first is a shortcut that is optimal in the set of simple shortcuts; this is different of being optimal in the set of all shortcuts and, in addition, to be simple.

▶ Proposition 1. Let $\mathcal{N}$ be a network whose locus $\mathcal{N}_\ell$ admits a simple shortcut, and let $\overline{\mathcal{N}}$ be the network resulting from adding to $\mathcal{N}$ all edges of the convex hull of $V(\mathcal{N})$. If all faces of $\overline{\mathcal{N}}$ are convex, then $\mathcal{N}_\ell$ has an optimal simple shortcut.

3 Path networks

We begin by noting that the insertion of a shortcut to a path can create a quadratic number of diametral pairs; as illustrated in the construction in Figure 1.

3.1 Diameter after inserting a shortcut

The diameter of $\mathcal{P}_\ell$ can be immediately computed in linear time, however, the addition of a shortcut s can create a linear number of new bounded faces, thus in principle it is not clear

[1] Two lines are *equivalent* if the half-planes to the right (left) they define contain the same vertices of $\mathcal{N}$

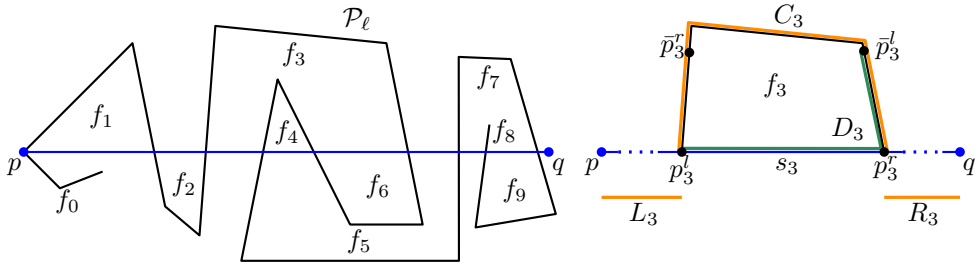


Figure 2 Left: faces created by s; f_0 and f_8 are degenerate faces. Right: detail for f_3 (note that the chain bounding f_4 is not considered for f_3). Thick lines are used here to denote distances.

whether $\mathrm{diam}(\mathcal{P}_\ell \cup \mathrm{s})$ can be computed in linear time, i.e., without computing the diameter between each pair of faces. The main result in this section is that this is still possible.

Suppose, without loss of generality, that $s = pq$ is horizontal and maximal. Assume (bounded) faces are numbered in the order of their left endpoints from left to right along s (using right endpoints to disambiguate). If the first vertex of $\mathcal{P}_\ell$ is not on s, we consider the path from its first vertex to the first intersection of $\mathcal{P}_\ell$ with s as a (degenerate) face (analogous for the last vertex of $\mathcal{P}_\ell$), see Figure 2(left). A face f_i can be bounded by several chains of $\mathcal{P}_\ell$. However, we are only interested in associating f_i to its chain with leftmost endpoint on s. Thus a face f_i will be defined by a subsegment s_i of s from point p_i^l to p_i^r, and a polygonal chain C_i on one side of s, see Figure 2(right). Let $|C_i|$ be the length of C_i, and let $L_i = |pp_i^l|$ and $R_i = |p_i^r q|$ be the distances to the leftmost and rightmost endpoints of s. Finally, let D_i be the distance on $\mathcal{P}_\ell \cup s$ from p_i^l to its furthest point $\bar{p}_i^l$ on f_i. Note that this is identical to the distance from p_i^r to the analogously defined point $\bar{p}_i^r$, and also to the semiperimeter of f_i, which is equal to $(|C_i| + |s_i|)/2$.

▶ **Observation 3.1** (Disjoint faces). *Let f_i, f_j be two faces of $(\mathcal{P}_\ell \cup s)$ with $s_i \cap s_j = \emptyset$ and s_i to the left of s_j. The diameter of $C_i \cup p_i^l p_j^r \cup C_j$ is $D_i + |p_i^r p_j^l| + D_j = D_i + R_i - R_j - |s_j| + D_j$ and is achieved by $\bar{p}_i^r$ and $\bar{p}_j^l$.*

▶ **Observation 3.2** (Nested faces). *Let f_i, f_j be two faces of $(\mathcal{P}_\ell \cup s)$ with $s_j \subset s_i$. The diameter of $C_i \cup s_i \cup C_j$ is $\frac{1}{2}(|C_i| + |p_i^l p_j^l| + |p_i^r p_j^r| + |C_j|) = \frac{1}{2}(|C_i| + L_j - L_i + R_j - R_i + |C_j|)$.*

▶ **Observation 3.3** (Overlapping faces). *Let f_i, f_j be two faces of $(\mathcal{P}_\ell \cup s)$ with $s_i \cap s_j \neq \emptyset$, $p_i^l \notin s_j$ and $p_j^r \notin s_i$. The diameter of $C_i \cup p_i^l p_j^r \cup C_j$ is $\frac{1}{2}(|C_i| + |p_i^l p_j^l| + |p_i^r p_j^r| + |C_j|) = \frac{1}{2}(|C_i| + L_j - L_i + R_i - R_j + |C_j|)$.*

The preceding observations reveal a key property: the linear ordering between faces induced by s defines uniquely how the diameter between two faces is achieved. Thus, the algorithm for computing $\mathrm{diam}(\mathcal{P}_\ell \cup \mathrm{s})$ in linear time starts by going along $\mathcal{P}_\ell$ and computing all intersections with s in the order of $\mathcal{P}_\ell$. Then we apply a linear-time algorithm for Jordan sorting [6] to obtain the intersections in the order along s, say, from right to left. Within the same running time we can obtain the necessary information of each face created by the insertion of s. Next we compute and store certain information for each face f_i: its furthest faces, respectively, to the right and to the left, its furthest face nested inside f_i, and its furthest face with one endpoint in f_i and the other one outside. When sweeping the faces along s, this information allows us to find in $O(1)$ time, for each face f_i, its furthest face from the ones seen so far, so the maximum distance between any two faces can be found in total linear time.

▶ **Theorem 3.4.** *Given a path $\mathcal{P}_\ell$ with n vertices and a shortcut s, the diameter of $(\mathcal{P}_\ell \cup s)$ can be computed in $\Theta(n)$ time.*

3.2 Optimal horizontal shortcuts

The observations in Section 3.1 also give us a way to compute an optimal horizontal shortcut for a path considerably faster than using the general method in Section 2. After a suitable rotation, this allows to find optimal shortcuts of any fixed orientation.

Assume again that shortcuts are horizontal and maximal, so they can be treated as horizontal lines. Now, consider the vertices in $\mathcal{P}_\ell$ sorted increasingly by y-coordinate, and let y_a, y_b, with $y_a < y_b$, be the y-coordinates of two consecutive vertices in that order. By Observations 3.1–3.3, the distance between any two faces f_i and f_j is a linear function $d_{ij}(y)$ for $y_a \leq y \leq y_b$. Thus, each face is associated with $k-1$ lines in 2D where k is the total number of faces, leading to a set $\mathcal{L}$ of $\Theta(k^2)$ lines (note that $k = O(n)$). The optimal shortcut over all $y \in [y_a, y_b]$ is given by the minimum of the upper envelope of $\mathcal{L}$, which can be computed in $O(k^2 \log k)$ time [5]. If this is done with each of the $n-1$ horizontal strips formed by consecutive vertices of $\mathcal{N}_\ell$, the optimal horizontal shortcut is obtained in total $O(n^3 \log n)$ time. Now, this method can be improved if, instead of computing from scratch the upper envelope of $\mathcal{L}$ at each horizontal strip, we maintain the upper envelope between consecutive strips and only add or remove the lines that change when going from one strip to the next one. The changes between two consecutive strips are of three types: (i) one of the two line segments bounding a face within the strip changes; (ii) a face ends; (iii) a new face appears. In the worst case, $n-1$ lines are removed from $\mathcal{L}$ and another $n-1$ lines are added to $\mathcal{L}$. Maintaining the upper envelope of n lines is equivalent to maintaining the convex hull of n points in 2D, which can be done in $O(n \log n)$ amortized time and using $O(n^2)$ space [1].

▶ **Theorem 3.5.** *For every path $\mathcal{P}_\ell$ with n vertices, it is possible to find an optimal horizontal shortcut in $O(n^2 \log n)$ time, using $O(n^2)$ space.*

3.3 Optimal simple shortcuts

Consider now a simple shortcut $s = pq$ for a path $\mathcal{P}_\ell$ with endpoints u, v. Suppose that point p is closer to u than q along P_ℓ; let $x = d(u, p)$ and $y = d(v, q)$. There is only one bounded face in $P_\ell \cup s$ whose boundary is a cycle $C(p, q)$. Let $\overline{p}$ and $\overline{q}$ be the farthest points from, respectively, p and q on $C(p, q)$, and let $z = (d_{P_\ell}(p, q) - |pq|)/2$. Note that $d(\overline{p}, \overline{q}) = |pq|$ and $z = d(p, \overline{q}) = d(\overline{p}, q)$. See Figure 3(left). There are three candidates for diametral path in $\mathcal{P}_\ell \cup s$ (see [3]): (1) the path from u to v via s is diametral if and only if $z = \min\{x, y, z\}$, (2) the path from u to $\overline{p}$ via s is diametral whenever $y = \min\{x, y, z\}$, (3) the path from v to $\overline{q}$ via s is diametral if and only if $x = \min\{x, y, z\}$. Thus, $\operatorname{diam}(\mathcal{P}_\ell \cup s) \in \{x + y + |pq|, x + z + |pq|, y + z + |pq|\}$. Further, in [3] it is proved that $\mathcal{P}_\ell$ has an optimal shortcut satisfying $x = y$, which allows to compute it in linear time. Their method does not apply here because, as explained in the Introduction, their definition of shortcut leads to a much simpler situation. Nevertheless, in the same fashion, we can prove the following lemma.

▶ **Lemma 3.6.** *Let pq be an optimal simple shortcut for $\mathcal{P}_\ell$. The following statements hold.*

1. *If neither p nor q are vertices of $\mathcal{P}_\ell$ then $x = y = z$.*
2. *If p or q are vertices of $\mathcal{P}_\ell$ then the two smallest values among x, y, z are equal.*

With Lemma 3.6 in hand, we first compute the points p, q where $x = y = z$ by solving $O(n)$ quadratic equations, and obtain $O(n)$ candidates for optimal simple shortcut such that

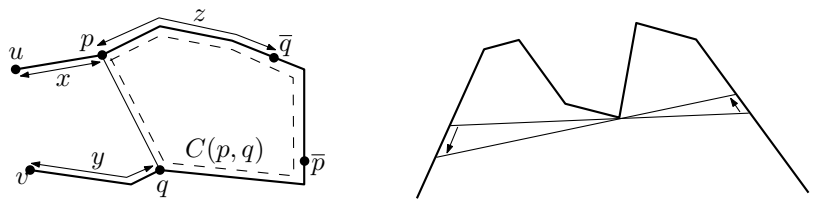


Figure 3 Left: Inserting a simple shortcut pq. Right: Shortcut that is pivoting on a vertex.

$p, q \notin V(\mathcal{P}_\ell)$. We then classify them into three sets: $\mathcal{S}$ of simple shortcuts, $\mathcal{L}$ of limit cases (the segment intersects $\mathcal{P}_\ell$ on three points), and shortcuts that intersect $\mathcal{P}_\ell$ on four points. Candidate segments with at least one endpoint in $V(\mathcal{P}_\ell)$ must then be included in $\mathcal{S}$ and $\mathcal{L}$; this last set also contains those segments that are pivoting on a vertex of $\mathcal{P}_\ell$ (see Figure 3(right)) and such that the two smallest values among x, y, z are equal. Finally, we obtain the minimum value of $\operatorname{diam}(\mathcal{P}_\ell \cup s)$ over $s \in \mathcal{S} \cup \mathcal{L}$; there exists an optimal simple shortcut whenever the minimum is attained by a segment in $\mathcal{S}$.

▶ **Theorem 3.7.** *It is always possible to decide whether a path $\mathcal{P}_\ell$ with n vertices has an optimal simple shortcut and compute one (in case of existence) in $O(n^2)$ time.*

4 Conclusion

We compute optimal shortcuts for general networks and improve our method for paths but restricted to simple shortcuts and those of any fixed direction. This is an ongoing research and our first priority is to investigate techniques that allow us to design a more efficient algorithm for computing an optimal shortcut (with no restriction) for a path. It would be then interesting to develop a similar algorithmic study for more general networks and to consider the analogous problems when augmenting the network with more than one shortcut.

References

1 G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 617–626, 2002.

2 J. Cáceres, D. Garijo, A. González, A. Márquez, M. L. Puertas, and P. Ribeiro. Shortcut sets for the locus of plane Euclidean networks, 2016. arXiv:1604.03878. Extended Abstract in ENDM, 54:163-168, 2016.

3 J. L. De Carufel, C. Grimm, A. Maheshwari, and M. Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. In *Proc. 15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016*, pages 27:1–27:14, 2016.

4 J. L. De Carufel, C. Grimm, S. Schirra, and M. Smid. Minimizing the continuous diameter when augmenting a tree with a shortcut. In *Algorithms and Data Structures. WADS 2017. LNCS 10389*, pages 301– 312, 2017.

5 M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008.

6 K. Y. Fung, T. M. Nicholl, R. E. Tarjan, and C. J. Van Wyk. Simplified linear-time Jordan sorting and polygon clipping. *Inf. Proc. Letters*, 35(2):85 – 92, 1990.

7 F. Hurtado and C. D. Tóth. Plane geometric graph augmentation: A generic perspective. In *Pach J. (eds) Thirty Essays on Geometric Graph Theory*, pages 327–354. Springer, 2013.

8 B. Yang. Euclidean chains and their shortcuts. *Theor. Comput. Sci.*, 497:55–67, 2013.

A Note on Flips in Diagonal Rectangulations*

Jean Cardinal[1], Vera Sacristán[2], and Rodrigo I. Silveira[2]

1 **Université libre de Bruxelles (ULB), Brussels, Belgium**
jcardin@ulb.ac.be
2 **Universitat Politècnica de Catalunya (UPC), Barcelona, Spain**
{vera.sacristan, rodrigo.silveira}@upc.edu

Abstract

Rectangulations are partitions of a square into axis-aligned rectangles. A number of results provide bijections between combinatorial equivalence classes of rectangulations and families of pattern-avoiding permutations. Other results deal with local changes involving a single edge of a rectangulation, referred to as flips, edge rotations, or edge pivoting. Such operations induce a graph on equivalence classes of rectangulations, related to so-called *flip graphs* on triangulations and other families of geometric partitions. In this note, we consider a family of flip operations on the equivalence classes of *diagonal rectangulations*, and their interpretation as transpositions in the associated *Baxter* permutations, avoiding the vincular patterns $\{3\underline{14}2, 2\underline{41}3\}$. This complements results by Law and Reading (JCTA, 2012) and provides a complete characterization of flip operations on diagonal rectangulations, in both geometric and combinatorial terms.

1 Introduction

In order to understand the underlying combinatorial structure of geometric space partitions such as triangle meshes or floorplans, it is often useful to define elementary operations that modify this structure locally. We can then connect distinct partitions using sequences of such operations. In triangulations, such a notion is known under the term of *flip*. A flip in a triangulation is typically defined as the replacement of an edge shared by two triangles forming a convex quadrilateral by the other diagonal of the quadrilateral. This allows the definition of a *flip graph*, the vertices of which are triangulations, and in which two triangulations are adjacent whenever one can be obtained from the other by a single flip. Flip graphs have applications in enumeration and random generation of geometric partitions as well as optimization, and have also been shown to have intimate links with many important structures in combinatorics, such as the Catalan objects, the Tamari lattice and the associahedra, cyclohedra, and partial cubes.

The objects of interest in this paper are *rectangulations*, defined as partitions of a square into axis-aligned rectangles. There exists a collection of results establishing bijections between classes of rectangulations and pattern-avoiding permutations [2, 4, 5, 8, 9]. A permutation σ is said to contain the *pattern* π, where π is another permutation, whenever there exists a subsequence of σ whose elements are in the same relative order as the elements of π. Pattern-avoiding permutations are families of permutations that do not contain any occurrence of one or more given patterns. We use the more general *vincular* notation for forbidden patterns, in which an underlined pair of elements indicates that they need to occur consecutively in the permutation . For instance, forbidding the pattern $3\underline{14}2$ amounts to forbidding all occurrences of the pattern 3142 with the added condition that 1 and 4 must occur consecutively.

* V.S. and R.I.S. were partially supported by projects Gen. Cat. 2017SGR1640 and MTM2015-63791-R (MINECO/FEDER). R.I.S. was also supported by MINECO through the Ramón y Cajal program.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Different types of local operations can be defined on rectangulations, which have been given different names, such as flips, local moves, edge rotations, or edge pivoting. In general, they all consist in replacing a horizontal edge of the rectangulation by a vertical one, or vice versa. In what follows, and with a slight abuse of terminology, we will refer to all those operations under the common name of *flip*.

Law and Reading [8] described a family of flips on rectangulations and provided an elegant combinatorial characterization. They showed that two rectangulations were connected by such a flip if and only if they were in the cover relation of a certain natural lattice structure, analogous to the Tamari lattice on triangulations. This lattice was also studied by Giraudo [7] under the name of *Baxter lattice*. Ackerman, Barequet and Pinter [3] defined related flip operations on rectangulations of a point set. These rectangulations are defined on a given point set so that every point lies on a segment of the rectangulation, and vice versa. Ackerman et al. studied the flip graph induced by these operations [1]. The flips considered by Ackerman et al. are the same as the ones in Law and Reading whenever the point set lies on the diagonal. Their results include a linear upper bound on the diameter of this flip graph (see [1], Section 4).

Our results. We first describe a known bijection from diagonal rectangulations to Baxter permutations, avoiding the vincular patterns $\{3\underline{14}2, 2\underline{41}3\}$. Then we consider flip operations on diagonal rectangulations, classify the different kinds of flips and give a combinatorial interpretation for each. Those involving edges that do not intersect the diagonal of the square, have already been characterized by Law and Reading [8]. For the others, we prove that the obtained flip graph is isomorphic to the graph on the corresponding Baxter permutations in which two Baxter permutations are adjacent whenever they differ by a single transposition of consecutive elements. This provides a complete one-to-one correspondence not only between rectangulations and Baxter permutations, but also between these sets of natural operations on the geometric and combinatorial structures. Due to space constraints, all proofs are omitted from this abstract, but can be found in the arXiv version.[1]

2 Diagonal rectangulations and Baxter permutations

The material of this section is adapted from Ackerman et al. [2], and Law and Reading [8]. A description of an essentially equivalent map in terms of pairs of twin binary trees was given by Felsner et al. [6].

A rectangulation is a partition of the unit square into axis-aligned rectangles. We define *vertices* as corners of the rectangles, and *edges* as line segments connecting two vertices, with no other vertex in between. The term *segment* is used to refer to inclusionwise maximal line segments of the rectangulation, possibly composed of several edges. We consider only rectangulations in which every vertex has exactly three incident edges, except the four vertices of the square, which have exactly two incident edges. We classify the vertices into four self-explanatory classes denoted by $\vdash$, $\dashv$, $\top$, and $\bot$.

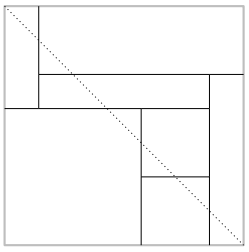


Figure 1 A diagonal rectangulation.

We refer to the top-left to bottom-right diagonal of the square as the *main* diagonal. A *diagonal* rectangulation is a rectangulation in which every rectangle intersects the main diagonal. An

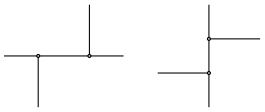


Figure 2 Forbidden configurations.

[1] https://arxiv.org/abs/1712.07919

example of diagonal rectangulation is given in Figure 1. However, we actually define diagonal rectangulations as equivalence classes of such partitions of the square, with respect to changes of vertex locations that preserve the adjacency relation between the rectangles. We have the following characterization of (the equivalence classes of) diagonal rectangulations.

▸ **Lemma 2.1.** *A rectangulation is diagonal if and only if it does not contain one of the two forbidden configurations of Figure 2.*

We can also consider the equivalence classes of rectangulations for which we can change the adjacency relation between the rectangles. Two rectangulations are then said to be equivalent when one can be obtained from the other by performing so-called *wall slides*, as shown on Figure 3. The equivalence relation is sometimes referred to as *R-equivalence* [4], and the R-equivalence classes are called *mosaic floorplans*.

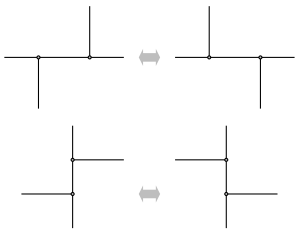


Figure 3 Wall slides.

▸ **Lemma 2.2.** *Every mosaic floorplan, or R-equivalence class, has a unique representative as a diagonal rectangulation.*

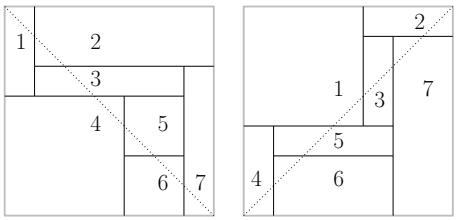


Figure 4 Illustration of the map B on the rectangulation R of Figure 1. The obtained Baxter permutation is $\mathrm{B}(R) = 4651372$.

We now describe the bijection B between diagonal rectangulations and *Baxter* permutations, which avoid the patterns $\{3\underline{1}42, 2\underline{4}13\}$. In order to define B, we define two linear orders on the rectangles of a rectangulation: the $\searrow$-order and the $\nearrow$-order. The $\searrow$-order is the order in which the rectangles are intersected by the main diagonal, from top-left to bottom-right. The $\nearrow$-order is obtained by taking the representative $\nearrow R$ of R in the equivalence class of mosaic floorplans such that the bottom-left to top-right diagonal intersects every rectangle. By Lemma 2.2, this representative exists and is unique. The order in which this diagonal intersects the rectangle is the $\nearrow$-order. The map B can then be described as follows:

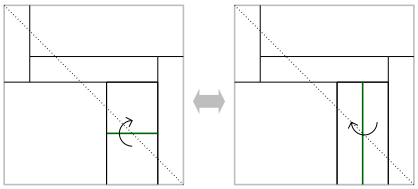
1. label the rectangles with respect to the $\searrow$-order,
2. enumerate the labels of the rectangles in the $\nearrow$-order.

▸ **Theorem 2.3.** [2] *The map* B *is a bijection between diagonal rectangulations with n rectangles and Baxter permutations on n elements.*

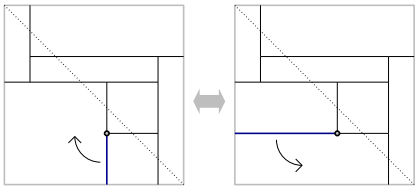
3 Flips

We consider only flipping edges that are not part of the boundary of the square. We say that an edge is *matched* at one of its endpoints whenever this endpoint is incident to another edge with the same (horizontal/vertical) orientation.

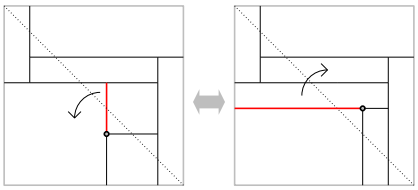
Simple flips involve edges cutting a rectangle into two rectangles, which are precisely the edges that are unmatched at both endpoints. In a diagonal rectangulation, all such edges must intersect the diagonal. A simple flip consists in replacing such a horizontal edge by a vertical one, or vice versa. When replacing the edge, we can always do it in such a way that the resulting rectangulation remains diagonal. An example of simple flip in the rectangulation of Figure 1 is given in Figure 5a.



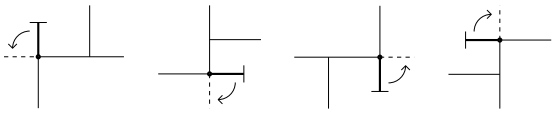
(a) Simple flips



(b) Flip involving an edge that does not intersect the diagonal



(c) Flip involving an edge intersecting the diagonal



(d) The four types of unflippable edges matched at only one endpoint

Figure 5 Flips in diagonal rectangulations.

In some cases, an edge that is matched only at one of its endpoints can be *rotated* about this endpoint to yield another diagonal rectangulation. Examples of such flips are given in Figures 5b and 5c.

However, not all edges can be flipped. An edge is said to be unflippable in two cases. If the edge is matched at both endpoints, and if rotating this edge about any of the two endpoints yields a partition that is not a rectangulation. It can also be the case that an edge is matched at only one endpoint, and rotating it about this endpoint yields a rectangulation, but the obtained rectangulation is not diagonal. Unflippable edges matched at only one endpoint can be shown to come in four types, illustrated in Figure 5d.

4 A complete combinatorial characterization of flips

A *transposition* maps a permutation $\pi = \pi(1)\pi(2)\dots\pi(j)\dots\pi(k)\dots\pi(n)$ to a permutation $\pi' = \pi(1)\pi(2)\dots\pi(k)\dots\pi(j)\dots\pi(n)$. Furthermore, if the two values j and k satisfy $|\pi(j) - \pi(k)| = 1$, then the transposition is said to be a *transposition of consecutive elements*. If $k = j + 1$, then the transposition is said to be an *adjacent transposition*. Note that an adjacent transposition corresponds to a transposition of consecutive elements in the inverse permutation.

We first summarize a result of Law and Reading, characterizing some of the flip operations described above as a cover relation in a lattice, which can be found in Section 7 of [8]. In what follows, we will use the term *Law-Reading flips* to refer to those flips. In the original description (Section 7 of [8]), Law-Reading flippable edges are defined in terms of a *locking* operation. The following lemma gives a simple alternative definition of Law-Reading flips.

▸ **Lemma 4.1.** *Law-Reading flips are exactly the flips that are either simple, or that involve the rotation of a flippable edge that does not intersect the diagonal, as illustrated in Figure 5b.*

We now give a combinatorial characterization of Law-Reading flips proved in [8] using the map from rectangulations to Baxter permutations. Before stating the result, we must define the lattice dRec_n of diagonal rectangulations with n rectangles.

The *weak order* (also known as the weak *Bruhat* order) is a partial order on the set S_n of permutations of n elements in which a permutation π is smaller than another permutation

π' whenever the set of inversions of π is a subset of the set of inversions of π'. The cover relation of the weak order is the set of pairs of permutations that differ by a single adjacent transposition. The lattice dRec_n on diagonal rectangulations can be defined as the restriction of the weak order to the Baxter permutations corresponding to diagonal rectangulations with n rectangles. Recall that $B(R)$ is the Baxter permutation associated with the diagonal rectangulation R.

▸ **Theorem 4.2** (Law and Reading [8]). *Let R and R' be two diagonal rectangulations. Then R and R' are connected by a Law-Reading flip if and only if $\mathrm{B}(R)$ and $\mathrm{B}(R')$ are in a cover relation in dRec_n.*

This means that the two Baxter permutations corresponding to the pair of rectangulations are related by a monotone sequence of adjacent transpositions, and the intermediate permutations, if any, are not Baxter permutations.

We define *Barcelona flips* as those flips that involve a flippable edge intersecting the main diagonal. Barcelona flips are either simple flips, or flips involving the rotation of an edge intersecting the diagonal, as shown in Figure 5c.

▸ **Lemma 4.3.** *Let R and R' be two diagonal rectangulations that are connected by a Barcelona flip. Then $\boxslash R$ and $\boxslash R'$ are connected by a Law-Reading flip.*

The lemma is illustrated in Figure 6. Combining the above lemma with an observation on the way to obtain $\boxslash R$ from the inverse permutation $B(R)^{-1}$, and the characterization of Law-Reading flips in Theorem 4.2, we can already conclude that a Barcelona flip in a rectangulation R corresponds to a sequence of adjacent transpositions in $B(R)^{-1}$, that is, a sequence of transpositions of consecutive elements in $B(R)$. In fact, we can prove the following precise correspondence, involving only *single* transpositions.

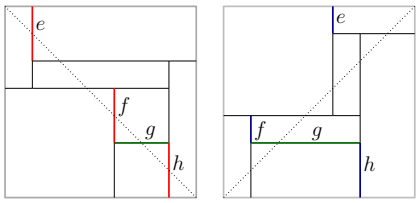


Figure 6 Illustration of Lemma 4.3: edges that can be flipped by a Barcelona flip in R (left) can be flipped by a Law-Reading flip in $\boxslash R$ (right).

▸ **Lemma 4.4.** *Let R and R' be two diagonal rectangulations. Then R and R' are connected by a Barcelona flip if and only if $\mathrm{B}(R)$ and $\mathrm{B}(R')$ differ by a single transposition of consecutive elements.*

The following theorem summarizes our results.

▸ **Theorem 4.5.** *Two diagonal rectangulations R and R' are connected by a flip if and only if one of these two conditions hold:*

- *$\mathrm{B}(R)$ and $\mathrm{B}(R')$ differ by a single transposition of consecutive elements,*
- *$\mathrm{B}(R)$ and $\mathrm{B}(R')$ are in a cover relation in dRec_n.*

Furthermore, R and R' are connected by a simple flip if and only if both conditions hold.

The flip graph on diagonal rectangulations with four rectangles is given in Figure 7.

Acknowledgments. This work was initiated while the first author was visiting UPC Barcelona in spring 2017. The authors wish to thank A. Asinowski, S. Felsner, and V. Pilaud for useful discussions and comments.

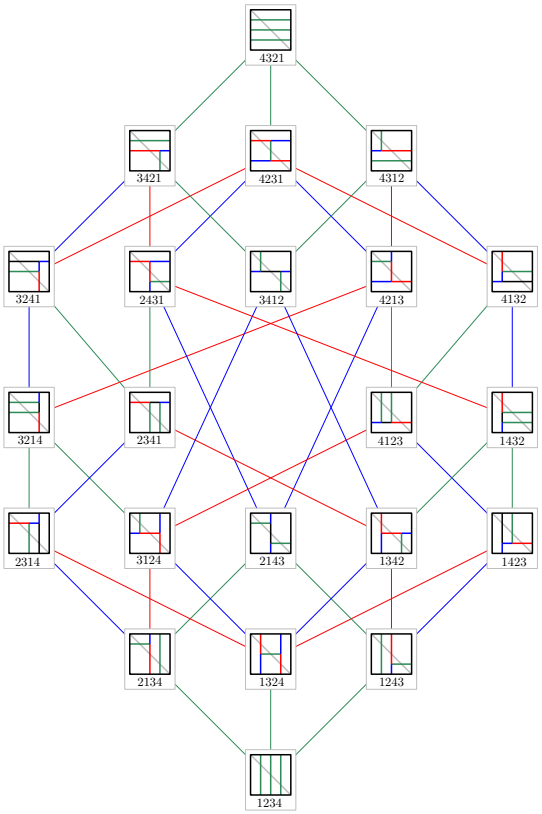


Figure 7 The flip graph on diagonal rectangulations made of four rectangles. In each rectangulation, the green edges are simply-flippable, and the blue and red edges are respectively Law-Reading and Barcelona-flippable, but not simply flippable.

References

1 Eyal Ackerman, Michelle M. Allen, Gill Barequet, Maarten Löffler, Joshua Mermelstein, Diane L. Souvaine, and Csaba D. Tòth. The Flip Diameter of Rectangulations and Convex Subdivisions. *Discrete Mathematics & Theoretical Computer Science*, 18(3), 2016.

2 Eyal Ackerman, Gill Barequet, and Ron Y. Pinter. A bijection between permutations and floorplans, and its applications. *Disc. App. Math.*, 154(12):1674–1684, 2006.

3 Eyal Ackerman, Gill Barequet, and Ron Y. Pinter. On the number of rectangulations of a planar point set. *J. Comb. Theory, Ser. A*, 113(6):1072–1091, 2006.

4 Andrei Asinowski, Gill Barequet, Mireille Bousquet-Mélou, Toufik Mansour, and Ron Y. Pinter. Orders induced by segments in floorplans and (2-14-3, 3-41-2)-avoiding permutations. *Electr. J. Comb.*, 20(2):P35, 2013.

5 Serge Dulucq and Olivier Guibert. Stack words, standard tableaux and Baxter permutations. *Disc. Math.*, 157(1):91–106, 1996.

6 Stefan Felsner, Éric Fusy, Marc Noy, and David Orden. Bijections for Baxter families and related objects. *J. Comb. Theory, Ser. A*, 118:993–1020, 2011.

7 Samuele Giraudo. Algebraic and combinatorial structures on pairs of twin binary trees. *J. Algebra*, 360(Supplement C):115–157, 2012.

8 Shirley Law and Nathan Reading. The Hopf algebra of diagonal rectangulations. *J. Comb. Theory, Ser. A*, 119(3):788–824, 2012.

9 Nathan Reading. Generic rectangulations. *Eur. J. Comb.*, 33(4):610–623, 2012.

3D-Disk-Packing*

Helmut Alt[1], Otfried Cheong[2], Ji-won Park[2], and Nadja Scharf[1]

1 **Institut für Informatik, Freie Universität Berlin**
alt@mi.fu-berlin.de,nadja.scharf@fu-berlin.de
2 **School of Computing, KAIST**
otfried@kaist.edu,wldnjs1727@kaist.ac.kr

Abstract

In this article, we consider the problem of finding in three dimensions a minimum volume axis-parallel cuboid container into which a given set of unit size disks can be packed under translations. The problem is neither known to be NP-hard nor to be in NP. We give a constant factor approximation algorithm based on reduction to finding a shortest Hamiltonian path in a weighted graph. As a byproduct, we can show that there is no finite size container into which all unit disks can be packed simultaneously.

1 Introduction

Packing a set of geometric objects in a nonoverlapping way into a minimum size container is an intriguing problem and because of its practical significance it has been widely investigated. For a survey see [1, 6] and the references therein. Even simple variants like packing a set of rectangles into a rectangular container turn out to be NP-hard [4]. Whereas that simple problem is in NP, in many cases not much is known about the true complexity of the problem.

Constant factor approximation algorithms of polynomial running time have been found for many variants of the packing, in particular for finding minimum size rectangular or convex containers for a set of convex polygons under translations [2], i.e., the objects may be translated but rotations are not allowed. Also, approximation algorithms for rigid motions (translations and rotations) are known in this case.

In three dimensions, approximation algorithms for packing cuboids or convex polyhedra into minimum volume cuboid or convex containers are known if rigid motions are allowed [3]. It remains an open problem whether this is possible for translations only. In this paper, we give a positive answer for a restricted set of possible objects, namely disks of unit radius and axis-parallel cuboid containers. So far, our approximation factor is forbiddingly high but it should be of theoretical interest that the problem, which is neither known to be NP-hard nor to be in NP, can be approximated in polynomial time at all.

Packing disks in 3D is meant in the following sense: We say that two disks *touch* if their intersection contains only one point and that two disks *intersect* if their intersection consists of more than one point. By *nonoverlapping*, we mean that no two disks intersect whereas it is allowed that two disks touch. The main problem we study in this work is then defined as follows:

▶ **Definition 1.1** (3D-3D-Disk-Packing)**.** Given a set of unit disks by their unit normal vectors in $\mathbb{R}^3$. The goal is to find

- an axis-parallel box of minimum volume such that all disks can be packed without overlapping under translation inside the box

* This work was partially supported by a fellowship within the FITweltweit program and by the Johann-Gottfried-Herder program of the German Academic Exchange Service (DAAD).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

- and the actual packing of the disks inside the box.

We assume that no two disks are the same, i.e., no two normal vectors are parallel.

We will reduce approximating this problem with a constant factor to approximating the following problem with a constant factor.

▶ **Definition 1.2** (1D-3D-Disk-Packing)**.** Given a set of nonidentical unit disks by their normal vectors in three dimensional space and an additional vector defining the direction of a line. The goal is to find an ordering of the disks with the following property: If the disks are placed nonoverlappingly with their centers in this order on the line, the distance from the center of the first to the center of the last disk is minimum. We call the distance of the center of the first to the center of the last disk when stabbed by the line the *length of the ordering*. See Figure 1 for a 2D example.

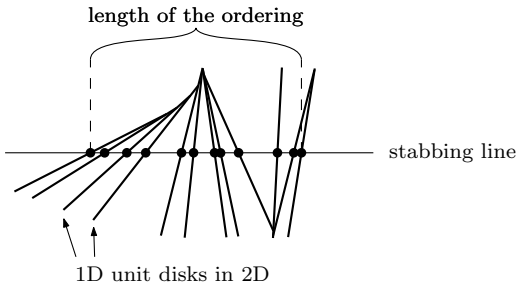


Figure 1 A (nonoptimal) solution to the 1D-2D-Interval-Packing problem. Here, the unit disks are unit line segments.

This problem then again will be reduced to finding the shortest Hamiltonian path in a complete weighted graph.

Let $a \in \mathbb{R}^3$ be a vector. Define $\mathrm{h}_a(D_1, D_2)$ to be the distance of the centers of the disks D_1 and D_2 when placed with their centers on a line parallel to the vector a such that D_1 and D_2 touch. $\mathrm{h}_a(D_1, D_2)$ can be computed easily from the normal vectors of D_1 and D_2 and it can be shown that $\mathrm{h}_a(D_1, D_2) = \mathrm{h}_a(D_2, D_1)$. The following lemma will be used for the reduction to Hamiltonian path.

▶ **Lemma 1.3.** *For disks D_1, D_2, D_3 and axis a, it holds that $\mathrm{h}_a(D_1, D_2) + \mathrm{h}_a(D_2, D_3) \geq \mathrm{h}_a(D_1, D_3)$, i.e., the triangle inequality holds.*

We omit the proof due to space constraints. It can be shown by contradiction, assuming that D_1 touches D_3 in a point x that is not part of D_2. Then considering the triangle formed by the centers of D_1 and D_3, and x, it can be shown that D_2 cannot fit between D_1 and D_3.

2 Approximation Algorithms

Next, we will show how to reduce the 1D-3D-Disk-Packing problem to finding the shortest Hamiltonian path in a complete weighted graph and obtain a constant factor approximation in this way. Afterwards we will use this approximation algorithm to compute a constant factor approximation for 3D-3D-Disk-Packing.

2.1 1D-3D-Disk-Packing Approximation

Algorithm 1 computes an approximate 1D-3D-Disk-Packing. In fact, since an ordering of the disks directly corresponds to a Hamiltonian path in G, the triangle inequality holds in G by

Lemma 1.3, and Hoogeveen's algorithm computes a $\frac{5}{3}$-approximation for it in polynomial time. So, we get the following theorem.

Input: n unit disks given by their normal vectors, vector a
Output: Ordering of the n disks
1 Generate complete weighted graph G with n vertices:
2 Set the weight of the edge (i, j) to $\mathrm{h}_a(D_i, D_j)$ for all $1 \leq i, j \leq n, i \neq j$;
3 For all $1 \leq i, j \leq n$ with $i \neq j$, approximate shortest Hamiltonian path on the graph with endpoints i and j with Hoogeveen's algorithm [5] and determine the overall shortest path;
4 **return** the ordering of the overall shortest path;

Algorithm 1: Approximation algorithm for 1D-3D-Disk-Packing

▶ **Theorem 2.1.** *Algorithm 1 computes a $\frac{5}{3}$-approximation for 1D-3D-Disk-Packing in polynomial time.*

In the next section, we will use Algorithm 1 to approximate 3D-3D-Disk-Packing.

2.2 3D-3D-Disk-Packing Approximation

We define $w_{\min}, d_{\min}, h_{\min}$ to be the maximum extension of any disk in x-,y-, and z-direction respectively and, thus, the minimum width, depth, and height any container for the disks must have. Let $w = s \cdot w_{\min}$ and $d = s \cdot d_{\min}$ for a constant $s > 1$ to be defined later. Algorithm 2 computes an approximate 3D-3D-Disk-Packing.

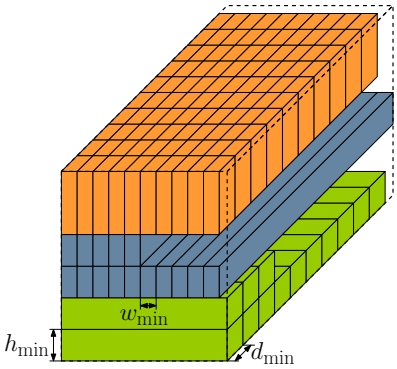


Figure 2 Example container for $s = 10.5$. The green boxes are the enlarged pieces obtained by dividing the container-box computed by Algorithm 1 for the disks in $\mathcal{X}$. Here, they form two layers. The blue boxes contain disks from $\mathcal{Y}$ and the orange boxes contain disks from $\mathcal{Z}$.

To analyze Algorithm 2 we first give a bound on W, D, and H. Observe that the angle between the normal vector of a disk and the axis it gets stabbed by in Algorithm 2 can be at most $\varphi = \arccos(\frac{1}{\sqrt{3}})$.

▶ **Lemma 2.2.** *It holds that*

$$W \leq 109 \cdot \frac{\mathrm{OPT}}{d_{\min} h_{\min}}, D \leq 109 \cdot \frac{\mathrm{OPT}}{w_{\min} h_{\min}}, H \leq 109 \cdot \frac{\mathrm{OPT}}{w_{\min} d_{\min}},$$

where OPT *is the volume of an optimal container.*

Input: n unit disks given by their normal vectors
Output: nonoverlapping packing of the disks into an axis-parallel box

1 Partition the n disks into three sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ according to the axis their normal vectors form the smallest angle with;
2 Call Algorithm 1 for the disks in $\mathcal{X}$ and vector $(1, 0, 0)$. If L_x is the length of the returned ordering, this can be interpreted as a packing of the disks in $\mathcal{X}$ into an axis-parallel box of width $W = L_x + w_{\min}$, depth $d_{\min}$, and height $h_{\min}$;
3 Analogously to Step 2 get packings for the disks in $\mathcal{Y}$ and $\mathcal{Z}$ into boxes of dimensions $w_{\min} \times D \times h_{\min}$ and $w_{\min} \times d_{\min} \times H$ respectively;
4 Divide the box obtained for $\mathcal{X}$ into pieces of width $w - w_{\min}$;
5 Assign each disk to the piece its point with smallest x-coordinate lies in;
6 Enlarge each piece from width $w - w_{\min}$ to width w such that all disks that are assigned to a piece are completely contained in that piece;
7 Divide the box obtained for $\mathcal{Y}$ into pieces of depth d analogously to Steps 4 to 6;
8 Divide the box obtained for $\mathcal{Z}$ into $\left\lfloor \frac{w}{w_{\min}} \right\rfloor \left\lfloor \frac{d}{d_{\min}} \right\rfloor$ pieces of width $w_{\min}$ and depth $d_{\min}$;
9 Analogously to Steps 5 and 6, enlarge the height of each piece by $h_{\min}$;
10 Arrange the pieces to a box of width w and depth d. The pieces containing disks of $\mathcal{X}$ form $\left\lceil \left\lceil \frac{W}{w-w_{\min}} \right\rceil / \left\lfloor \frac{d}{d_{\min}} \right\rfloor \right\rceil$ layers of height $h_{\min}$, the pieces containing disks of $\mathcal{Y}$ form $\left\lceil \left\lceil \frac{D}{d-d_{\min}} \right\rceil / \left\lfloor \frac{w}{w_{\min}} \right\rfloor \right\rceil$ layers of height $h_{\min}$, and the pieces containing disks from $\mathcal{Z}$ form one layer of height $H / \left(\left\lfloor \frac{w}{w_{\min}} \right\rfloor \left\lfloor \frac{d}{d_{\min}} \right\rfloor \right) + h_{\min}$ (See Figure 2 for an example);
11 **return** the resulting box with the packed disks;

Algorithm 2: Approximation algorithm for 3D-3D-Disk-Packing

Proof. Consider an optimal container with width W_{OPT}, depth D_{OPT}, and height H_{OPT} and let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be the partition of disks into subsets as in Algorithm 2. Furthermore consider a grid with some side length g on the x-z-plane and lines parallel to the y-axis through the grid cell centers. Then, each point has distance at most $\frac{g}{\sqrt{2}}$ to the closest line. So, every disk in $\mathcal{Y}$ is stabbed by a line in a point of distance at most $\frac{g}{\sqrt{2}\sin\left(\frac{\pi}{2}-\varphi\right)}$ from the disk center if g is small enough, i.e. $cg < 1$, where $c = \frac{1}{\sqrt{2}\sin\left(\frac{\pi}{2}-\varphi\right)} = \sqrt{\frac{3}{2}}$. See Figure 3 for illustration. Therefore, each disk in $\mathcal{Y}$ contains a disk of radius $1 - cg$ stabbed by a line through its center.

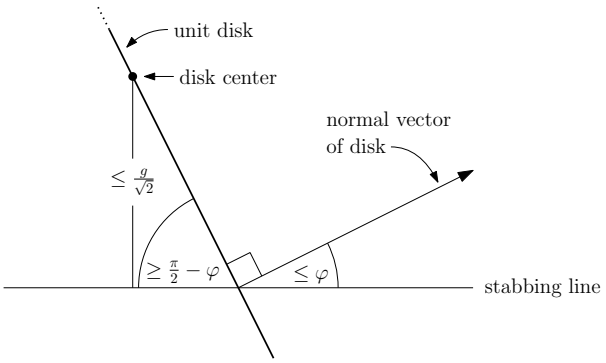


Figure 3 Distance of a disk center to the stabbing line

So, by placing the line segments that are the intersection of the container and the lines

behind each other so that they touch, we get a solution to the 1D-3D-Disk-Packing-Problem for the disks in $\mathcal{Y}$ but with radius $1 - cg$. By stretching this solution by $1/(1 - cg)$, we get a solution for disks of radius 1. Let $L_{\mathrm{OPT}_\mathcal{Y}}$ be the length of an optimal solution for the 1D-3D-Disk-Packing problem for the disks in $\mathcal{Y}$. Then, this length can be at most the length of our solution, i.e.,

$$L_{\mathrm{OPT}_\mathcal{Y}} \leq \left\lceil \frac{H_{\mathrm{OPT}}}{g} \right\rceil \left\lceil \frac{W_{\mathrm{OPT}}}{g} \right\rceil D_{\mathrm{OPT}} \cdot \frac{1}{1 - cg}.$$

By using $w_{\min}, h_{\min} \leq 2$ and $W_{\mathrm{OPT}} \geq w_{\min}, H_{\mathrm{OPT}} \geq h_{\min}$, it can be shown that

$$L_{\mathrm{OPT}_\mathcal{Y}} \leq \frac{(g+2)^2}{g^2(1-cg)} \cdot \frac{\mathrm{OPT}}{w_{\min} h_{\min}}. \tag{1}$$

Since we use Algorithm 1 to compute a 1D-3D-Disk-Packing solution for $\mathcal{Y}$, we get by Theorem 2.1

$$D \leq \frac{5}{3} \cdot L_{\mathrm{OPT}_\mathcal{Y}} + d_{\min},$$

where the extra term $d_{\min}$ comes from the fact that the length of a 1D-3D-Disk-Packing is defined as the distance of the center of the first disk to the center of the last disk and we are interested in the total depth of the packing. By inequality (1),

$$D w_{\min} h_{\min} \leq \left(\frac{5(g+2)^2}{3g^2(1-cg)} + 1 \right) \mathrm{OPT}.$$

Optimizing for g yields $g = \sqrt{\frac{1}{3}\left(27 + 4\sqrt{6}\right)} - 3$ and a factor of approximately 108.49. The calculations for W and H are analogous. This implies the lemma. ◀

Now, we are ready to state the main theorem of this article.

▶ **Theorem 2.3.** *Algorithm 2 computes a 593-approximation for 3D-3D-Disk-Packing in polynomial time.*

Proof. The container computed by Algorithm 2 is a box with base area $w \cdot d$ and height $\left\lceil \left\lceil \frac{W}{w - w_{\min}} \right\rceil / \left\lfloor \frac{d}{d_{\min}} \right\rfloor \right\rceil h_{\min} + \left\lceil \left\lceil \frac{D}{d - d_{\min}} \right\rceil / \left\lfloor \frac{w}{w_{\min}} \right\rfloor \right\rceil h_{\min} + H / \left(\left\lfloor \frac{w}{w_{\min}} \right\rfloor \left\lfloor \frac{d}{d_{\min}} \right\rfloor \right) + h_{\min}$ (See step 10 in Algorithm 2). Using Lemma 2.2, the definition of w and d (see the beginning of this section), and $w_{\min} d_{\min} h_{\min} \leq \mathrm{OPT}$ it can be shown that the volume of the container is at most

$$s^2 \left(\frac{2 \cdot \frac{109}{s-1} + 1}{s-1} + \frac{109}{(s-1)^2} + 3 \right) \mathrm{OPT}.$$

Optimizing for s gives a long term as approximation factor that is smaller than 593. ◀

3 Unbounded containers are necessary

In this section, we will conclude from our previous results that there is no bounded size container into which all unit disks can be packed. More precisely, we will show:

▶ **Theorem 3.1.** *Packing a set of n unit disks requires a container of size $\Omega(\sqrt{n})$ in the worst case.*

Proof. In the following, we will show that $\Omega(\sqrt{n})$ is a lower bound for the container constructed by Algorithm 2 which is within a constant factor of the optimal container. From that the theorem follows immediately.

Identify any unit disk with its normal vector in the unit sphere S^2. Consider a sufficiently small rectangular surface patch $P = I_1 \times I_2 \subset S^2$ where I_1, I_2 are nonempty intervals of spherical coordinates. Let P be symmetric to the equator and I_1 and I_2 sufficiently small, so that all disks corresponding to points in P are stabbed by the same axis in Algorithm 2. Furthermore, for any two points in P the shorter grand circle segment connecting them should lie completely inside P. For a given $\varepsilon > 0$, subdivide P by horizontal and vertical lines at distance ε, yielding a grid of points in P of size $n \geq c_1/\varepsilon^2$ for some constant $c_1 > 0$. Let A be the set of unit disks corresponding to the grid points. With standard geometric arguments it is possible to prove the following

▶ Claim 3.2. There is a constant $c_2 > 0$ such that for any two grid points having distance δ on S^2 the centers of the corresponding unit disks have distance at least $c_2\delta$ when stabbed consecutively on a line as in Algorithm 1.

Now observe, that if P is chosen close enough to the equator, any two distinct points in A have distance at least $c_3\varepsilon$ for some constant $c_3 > 0$. Therefore, by the previous claim the distance of the centers of the corresponding unit disks, when stabbed consecutively, is at least $c_4\varepsilon$ for some constant $c_4 > 0$. Consequently the length of a line segment stabbing all disks in A must be at least $c_4\varepsilon(n-1)$. Since $n \geq c_1/\varepsilon^2$, this is in $\Omega(\sqrt{n})$ as ε tends to 0. From Lemma 2.2 follows that this is also a lower bound for the volume of a container computed by Algorithm 2. ◀

From Theorem 3.1, we obtain immediately

▶ **Corollary 3.3.** *There is no finite size container into which all unit disks can be packed.*

This seems obvious at first glance, but observe that for the case of one-dimensional objects it is false. In fact, all unit length line segments can be packed in arbitrary dimension $d \geq 2$ into a container of finite size for example by placing them with one endpoint in the origin.

References

1 Helmut Alt. Computational aspects of packing problems. *Bulletin of the EATCS*, 118, 2016.

2 Helmut Alt, Mark de Berg, and Christian Knauer. Approximating Minimum-Area Rectangular and Convex Containers for Packing Convex Polygons. *JoCG*, 8(1):1–10, 2017.

3 Helmut Alt and Nadja Scharf. Approximating Smallest Containers for Packing Three-Dimensional Convex Objects. In *Proc. 27th Annu. Internat. Sympos. Algorithms Comput. (ISAAC)*, 2016.

4 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal Packing and Covering in the Plane are NP-Complete. *Inf. Process. Lett.*, 12(3):133–137, 1981.

5 J. A. Hoogeveen. Analysis of Christofides' Heuristic: Some Paths Are More Difficult Than Cycles. *Oper. Res. Lett.*, 10(5):291–295, 1991.

6 Guntram Scheithauer. *Zuschnitt- und Packungsoptimierung: Problemstellungen, Modellierungstechniken, Lösungsmethoden.* Vieweg+Teubner Verlag, 2008.

Combinatorics of Beacon-based Routing in Three Dimensions*

Jonas Cleve and Wolfgang Mulzer

Institut für Informatik, Freie Universität Berlin
{jonascleve,mulzer}@inf.fu-berlin.de

Abstract

A beacon is a point-like object that can be enabled to exert a magnetic pull on other point-like objects in space. Those objects then move towards the beacon in a greedy fashion until they are either stuck at an obstacle or reach the beacon's location. Beacons placed inside polyhedra can be used to route point-like objects from one location to another.

The notion of beacon-based routing was introduced by Biro et al. [FWCG'11] in 2011 for two dimensions and covered in detail by Biro in his PhD thesis [SUNY-SB'13].

We extend Biro's results to polyhedra in three dimensions. We show that $\lfloor \frac{m+1}{3} \rfloor$ beacons are always sufficient and sometimes necessary to route between any pair of points in a given polyhedron P, where m is the number of tetrahedra in a tetrahedral decomposition of P. This is one of the first results that show that beacon routing is also possible in three dimensions.

1 Introduction

A *beacon* b is a point-like object in a polyhedron P that, when enabled, exerts a magnetic pull on points inside P. The points then move in the direction in which the distance to b decreases most rapidly, possibly moving along obstacles. If an attracted point p ends its movement at b, we say that b *covers* p. A point p can be *routed via beacons* towards a point q if there exists a sequence of beacons $b_1, b_2, \ldots, b_k = q$ such that b_1 covers p and b_{i+1} covers b_i for all $1 \leq i < k$. The target $q = b_k$ is an *implicit* beacon, i.e., we need only $k - 1$ additional beacons. We can route *between two points* p and q if p can be routed via beacons towards q and vice versa. In our model, at most one beacon is enabled at any time and a point has to reach the beacon's location before the next beacon can be enabled. The notion of beacon attraction was introduced by Biro et al. [4,5] for two dimensions. It extends the classic notion of visibility [8]: the visibility region of a point p is a subset of the attraction region of p.

Here, we study the case of three-dimensional polyhedra. A three-dimensional *polyhedron* is a compact connected set bounded by a piecewise linear 2-manifold. The results in this work are based on the master's thesis of the first author [7] in which various aspects of beacon-based routing and guarding were studied in three dimensions. Simultaneously, Aldana-Galván et al. [1,2] looked at orthogonal polyhedra and introduced the notion of *edge beacons*.

For two dimensions, Biro [4] provided bounds on the number of beacons for routing in a polygon. He also showed that it is NP-hard and APX-hard to find a minimum set of beacons for a given polygon such that it is possible to (a) route between any pair of points, (b) route one specific source point to any other point, (c) route any point to one specific target point, or (d) cover the polygon, i.e., every point in P is covered by at least one beacon.

It is easy to reduce the two-dimensional problems to their three-dimensional counterparts by lifting the polygon into three dimensions. Thus the corresponding problems in three dimensions are also NP-hard and APX-hard. More details can be found in [7, Chapter 4].

* Supported in part by DFG grant MU 3501/1 and ERC StG 757609.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2 Preliminary Thoughts on Tetrahedral Decompositions

In two dimensions, Biro et al. [5] look at a triangulation of a polygon to show that for every two additional triangles at most one beacon is needed. This yields the upper bound of $\lfloor \frac{n}{2} \rfloor - 1$ beacons. Even though there is a slight flaw in the case analysis of their proof, this can be easily repaired, see [7, Chapter 3.1] for more details and the working proof. We extend their approach to three dimensions by looking at the decomposition of a polyhedron into tetrahedra. However, Lennes [9] has shown that polyhedra exist which cannot be decomposed into tetrahedra without additional vertices. To decide whether such a decomposition (without additional vertices) exists was proved to be NP-complete by Ruppert and Seidel [10].

In two dimensions, every triangulation of a polygon with n vertices and h holes has exactly $n - 2 + 2h$ triangles ($h = 0$ for simple polygons). In three dimensions, however, Chazelle [6] showed that there exist polyhedra with $\Theta(n)$ vertices for which $\Omega(n^2)$ convex parts are needed in every decomposition. Bern and Eppstein [3, p. 52] show that all polyhedra can be triangulated with $\mathcal{O}(n^2)$ tetrahedra with the help of $\mathcal{O}(n^2)$ additional *Steiner points*.

Additionally, one polyhedron can have different tetrahedral decompositions with different numbers of tetrahedra, see [10, p. 228]. Our results will therefore be relative to the number of tetrahedra m rather than the number of vertices n. We do not assume general position and thus decompositions with Steiner points are also allowed.

To successfully apply the ideas for two dimensions to three dimensions, we need the following preliminary definition and lemma.

▶ **Definition 2.1.** Given a polyhedron with a tetrahedral decomposition $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ into m tetrahedra, its *dual graph* is an undirected graph $D(\Sigma) = (V, E)$ where

(1) $V = \{\sigma_1, \ldots, \sigma_m\}$ and
(2) $E = \{\{\sigma_i, \sigma_j\} \in \binom{V}{2} \mid \sigma_i \text{ and } \sigma_j \text{ share exactly one triangular facet}\}$.

▶ **Observation 2.2.** *Unlike in two dimensions, the dual graph of a tetrahedral decomposition is not necessarily a tree. Still, each node in the dual graph has at most* 4 *neighbors.*

▶ **Lemma 2.3.** *Given a tetrahedral decomposition Σ of a polyhedron together with its dual graph $D(\Sigma)$ and a subset $S \subseteq \Sigma$ of tetrahedra from the decomposition whose induced subgraph $D(S)$ of $D(\Sigma)$ is connected, then*

(1) $|S| = 2$ implies that the tetrahedra in S share one triangular facet,
(2) $|S| = 3$ implies that the tetrahedra in S share one edge, and
(3) $|S| = 4$ implies that the tetrahedra in S share at least one vertex.

Proof. We show this seperately for every case.

(1) This follows directly from Definition 2.1.
(2) In a connected graph of three nodes there is one node neighboring the other two. By Definition 2.1, the dual tetrahedron shares one facet with each of the other tetrahedra. In a tetrahedron every pair of facets shares one edge.
(3) By case (2), there is a subset of three (connected) tetrahedra that shares one edge e. This edge is therefore part of each of the three tetrahedra. By Definition 2.1, the fourth tetrahedron shares a facet f with at least one of the other three (called σ). Since f contains three and e two vertices of σ, they share at least one vertex. ◀

3 An Upper Bound for Beacon-based Routing

We can now show an upper bound on the number of beacons needed to route within a polyhedron with a tetrahedral decomposition. The idea of the proof is based on the proof by

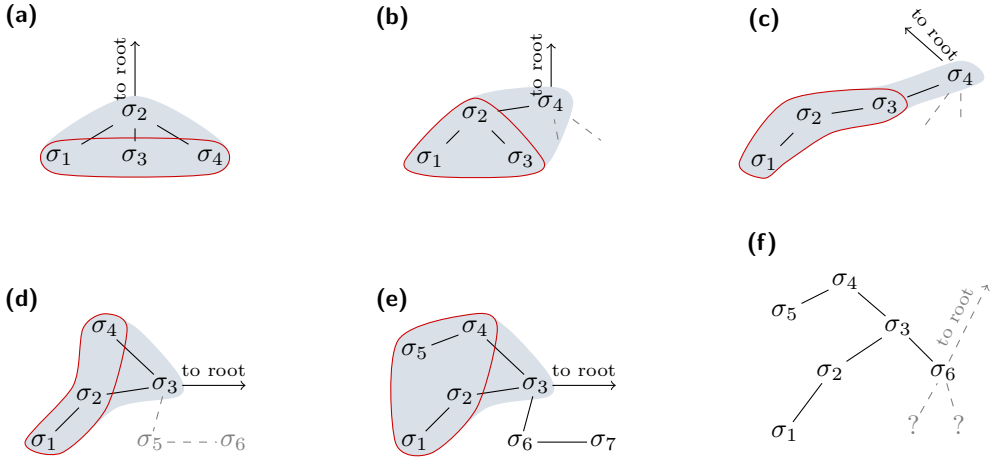


Figure 1 The possible configurations in the inductive step. The shaded region can be covered by one beacon, the circled tetrahedra are removed. Subfigure (f) has to be looked at independently.

Biro et al. [5] for (two-dimensional) polygons. We want to show the following

▶ **Hypothesis 3.1.** *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra, it always suffices to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Due to the length and number of cases, the proof is split up into various lemmas which are finally combined in Theorem 3.5.

Given the polyhedron P and a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra, we look at the dual graph $D(\Sigma)$ of the decomposition. We look at a spanning tree T of $D(\Sigma)$ rooted at some arbitrary leaf node because we want the dual graph to be a tree. This can only lead to more beacons being placed—never less. We will refer to nodes of T as well as their corresponding tetrahedra with σ_i—the meaning should be clear from the context.

The main idea of the proof is as follows: In a recursive way, we are going to place a beacon and remove tetrahedra until no tetrahedra are left. As will be shown, for each beacon we place, we can remove at least three tetrahedra. This yields the claimed upper bound. We will show this by induction on the number of tetrahedra. We first cover the base case:

▶ **Lemma 3.2.** *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma| \leq 4$ tetrahedra, it always suffices to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. If $m = 1$, then P is a tetrahedron and, due to convexity, no beacon is needed.

If $2 \leq m \leq 4$, we can apply Lemma 2.3 which shows that all tetrahedra share at least one common vertex v. Since v is contained in all tetrahedra and can thus attract and be attracted by all points in P, we are done by placing one beacon at v. ◀

We can now proceed with the inductive step, that is, polyhedra with a tetrahedral decomposition of $m > 4$ tetrahedra. Our goal is to place k beacons that are contained in at least $3k + 1$ tetrahedra and can therefore mutually attract all points in those tetrahedra. Afterwards, we will remove at least $3k$ tetrahedra, leaving a polyhedron with a tetrahedral decomposition of strictly less than m tetrahedra, to which we can apply the induction hypothesis. We then need to show how to route between the smaller polyhedron and the removed tetrahedra.

To do this, we look at a deepest leaf σ_1 of the spanning tree T. If multiple leaves with the same depth exist, we choose the one whose parent σ_2 has the largest number of children,

breaking ties arbitrarily. In Fig. 1, we can see different cases how the part of T that contains σ_1 and σ_2 might look like. We first show the inductive step for Figs. 1a to 1e. Note that in all five cases there needs to be at least one additional root node—either because we have strictly more than four tetrahedra or because the tree is required to be rooted at a leaf node. The inductive step for Fig. 1f will be dealt with in Lemma 3.4.

▶ **Lemma 3.3.** *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma| > 4$ tetrahedra and a spanning tree T of its dual graph $D(\Sigma)$ rooted at some arbitrary leaf node. Let σ_1 be a deepest leaf of T with the maximum number of siblings and let σ_2 be its parent. Assume further that either*

(1) the configuration in the neighborhood of σ_1 and σ_2 looks like any of Figs. 1a to 1d or
(2) the configuration in the neighborhood of σ_1 and σ_2 looks like Fig. 1e.

Then we can place one *beacon b at a vertex of σ_1 that is contained in at least four tetrahedra. We can then remove at least three tetrahedra containing b without violating the tree structure of T and while there is at least one tetrahedron left in T that contains b.*

Proof. We show this individually for the conditions.

(1) In all those cases the induced subgraph of the nodes σ_1, σ_2, σ_3, and σ_4 is connected. From Lemma 2.3(3) it follows that they share at least one vertex at which b is placed. Afterwards the circled tetrahedra are removed from T which preserves the tree structure of T. Additionally, at least one tetrahedron covered by b remains in T.

(2) Looking at Fig. 1e we see that we have three different sets, each containing σ_3, a child σ_i of σ_3, and σ_i's child: $\{\sigma_1, \sigma_2, \sigma_3\}$, $\{\sigma_5, \sigma_4, \sigma_3\}$, and $\{\sigma_7, \sigma_6, \sigma_3\}$. By Lemma 2.3(2) each set shares one edge giving us three edges of σ_3. Since at most two edges in any tetrahedron can be disjoint, at least two must share a common vertex. Without loss of generality, let these be the edges shared by $\{\sigma_1, \sigma_2, \sigma_3\}$ and $\{\sigma_5, \sigma_4, \sigma_3\}$. We can then place b at the shared vertex and afterwards remove σ_1, σ_2, σ_4, and σ_5. The beacon b is also contained in σ_3 which remains in T. ◀

Until now, we have ignored the configuration in Fig. 1f. The problem here is that, in general, to remove the tetrahedra σ_1 to σ_5, we need to place two beacons. Placing two beacons but only removing five tetrahedra violates our assumption that we can always remove at least $3k$ tetrahedra by placing k beacons. If we removed σ_6 and σ_6 had additional children, then T would no longer be connected which also leads to a non-provable situation. Thus, we need to look at the number and different configurations of the (additional) children of σ_6.

Since there are many different configurations of σ_6's children (and their subtrees) we decided to use a brute force approach to generate all cases we need to look at. Afterwards we removed all cases where Lemma 3.3 can be applied and all cases where only the order of the children differed. This leaves us with nine different cases and the following

▶ **Lemma 3.4.** *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma| > 4$ tetrahedra and a spanning tree T of its dual graph $D(\Sigma)$ rooted at some arbitrary leaf node. Let $T' \subseteq T$ be a subtree of T with height 3 for which Lemma 3.3 cannot be applied.*

In T' we can then place $k \geq 2$ beacons which are contained in at least $3k + 1$ tetrahedra and whose induced subgraph is connected.

We can then remove at least $3k$ tetrahedra from T', each of which contains a beacon, without violating the tree structure of T. After removal there is at least one tetrahedron left in T which contains one of the beacons. ◀

Due to space constraints we omit the proof which can, however, be found in [7, Lemma 5.9, pp. 34ff.]. There we also argue why either of the Lemmas 3.3 and 3.4 can always be applied.

We can now restate Hypothesis 3.1 as a theorem:

▶ **Theorem 3.5.** *Given a polyhedron P with a tetrahedral decomposition Σ with $m = |\Sigma|$ tetrahedra it always suffices to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. We show this by induction. The base case is shown by Lemma 3.2.

Look at a spanning tree T of the dual graph $D(\Sigma)$ rooted at an arbitrary leaf node. Let σ_1 be a deepest leaf node with the largest number of siblings, breaking ties arbitrarily. We can then apply either Lemma 3.3 or Lemma 3.4 and know at least the following:

(1) We have placed $k \geq 1$ beacons and removed at least $3k$ tetrahedra.
(2) Every removed tetrahedron contains at least one beacon.
(3) The induced subgraph of the beacons on the vertices and edges of P, i.e. the graph which contains only the beacons as vertices and the edges between two beacons, is connected.
(4) There is at least one beacon b contained in the remaining polyhedron P'.

From (1) it follows that the new polyhedron P' has a tetrahedral decomposition of $m' \leq m - 3k$ tetrahedra. We can then apply the induction hypothesis for P'. Thus we only need to place $k' = \lfloor \frac{m'+1}{3} \rfloor \leq \lfloor \frac{m-3k+1}{3} \rfloor = \lfloor \frac{m+1}{3} \rfloor - k$ beacons in P' to route between any pair of points in P'. Since $k' + k \leq \lfloor \frac{m+1}{3} \rfloor$ we never place more beacons than we are allowed.

With (2) to (4) we know that we can route from any point in P' to b, every point in the removed tetrahedra to at least one placed beacon, and between all placed beacons.

This completes the inductive step and thus, by induction, we have proved the theorem. ◀

4 A Lower Bound for Beacon-based Routing

We now want to show a lower bound for the number of beacons needed to route within polyhedra with a tetrahedral decomposition. To do this we first show a different lower bound proof for two dimensions which can then be easily applied to three dimensions. The idea for the following construction is similar to the one used by Shermer [11] for the lower bound for beacon-based routing in orthogonal polygons. We will present the results very briefly, for more details see [7].

We construct a class of outwards-spiraling polygons for which for every two additional vertices one additional beacon is needed. An example with $n = 12$ vertices and thus $\lfloor \frac{n}{2} \rfloor - 1 = 5$ needed beacons is shown in Fig. 2. It can easily be shown that it is not possible to route from s to t with less beacons, which gives the following

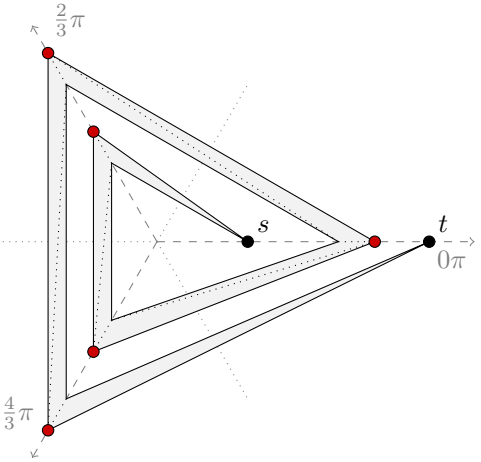


Figure 2 A spiral polygon with $n = 12$ vertices and $c = 5$ corners. Every such polygon needs $c = \frac{n}{2} - 1$ beacons to route from s to t.

▶ **Lemma 4.1.** *Given a spiral polygon with c corners and $n = 2c + 2$ vertices, c beacons are necessary to route from s to t.* ◀

The construction of the spiral polygon can be easily lifted to three dimensions by adding one vertex to each corner. This then directly leads to

▶ **Lemma 4.2.** *Given a spiral polyhedron with c corners and $n = 3c + 2$ vertices, c beacons are necessary to route from s to t.* ◀

5 A Sharp Bound for Beacon-based Routing

▶ **Theorem 5.1.** *Given a polyhedron P for which a tetrahedral decomposition with m tetrahedra exists, it is always sufficient and sometimes necessary to place $\lfloor \frac{m+1}{3} \rfloor$ beacons to route between any pair of points in P.*

Proof. In Theorem 3.5 we have shown that $\lfloor \frac{m+1}{3} \rfloor$ is an upper bound.

For any given m we can construct a spiral polyhedron P_m with $c = \lfloor \frac{m+1}{3} \rfloor$ corners for which, by Lemma 4.2, c beacons are necessary. The number of tetrahedra in P_m is $m' = 3c-1$ and this is also the smallest number of tetrahedra in any tetrahedral decomposition of P_m: If there was a tetrahedral decomposition with less tetrahedra then by Theorem 3.5 less than c beacons would be needed which contradicts Lemma 4.2.

If $m' < m$, i.e., due to the flooring function the c-corner spiral contains one or two tetrahedra less than m, we add the missing tetrahedra as if constructing a spiral polyhedron with $c + 1$ corners. This does not lead to less beacons being needed. ◀

References

1 I. Aldana Galván, J. L. Álvarez Rebollar, J. C. Catana Salazar, N. Marín Nevárez, E. Solís Villarreal, J. Urrutia, and C. Velarde. Beacon coverage in orthogonal polyhedra. In *29th Canadian Conference on Computational Geometry*, Ottawa, July 2017.

2 I. Aldana-Galván, J. L. Álvarez-Rebollar, J. C. Catana-Salazar, N. Marín-Nevárez, E. Solís-Villarreal, J. Urrutia, and C. Velarde. Covering orthotrees with guards and beacons. In *XVII Spanish Meeting on Computational Geometry (XVI ECG)*, Alicante, June 2017.

3 Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean geometry*, 4:47–123, 1995.

4 Michael Biro. *Beacon-Based Routing and Guarding.* PhD thesis, State University of New York at Stony Brook, 2013.

5 Michael Biro, Jie Gao, Justin Iwerks, Irina Kostitsyna, and Joseph S. B. Mitchell. Beacon-based routing and coverage. In *21st Fall Workshop on Computational Geometry*, 2011.

6 Bernard Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507, 1984.

7 Jonas Cleve. Combinatorics of beacon-based routing and guarding in three dimensions. Master's thesis, Freie Universität Berlin, Berlin, Germany, March 2017.

8 Subir Kumar Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, Cambridge, 2007. doi:10.1017/CBO9780511543340.

9 N. J. Lennes. Theorems on the simple finite polygon and polyhedron. *American Journal of Mathematics*, 33(1/4):37, 1911. doi:10.2307/2369986.

10 Jim Ruppert and Raimund Seidel. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete & Computational Geometry*, 1992. doi:10.1007/BF02187840.

11 Thomas C. Shermer. A combinatorial bound for beacon-based routing in orthogonal polygons. *arXiv preprint arXiv:1507.03509*, 2015.

Sequences of spanning trees for L_∞-Delaunay triangulations*

Prosenjit Bose[1], Pilar Cano[1,2], and Rodrigo I. Silveira[2]

1 **School of Computer Science, Carleton University**
jit@scs.carleton.ca
2 **Department de Matemàtiques, Universitat Politècnica de Catalunya**
{m.pilar.cano,rodrigo.silveira}@upc.edu

Abstract

We extend a known result about L_2-Delaunay triangulations to L_∞-Delaunay. Let $\mathcal{T}_S$ be the set of all non-crossing spanning trees of a planar n-point set S. We prove that for each element T of $\mathcal{T}_S$, there exists a length-decreasing sequence of trees $T_0, \ldots, T_k$ in the L_∞-metric such that $T_0 = T, T_k = MST_\square(S)$ and T_i does not cross T_{i-1} for all $i = 1, \ldots, k$, where $MST_\square(S)$ denotes the minimum spanning tree of S in the L_∞ metric. We also give an $\Omega(\log n)$ lower bound for the length of the sequence.

1 Introduction

The Delaunay triangulation is one of the most studied objects in computational geometry. In its most common form, L_2 or *circle-based*, it can be defined for a set of points $S \subset \mathbb{R}^2$ as a graph $DT_\bigcirc(S)$ where the vertex set is S, and an edge between two vertices u and v exists if and only if there exists a circle with u and v on its boundary containing no point of S in its interior. It is well-known that $DT_\bigcirc(S)$ is a triangulation if no four points of S are co-circular. The L_2-Delaunay triangulation has received a vast amount of attention. The aspects studied range from graph-theoretic properties like Hamiltonicity [5], or functionals for which it is optimal [6], to recent results on their spanning and routing properties [2, 7].

The definition of Delaunay graphs is easy to generalize by using some shape other than the circle that is required to be empty, such as a triangle, a square, or in fact any convex shape, leading to so-called *convex*-Delaunay graphs [4]. However, there are relatively few results on properties of convex Delaunay graphs for shapes other than circles, despite the fact that different shapes result in different properties of the Delaunay graphs. A recent result by Bonichon et al. [3] shows that, when the shape is a triangle, every plane triangulation can be realized as a triangle-based Delaunay triangulation, for some point set in the plane. Whereas, it is known that there exist triangulations that cannot be $DT_\bigcirc$-realizable.

For this reason, it is interesting to understand which properties of the Delaunay graphs depend on the circular shape, and what properties do not. In this paper we explore one of the properties of $DT_\bigcirc(S)$, and show that it also holds for the square or L_∞-Delaunay graph, denoted $DT_\square(S)$. Aichholzer et al. [1] showed that the minimum spanning tree of S can be obtained by repeatedly computing minimum spanning trees of constrained Delaunay triangulations. More precisely, the following iterative procedure converges to the minimum spanning tree of S. Start by computing an arbitrary spanning tree T_0 of S. Compute

* P. Bose is supported in part by NSERC. P.C. was supported by CONACyT. R. S. was supported by MINECO through the Ramón y Cajal program. P.C. and R.S. were also supported by projects MINECO MTM2015-63791-R and Gen. Cat. 2017SGR1640. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

the Delaunay triangulation of S taking the edges of T_0 as constraints. Next, compute the minimum spanning tree T_1 of that constrained triangulation, and repeat.

In this paper we show that the same occurs with squares, i.e., when the L_∞ metric is used, this procedure converges to the L_∞-minimum spanning tree of S. The main ingredient of our result is, as in [1], a fixed tree theorem (Theorem 3.1) that states that once one iteration of the above procedure does not produce a change, then it must have reached the minimum spanning tree of S. We note, however, that our proofs are different from those used for the L_2-metric, since several key lemmas in [1] rely on properties of circles.

1.1 Definitions

Let G be a plane geometric graph on S and let Q be an axis-aligned square. In the L_∞ metric, the set of points within a fixed distance from a given point is an axis-aligned square centered on the point. Recall that in $\mathbb{R}^n$, $||x||_\infty = \sup\{|x_i| : i \in \{1, \ldots, n\}\}$. The Delaunay graph of S with respect to the L_∞ metric, denoted as $DT_\square(S)$, contains an edge between p and q if and only if there is some homothet of Q with p and q on its boundary without any other point of S in its interior.

Given two vertices p and q, we denote by $Q(p, q)$ any homothet of Q with p and q on its boundary. Let $G = (S, E)$ where $E(G)$ is called a set of *constraints* and each element of $E(G)$ is called a *constraint*. We say that a point p is visible to q if no constraint crosses the line segment pq, and in that case we say that (p, q) is a *visible* edge in G. We define the *visibility graph* of G as the graph with vertices S and the set of all visible edges in G as the edge set. Notice that the visibility graph is always a connected graph. Also, that every constraint is a visible edge. We define $CDT_\square(G)$, the constrained L_∞-Delaunay graph of G, as follows. The constrained L_∞-Delaunay graph contains an edge between p and q if and only if pq is a constraint or there exists a $Q(p, q)$ such that there are no vertices of S in the interior of $Q(p, q)$ visible to both p and q. It is known that any *convex shape-$CDT(G)$* is a plane graph.

We say that p and q are *separated* in a square Q if p and q are in Q and there exists a constraint e crossing Q such that e divides Q into two different convex sets and with p and q in different sets, we refer to Figure 1. For any two points $p = (x_1, y_1)$ and $q = (x_2, y_2)$, we define the *width* (respectively *height*) between p and q as $w(p, q) = |x_1 - x_2|$ ($h(p, q) = |y_1 - y_2|$).

Let G be a graph with vertex set S and p, q be vertices in G and let P be a path from p to q. We will refer to P as a pq-path. We denote by $MST(G)$ the minimum spanning tree of the visibility graph of G. An important property of the classic Delaunay triangulation of a point set S is that $MST_\bigcirc(S) \subset DT_\bigcirc(S)$. The same property holds for L_∞-Delauny triangulation, i.e., $MST_\square(S) \subset DT_\square(S)$. We assume throughout that all edges in $S \times S$ have different lengths.

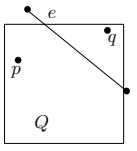


Figure 1 The points p and q are separated in Q by the edge e.

Let $\mathcal{T}_S$ be the set of all crossing-free spanning trees of S. For each element $T \in \mathcal{T}_S$ of S, we define the sequence $T_0, T_1, T_2, \ldots$ where $T_0 = T$, $T_i = MST(CDT_\square(T_{i-1}))$ for all $0 < i$. In this paper we will prove the convergence of this sequence to the $MST_\square(S)$. We begin with some helping lemmas.

2 Properties of L_∞-minimum spanning trees and L_∞-Delaunay graphs

The following properties will be essential for the proof of the fixed tree theorem.

▶ **Property 2.1.** *An edge $e \in G$ is not present in $MST(G)$ if and only if there is a path in G between e's endpoints that solely consists of edges shorter than e.*

The next two lemmas show that if there exists a visibility path between two points p and q contained in a square Q, then there exists a pq-path of the L_∞-constrained Delaunay triangulation contained in Q as well.

▶ **Lemma 2.2.** *Let p, q be two vertices such that (p, q) is visible in G. Then any $Q(p, q)$ contains a pq-path in $CDT_\square(G)$.*

The proof of Lemma 2.2 follows by induction on the number of vertices of S contained in $Q(p, q)$.

▶ **Lemma 2.3.** *Let Q be a square that contains p and q such that there is no edge separating p and q in Q, then there exists a pq-path of $CDT_\square(G)$ contained in Q.*

Proof. Since there is no edge separating p and q in Q there exists a pq-path in the visibility graph of G in Q. The proof follows from Lemma 2.2. ◀

Let $t = (u, v)$ be an edge of the $MST_\square(S)$ crossed by a set of constraints $BE_i(t)$ from T_i. Let Q_t be a smallest empty square containing u and v on its boundary, i.e., $||uv||_\infty = w(Q_t)$ and $u, v \in Q_t$. We say that $c \in BE_i(t)$ is a *diagonal edge* when c crosses t in consecutive sides of Q_t, or call it *vertical edge* otherwise. We say that c is the constraint nearest to u if it is the first constraint crossing t from u to v. Similarly c is the constraint nearest to v if it is the first constraint crossing t from v to u. The weight of a square Q corresponds to its sides length, denoted by $W(Q)$.

The following lemma basically states that if a constraint $c = (a, b)$ crossing t is vertical, then the distances $||av||_\infty, ||au||_\infty, ||bv||_\infty$ and $||bu||_\infty$ are shorter than $||ab||_\infty$, we refer to Figure 2.

▶ **Lemma 2.4.** *Let $T \in \mathcal{T}_S$, and let $t = (u, v) \in MST_\square(T)$ and Q_t defined as above. Let $c = (a, b)$ be a vertical edge crossing t. Then there exist squares $Q(a, v)$, $Q(b, v)$, $Q(a, u)$ and $Q(b, u)$ with weight than $||ab||_\infty$.*

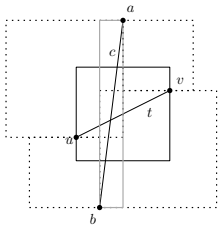


Figure 2 The dotted squares represent $Q(a, v)$, $Q(b, v)$, $Q(a, u)$ and $Q(b, u)$, which have shorter weight than $||ab||_\infty = h(a, b)$.

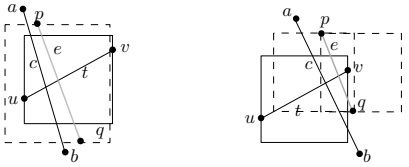
The next lemma shows a similar property fo diagonal edges.

▶ **Lemma 2.5.** *Let $T \in \mathcal{T}_S$, $t = (u, v) \in MST_\square(S)$ and Q_t defined as above. Let $c = (a, b) \in T$ be the nearest edge to v crossing t. If c is a diagonal edge such that c is crossing the side of Q_t containing v, then no edge from $CDT_\square(T)$ crosses t nearer to v than c.*

Proof. Assume by contradiction that there exists an edge $e = (p, q) \in CDT_\square(T)$ nearer to v than c. Then e is also diagonal and crosses the same sides of Q_t as c. Since e is diagonal and crosses the side of Q_t that contains v, the rectangle with diagonal e contains v in its interior. Hence, any square containing e's endpoints has v in its interior, we refer to Figure 3b. Since c was the nearest constraint to v crossing t, then any square containing e's endpoints has a visible point to both endpoints, which is a contradiction with e being in $CDT_\square(T)$. ◀

▶ **Lemma 2.6.** *Let $T \in \mathcal{T}_S$, $t = (u, v) \in MST_\square(S)$ and Q_t defined as above. Let $c = (a, b) \in T$ be the nearest edge to v crossing t. If c is a vertical edge and there exists an edge $e = (p, q)$ in $CDT(T)$ crossing t nearer to v than c, then e is shorter than c.*

Proof. Using the same arguments in the proof of Lemma 2.5 we notice that e cannot be a diagonal edge. Thus, e is vertical and crosses the same sides of Q_t as c, we refer to



(a) Vertical case. **(b)** Diagonal case.

Figure 3 Examples where edge e is a nearer to v than c.

Figure 3a. Without loss of generality, assume that e crosses top and bottom sides of Q_t. Let $Q_e = Q(p, q)$ be a smallest square defining e in $CDT_\square(T)$, then $W(Q_e) = ||e||_\infty$. Notice that since e crosses opposite sides of Q_t, then $||e||_\infty > ||uv||_\infty$. Hence, Q_e contains either u or v in its interior. This point is u otherwise any square containing e would have a visible point to both e's endpoints in its interior. Hence, c blocks u from both p and q. Thus, c crosses Q_e at opposite sides, otherwise Q_e would contain a visible point for both of p and q. Therefore, $||c||_\infty > ||e||_\infty$. ◀

3 Main result

3.1 Fixed tree theorem

▶ **Theorem 3.1.** *Let $T \in \mathcal{T}_S$. $T = MST_\square(CDT_\square(T))$ if and only if $T = MST_\square(S)$.*

Proof. The "if" part is trivial by definition of $MST_\square(S)$. Let us prove the "only if" part. Assume by contradiction that $T \neq MST_\square(S)$ then there exists an edge $t = (u, v) \in MST_\square(S)$ that does not belong to T. Since $t \notin T$ then $t \notin CDT_\square(T)$, otherwise t must be in $MST_\square(CDT_\square(T))$. Hence, there is at least one constraint crossing t in $CDT_\square(T)$.

Let $Q_t = Q(u, v)$ be a smallest square defining t in $DT_\square(S)$. Q_t exists since t is in $MST_\square$. Without loss of generality suppose that u and v belong to the left and right side of Q_t respectively. Thus each edge in $CDT_\square(T)$ crossing t crosses Q_t and has its endpoints outside Q_t. Let $c = (a, b) \in T$ be the nearest constraint to v and let Q_c be a square $Q(a, b)$ with sides of length $||c||_\infty$.

Case 1) c is diagonal. Let $Q_a = Q(a, v)$ be a square with size $||av||_\infty$ and $Q_b = Q(v, b)$ be a square with size $||vb||_\infty$.

If c crosses the right side of Q_t, then $w(a, b) = w(a, v) + w(v, b)$ and $h(a, b) = h(a, v) + h(v, b)$. Hence Q_a and Q_b have smaller size than Q_c. Since c is the nearest constraint to v, then by Lemma 2.5 there is no edge separating a from v in Q_a nor b from v in Q_b. By Lemma 2.3 there is a av-path, P_a and a vb-path, P_b, in Q_a and Q_b respectively. Therefore there is an ab-path in $P_a \cup P_b$ with edges solely shorter than c, contradicting our hypothesis by Property 2.1.

If c crosses the left side of Q_t then let $c' = (a', b')$ be the nearest constraint to u. The edge c' crosses the left side of Q_c as well, otherwise c' and c cross each other which is a contradiction. Thus, $w(a', b') = w(a', u) + w(u, b')$ and $h(a', b') = h(a', u) + h(a, b')$. Analogously as before we get a contradiction.

Case 2) c is vertical. By Lemma 2.4 there exist $Q_a^v = Q(a, v)$ and $Q_b^v = Q(b, v)$ with lower size than Q_c. Since c is the first constraint crossing t from v to u, by Lemma 2.6 there does not exist an edge separating a and v in Q_a^v nor an edge separating b and v in Q_b^v. From Lemma 2.3 there exists an av-paths P_a in Q_a^v and a vb-path P_b in Q_b^v. Hence, there exists

a path in $P_a \cup P_b$ from a to b with edges solely shorter than c which is a contradiction by Property 2.1. Therefore $T = MST_{\square}(S)$. ◀

Consider an arbitrary tree $T \in \mathcal{T}_S$ and a sequence $T_0, T_1, \ldots$, such that $T_0 = T, T_i = MST(CDT_{\square}(T_{i-1}))$ for all $i \geq 1$. Notice that this is a length-decreasing sequence, since T_i is shorter than T_{i-1} unless both are identical trees. Also, by definition of T_i, the trees T_i and T_{i-1} do not cross since they belong to the same plane graph, namely $CDT_{\square}(T_{i-1})$, for all $i > 1$. As a consequence of the Fixed tree theorem we obtain a length-decreasing sequence of trees in $\mathcal{T}_S$ which reaches a fixed point $T_k = MST_{\square}(S)$ in a finite number of steps.

▶ **Theorem 3.2.** *For any $T \in \mathcal{T}_S$ there exists a sequence $T_0, T_1, \ldots, T_k$ such that $T_0 = T$ and $T_k = MST_{\square}(S)$.*

Figure 4 shows a sequence $T_0, T_1, \ldots, T_k$ such that $T_0 = T$, $T_i = MST_{\square}(T_{i-1})$ and $T_k = MST_{\square}(S)$ for all $1 \leq i \leq k$ which converges in 4 steps.

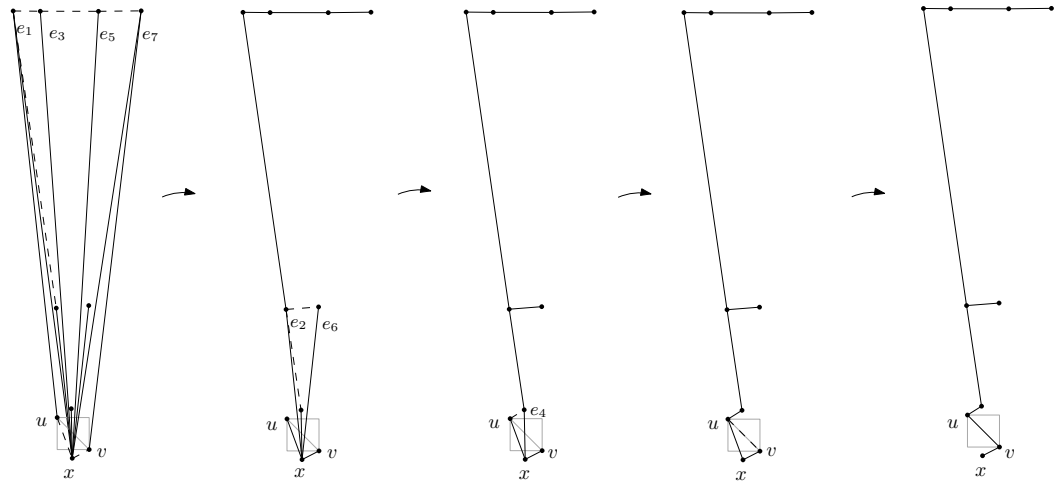


Figure 4 Example of a sequence with a spanning tree of a 10-point set that converges in 4 steps. The dashed edges represent the appearing edges at stage $i + 1$.

3.2 Lower bound

A natural question, once we know that this sequence converges, is how fast it reaches the $MST_{\square}(S)$. As a first step for answering this question we give a lower bound based on a construction shown in Figure 4, similar to the one given in [1], which has length $\Theta(\log n)$. The edge $t = (u, v)$ is an edge of the $MST_{\square}(S)$ where t is a diagonal of the square Q that defines the edge t in $DT_{\square}(S)$. Let $n = 2^m + 2$ for some $m \in \mathbb{N}$. Let x be a point below Q such that all the $n - 1$ edges of the initial tree T are incident to x and the edges that are not incident to u and v are vertical edges, i.e., edges crossing the top and bottom sides of Q_t. We order the vertical edges $e_1, e_2, \ldots e_{n-3}$ from left to right, refer to Figure 4. The edges with odd index have length ℓ. For $i = 2, \ldots, m$, the length of the edge e_j where $j \equiv 2i$ modulo 2^{i+1} is $\frac{\ell}{3^{i-1}}$. Notice that at step $i - 1$, the longest edges of T_{i-1} are the edges with length $\frac{\ell}{3^{i-1}}$ and these edges are the only ones of T_{i-1} not present at T_i for $1 \leq i < m$. Indeed, let $e_k = (a, x)$ be a vertical edge crossing t in T_{i-1} where $k \not\equiv 2i$ modulo 2^{i+1}, then $||e||_\infty \leq \frac{\ell}{3^{i-1}}$. Then any xa-path different from (x, a) would contain an edge with length $\frac{2\ell}{3^{i-1}}$, refer to Figure 5. Also, the edges e_j where $j \equiv 2i$ modulo 2^{i+1} disappear at stage $i - 1$, since there

exists a path between e_j's endpoints with edges shorter than e_j, refer to Figure 5. Hence the tree undergoes $m + 1 = \log_2(n-2) + 1$ iterations.

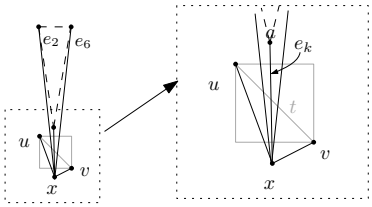


Figure 5 The dashed edges have length $||e||_\infty = \frac{2\ell}{3^{i-1}}$ and all the xa-paths different from (x, a) contain one of the endpoint different from x of e_2 or e_6.

▶ **Theorem 3.3.** *There exists a point set S and $T \in \mathcal{T}_S$, such that the sequence $T_0, T_1, \ldots, T_k$ with $T_0 = T, T_i = MST_\square(CDT_\square(T_{i-1}))$ and $T_k = MST_\square(S)$ has length $\Theta(\log n)$.*

4 Conclusions

We have extended the convergence of sequences of crossing-free spanning trees for the L_2 metric [1] to the L_∞ metric. We have also given an $\Omega(\log n)$ lower bound for the maximum length of these sequences. In the full version of this paper, which is in preparation, we show that every sequence $T_0, \ldots, T_k$ has length $O(\log n)$, but due to space limitations we cannot include more details here. We also believe that the same techniques can be applied to other Delaunay triangulations defined by regular polygonal shapes. However, the difficulty is that we have to consider many different cases because of the different variations of edges crossing such polygons (for squares we only had to consider *vertical* and *diagonal* edges).

Acknowledgments. The authors thank Vera Sacristán for useful comments and discussions.

References

1 Oswin Aichholzer, Franz Aurenhammer, and Ferran Hurtado. Sequences of spanning trees and a fixed tree theorem. *Computational Geometry*, 21(1):3–20, 2002.

2 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Ljubomir Perkovic, and André van Renssen. Upper and lower bounds for online routing on Delaunay triangulations. *Discrete & Computational Geometry*, 58(2):482–504, 2017.

3 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and David Ilcinkas. Connections between Theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 266–278. Springer, 2010.

4 Prosenjit Bose, Paz Carmi, Sébastien Collette, and Michiel Smid. On the stretch factor of convex Delaunay graphs. *Journal of computational geometry*, 1(1):41–56, 2010.

5 Michael B Dillencourt. Toughness and Delaunay triangulations. *Discrete & Computational Geometry*, 5(6):575–601, 1990.

6 Oleg R Musin. Properties of the Delaunay triangulation. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 424–426. ACM, 1997.

7 Ge Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM Journal on Computing*, 42(4):1620–1659, 2013.

Maximizing Ink in Symmetric Partial Edge Drawings of k-plane Graphs

Michael Höller, Fabian Klute, Soeren Nickel, Martin Nöllenburg, and Birgit Schreiber

Algorithms and Complexity Group, TU Wien, Vienna, Austria
[fklute|noellenburg]@ac.tuwien.ac.at, firstname.lastname@student.tuwien.ac.at

Abstract

Partial edge drawing (PED) is a drawing style for non-planar graphs, in which edges are drawn only partially as pairs of opposing stubs on the respective end-vertices. In a PED, by erasing the central parts of edges, all edge crossings and the resulting visual clutter are hidden in the undrawn parts of the edges. We study symmetric partial edge drawings (SPEDs), in which the two stubs of each edge are required to have the same length. It is known that maximizing the ink (or the total stub length) when transforming a straight-line drawing with crossings into a SPED is tractable for 2-plane input drawings, but generally NP-hard. We show that the problem remains NP-hard even for 3-plane input drawings. Yet, for k-plane input drawings whose edge intersection graph forms a collection of trees or cacti we present efficient algorithms for ink maximization.

1 Introduction

Visualizing non-planar graphs as node-link diagrams is challenging due to the visual clutter caused by edge crossings. The layout readability deteriorates as the edge density and thus the number of crossings increases. Therefore alternative layout styles are necessary for non-planar graphs. A radical approach first used in applied network visualization work by Becker et al. [1] is to start with a traditional straight-line graph drawing and simply drop a large central part of each edge and with it many of the edge crossings. This idea relies on the closure and continuation principles in Gestalt theory, which imply that humans can still see a full line segment based only on the remaining edge stubs by filling in the missing information in our brains. User studies have confirmed that such drawings remain readable while reducing clutter significantly [6, 7].

The idea of drawing edges only partially has been formalized in graph drawing as follows [5]. A *partial edge drawing (PED)* is a graph drawing that maps vertices to points and edges to pairs of crossing-free edge stubs of positive length pointing towards each other. These edge stubs are obtained by erasing one contiguous central piece of the straight-line segment connecting the two endpoints of each edge. In other words each straight-line edge is divided into three parts, of which only the two outer ones are drawn (see Fig. 1). More restricted and better readable [2] variations of PEDs are *symmetric* PEDs, in which both stubs of an edge must have the same length (see Fig. 1(b)), and *homogeneous* PEDs, in which the ratio of the stub length to the total edge length is the same for all edges. Symmetric stubs facilitate finding adjacent vertices due to the identical stub lengths at both vertices, and symmetric homogeneous stubs additionally indicate the distance at which to find a neighboring vertex. The natural optimization problem in this formal setting is *ink*

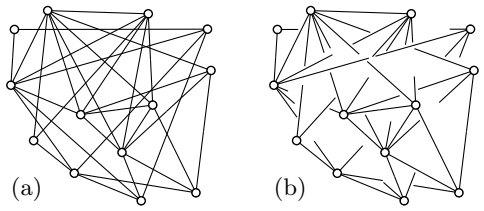


Figure 1 A straight-line graph drawing (a) and a maximum-ink symmetric partial edge drawing (b) of the same graph.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

maximization, i.e., maximizing the total stub length, so that as much information as possible is given in the drawing while all crossings disappear in the negative background space.

We study the ink maximization problem for symmetric partial edge drawings (SPEDs) with a given geometric input drawing. This problem is known as MaxSPED. Bruckdorfer and Kaufmann [5] presented an integer linear program for solving MaxSPED. Later, Bruckdorfer et al. [4] gave an $O(n \log n)$-time algorithm for MaxSPED on the class of 2-plane input drawings (no edge has more than two crossings), where n is the number of vertices, and an efficient 2-approximation algorithm for the dual problem of minimizing the amount of erased ink for arbitrary input drawings. Bruckdorfer [3] further gives an NP-hardness proof for MaxSPED.

Contribution. We extend the results of Bruckdorfer et al. [4] on 2-plane geometric graph drawings to k-plane graph drawings for $k > 2$. In particular, we show that MaxSPED is NP-hard even for 3-plane input drawings. However, for k-plane graph drawings whose edge intersection graphs are collections of trees or cacti (which have maximum degree k), we give polynomial-time algorithms for solving MaxSPED.

2 Preliminaries

Throughout the paper let G be a *simple graph* with edge set $S = \{s_1, \ldots, s_m\}$ and Γ a straight-line drawing of G in the plane. We call Γ *k-plane* if every edge $s_i \in S$ is crossed by at most k other edges from S in Γ. We use the terms edge in S and segment in Γ interchangeably. Hence we can also interpret S as a set of line segments.

The *intersection graph* $C = (V, E)$ of Γ is the graph containing a vertex v_i in V for every $s_i \in S$ and an edge $v_iv_j \in E$ between vertices $v_i, v_j \in V$ if the corresponding edges $s_i, s_j \in S$ intersect in Γ. We also denote the segment in S corresponding to a vertex $v \in V$ by $s(v)$. Observe that the intersection graph C of a k-plane drawing Γ has maximum degree k. Using a standard sweep-line algorithm, computing the intersection graph C of a set of m line segments takes $O(m \log m + |E|)$ time [8], where $|E|$ is the number of intersections.

A *symmetric partial edge drawing* (SPED) D of Γ draws a fraction $0 < f_s \leq 1$ of each edge $s = uv \in S$ by drawing two symmetric edge stubs at u and v of length $f_s \cdot |s|/2$ each. No two stubs in D may intersect. The *ink* or *ink value* $I(D)$ of a SPED D is the total stub length $I(D) = \sum_{s \in S} f_s|s|$. In the problem MaxSPED, the task is to find for a given drawing Γ a SPED D^* such that its ink $I(D^*)$ is maximum over all SPEDs.

3 Hardness of MaxSPED for $k \geq 3$

In this section we close the gap between the known hardness of MaxSPED [3] and the polynomial-time algorithm for 2-plane drawings [4] as stated in the following theorem.

▶ **Theorem 3.1.** MaxSPED *is* NP*-hard even for 3-plane graph drawings.*

Proof. We reduce from the NP-hard problem Planar 3-Sat [9] using similar ideas as in Bruckdorfer's sketch of the hardness proof for general MaxSPED [3]. Here we specify precisely the maximum ink contributions of all gadgets needed for a satisfying variable assignment. Our variable gadgets are cycles of edge pairs that admit exactly two maximum-ink states. Finally we construct clause gadgets consisting of three pairwise intersecting edges so that all crossings are between two edges only, while Bruckdorfer's gadgets have multiple edges intersecting in the same point. Let ϕ be a planar 3-Sat formula with n variables $\{x_1, \ldots, x_n\}$ and m clauses $\{c_1, \ldots, c_m\}$, each consisting of three literals. We can assume

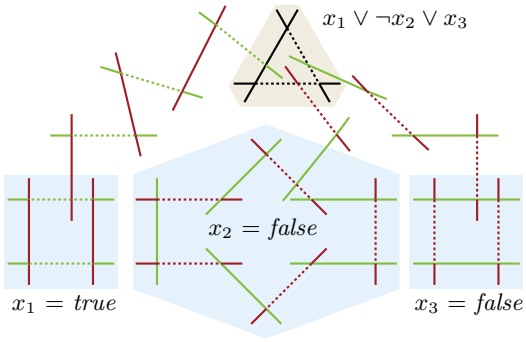


Figure 2 Example of three variable gadgets and a satisfied clause gadget. Dotted parts do not belong to the SPED.

that ϕ comes with a planar drawing of its variable-clause graph H_ϕ, which has a vertex for each variable x_i and a vertex for each clause c_j. Each clause vertex is connected to the three variables appearing in the clause. In the drawing of H_ϕ all variable vertices are placed on a horizontal line and the clause vertices connect to the adjacent variable vertices either from above or from below the horizontal line. In our reduction (see Fig. 2) we mimic the drawing of H_ϕ by creating a 3-plane drawing Γ_ϕ as a set of line segments of uniform length and a value L such that Γ_ϕ has a SPED with ink at least L if and only if ϕ is satisfiable.

All segments in the gadgets are of length 5. We use pairs of intersecting segments, alternatingly colored red and green. The intersection point of each red-green segment pair is at distance 1 from an endpoint. Thus, the maximum amount of ink contributed by such a pair is 7 (one full segment of length 5 and the other one with two stubs of length 1 each).

Each variable gadget is a cycle of segment pairs, with (at least) one pair for each occurrence of the variable in ϕ, see Fig. 2. Observe that this cycle has exactly two ink-maximal SPEDs: either all red edges are full segments and all green edges are length-1 stubs or vice versa. We associate the configuration with green stubs and full red segments with the value *true* and the configuration with full green segments and red stubs with the value *false*.

For each clause we construct a triple of mutually intersecting segments, see the gadget on yellow background in the upper part of Fig. 2. Again, their intersection points are at distance 1 from the endpoints. It is clear that in such a clause triangle at most one of the three segments can be fully drawn, while the stubs of the other two can have length at most 1. Hence, the maximum amount of ink in a SPED contributed by a clause gadget is 9.

Finally, we connect variable and clause gadgets in such a way that a clause gadget can contribute its maximum ink value of 9 if and only if the clause is satisfied by the selected truth assignment to the variables. For a positive (negative) literal, we create a path of even length between a green (red) edge of the variable gadget and one of the three edges of the clause gadget as shown in Fig. 2. The first edge s of this path intersects the corresponding variable edge s' such that s' is split into a piece of length 2 and a piece of length 3, whereas s is split into a piece of length 1 and a piece of length 4. The last edge of the path intersects the corresponding clause edge with the same length ratios. The path itself consists of a chain of red-green segment pairs, so each pair contributes an ink value of at most 7.

Since the drawing of H_ϕ has polynomial size, the number of segments created in the reduction is polynomial. Further, no segment intersects more than three other segments, so the constructed drawing is 3-plane.

For the correctness of the reduction, let L be the ink value obtained by counting 7 for each red-green segment pair and 9 for each clause gadget. First assume that ϕ has a satisfying

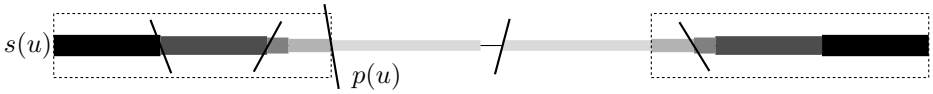


Figure 3 A segment $s(u)$ with five intersecting segments and the different induced stub lengths. The boxed stub lengths are considered in $short(u)$ and do not affect $p(u)$.

truth assignment and put each variable gadget in its corresponding state. For each clause, select exactly one literal with value *true* in the satisfying truth assignment. We draw the clause segment that connects to the selected literal as a full segment and the other two as length-1 stubs. Recall that the literal paths are oriented from the variable gadget to the clause gadget. Since the last segment of the selected literal path must be drawn as length-1 stubs, the only way of having a maximum contribution of that path is by alternating stubs and full segments. Hence, the first segment of the path must be a full segment. But because the variable is in the state that sets the literal to *true*, the intersecting variable segment is drawn as two stubs and the path configuration is valid. For the two non-selected literals, we can draw the last segments of their paths as full segments, as well as every segment at an even position, while the segments at odd positions are drawn as stubs. This is compatible with any of the two variable configurations and proves that we can indeed achieve ink value L.

Conversely, assume that we have a SPED with ink value L. By construction, every red-green segment pair and every clause gadget must contribute its respective maximum ink value. In particular, each variable gadget is either in state *true* or *false*. By design of the gadgets it is straight-forward to verify that the corresponding truth assignment satisfies ϕ. ◀

4 Two polynomial special cases

Section 3 showed that MaxSPED is generally NP-hard for $k \geq 3$. Now we consider the special case that the intersection graph of the k-plane input drawing is a tree or a cactus. In both cases we present polynomial-time dynamic programming algorithms. Let $C = (V, E)$ be the intersection graph of a given drawing Γ of a graph G as defined in Section 2. Let $u \in V$ and $\delta = \deg(u)$. Then for the corresponding segment $s(u) \in S$ there are $\delta + 1$ relevant stub pairs including the whole segment, see Fig. 3. Let $\ell_1(u), \ldots, \ell_\delta(u) \in \mathbb{R}_+$ be the stub lengths induced by the intersection points of $s(u)$ with the segments of the neighbors of u, sorted from shorter to longer stubs. We define $\ell_0(u)$ as the length of the whole segment $s(u)$.

4.1 Trees

Here we assume that $C = (V, E)$ is a rooted tree of maximum degree k. We give a bottom up dynamic programming algorithm for solving MaxSPED on C. For each vertex $u \in V$ we compute and store the maximum ink values $T_i(u)$ for $i = 0, \ldots, \delta$ with $\delta = \deg(u)$ for the subtree rooted at u such that $s(u)$ is drawn as a pair of stubs of length $\ell_i(u)$. For $u \in V$ let $p(u)$ denote the parent of u in C and let $c(u)$ denote the set of its children. For $u \in V$ let i_p be the index of the stub length $\ell_{i_p}(u)$ induced by the intersection point of $s(u)$ and $s(p(u))$. We define the following two values, which allow us to categorize the stub lengths into those not affecting the stubs of the parent and those that do affect the parent:

$$short(u) = \max\{T_1(u), \ldots, T_{i_p}(u)\}$$
$$long(u) = \max\{T_0(u), \ldots, T_\delta(u)\}.$$

Figure 3 highlights the stub lengths that are considered in $short(u)$. We recursively define

$$T_i(u) = \ell_i(u) + \sum_{v \in c(u)} \begin{cases} short(v) & \text{if } s(u) \text{ with length } \ell_i(u) \text{ intersects } s(v) \\ long(v) & \text{otherwise.} \end{cases} \tag{1}$$

The correctness of Recurrence (1) follows by induction. For a leaf u in C the set $c(u)$ is empty and the correctness of $T_i(u)$ is immediate. Further, $short(u) = T_1(u)$ and $long(u) = T_0(u)$ are set correctly for the parent $p(u)$. For an inner vertex u with degree δ we can assume by the induction hypothesis that the values $short(v)$ and $long(v)$ are computed correctly for all children $v \in c(u)$. Each value $T_i(u)$ for $0 \leq i \leq \delta$ is then the stub length $\ell_i(u)$ plus the sum of the maximum ink we can achieve among the children subject to the stubs of u being drawn with length $\ell_i(u)$. Setting $long(u)$ and $short(u)$ as above yields the two maximum ink values that are relevant for $p(u)$.

Recurrence (1) can be solved naively in $O(mk^2)$ time, where $m = |V|$. Using the order on the stub lengths we can improve this to $O(mk)$ time by computing all $T_i(u)$ for one $u \in V$ in $O(k)$ time. Let $u \in V$ be a vertex with degree $\deg(u) = \delta$. The values $T_0(u) = \ell_0(u) + \sum_{v \in c(u)} short(v)$ and $T_1(u) = \ell_1(u) + \sum_{v \in c(u)} long(v)$ for the whole segment $s(u)$ and the shortest stubs can be computed in $O(k)$ time each. Now $T_{j+1}(u)$ can be computed from $T_j(u)$ in $O(1)$ time as follows. Let v_j be the neighbor of u that induces stub length $\ell_j(u)$ and assume $v_j \neq p(u)$. In $T_j(u)$ we could still count the value $long(v_j)$, but in $T_{j+1}(u)$ the stub length of u implies that v_j can contribute only to $short(v_j)$. Then $T_{j+1}(u) = T_j(u) - long(v_j) + short(v_j)$. If $v_j = p(u)$, then the two values $T_j(u)$ and $T_{j+1}(u)$ are equal as the corresponding change in stub length has no effect on the children of u. Computing $short(u)$ and $long(u)$ clearly takes $O(k)$ time.

So by solving Recurrence (1) in $O(mk)$ time we find an optimal solution to the MaxSPED problem on G with drawing Γ using standard backtracking from $\max\{T_0(r), \ldots, T_{\deg(r)}(r)\}$ for the root r of C. Since the intersection graph C is a tree with $O(m)$ edges it can be computed in $O(m \log m)$ time and we obtain the following theorem.

▶ **Theorem 4.1.** *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph $C = (V, E)$ of G is a tree with maximum degree $k \in \mathbb{N}$, then problem* MaxSPED *can be solved in $O(mk + m \log m)$ time and space.*

4.2 Cactus graphs

We now generalize our previous algorithm to cactus graphs. Due to space constraints we provide only a sketch of the arguments and omit further details. Let the intersection graph $C = (V, E)$ be a *cactus graph*, i.e. a simple graph in which no edge $e \in E$ is part of more than one cycle. For a cactus graph C the *block-cut tree* is an arbitrarily rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ contains a node for every cycle and for every cut vertex in C. We call a node $u \in \mathcal{V}$ a *block node* if it represents an induced cycle on vertices $V(u) \subseteq V$ and an *articulation node* of the cut vertex $a(u) = v \in V$ otherwise. We have an edge $uv \in \mathcal{E}$ between a block node u and an articulation node v whenever $a(v) \in V(u)$ and an edge $uv \in \mathcal{E}$ between two articulation nodes whenever $a(u)a(v) \in E$. We re-use the notation $p(u)$ and $c(u)$ to refer to the parent and to the children of a node $u \in \mathcal{V}$.

We further define for each node $u \in \mathcal{V}$ two indices $i_p(u) \leq i_q(u)$. For an articulation node u, i_p and i_q are the indices corresponding to the stub lengths induced by the intersection points of $s(a(u))$ and its neighbors in C among the vertex set of $p(u)$. For a block node u, i_p and i_q are the indices corresponding to the intersection points of the parent's segment

$s(a(p(u)))$ with its neighbors in $V(u)$. Similar to Section 4.1 we now define three values for each articulation node $u \in \mathcal{V}$ with degree δ:

$$\begin{aligned} \mathit{long}(u) &= \max\{T_0(u), \dots, T_\delta(u)\} \\ \mathit{mid}(u) &= \max\{T_1(u), \dots, T_{i_q}(u)\} \\ \mathit{short}(u) &= \max\{T_1(u), \dots, T_{i_p}(u)\}. \end{aligned}$$

Further, we define the recurrence $T_i(u)$ for all articulation nodes $u \in \mathcal{V}$:

$$T_i(u) = \ell_i(a(u)) + \sum_{v \in c(u)} \begin{cases} T_{i_p(v)}(v) & \text{if } 1 \le i < i_p(v) \\ T_{i_q(v)}(v) & \text{if } i_p(v) < i \le i_q(v) \\ T_0(v) & \text{otherwise.} \end{cases} \tag{2}$$

For a block node $u \in \mathcal{V}$ we cut open the cycle it represents at vertex $a(p(u))$ (refer to Bruckdorfer et al. [4] for details) and consider the three relevant stub lengths of the parent's segment $s(a(p(u)))$ induced by $i_p(u)$ and $i_q(u)$ and the whole segment. The resulting subgraph is a path on $V(u)$ with attached children that are articulation vertices in $\mathcal{T}$, so we can apply our algorithm of Section 4.1 with minor modifications. Once all values in the subgraph of block node u are computed, we can derive the values $T_i(u)$ for the block node u.

Correctness can be proved by modifying the inductive arguments of Section 4.1 accordingly. Combining the linear running time for cycles of Bruckdorfer et al. [4] with the running time for trees of Theorem 4.1 we can argue that cactus graphs can in fact be handled in the same time and space bounds as trees.

▶ **Theorem 4.2.** *Let G be a simple graph with m edges and Γ a straight-line drawing of G. If the intersection graph $C = (V, E)$ of G is a cactus with maximum degree $k \in \mathbb{N}$, problem* MaxSPED *can be solved in $O(mk + m \log m)$ time and space.*

References

1 Richard A. Becker, Stephen G. Eick, and Allan R. Wilks. Visualizing network data. *IEEE Trans. Visualization and Computer Graphics*, 1(1):16–28, 1995.

2 Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani, and Alessandra Tappini. Partial edge drawing: Homogeneity is more important than crossings and ink. In *Information, Intelligence, Systems Applications (IISA'16)*, pages 1–6. IEEE, 2016.

3 Till Bruckdorfer. *Schematics of Graphs and Hypergraphs.* PhD thesis, Uni Tübingen, 2015.

4 Till Bruckdorfer, Sabine Cornelsen, Carsten Gutwenger, Michael Kaufmann, Fabrizio Montecchiani, Martin Nöllenburg, and Alexander Wolff. Progress on partial edge drawings. *J. Graph Algorithms Appl.*, 21(4):757–786, 2017.

5 Till Bruckdorfer and Michael Kaufmann. Mad at edge crossings? Break the edges! In *Fun with Algorithms (FUN'12)*, volume 7288 of *LNCS*, pages 40–50. Springer, 2012.

6 Till Bruckdorfer, Michael Kaufmann, and Simon Leibßle. PED user study. In *Graph Drawing (GD'15)*, volume 9411 of *LNCS*, pages 551–553. Springer, 2015.

7 Michael Burch, Corinna Vehlow, Natalia Konevtsova, and Daniel Weiskopf. Evaluating partially drawn links for directed graph edges. In *Graph Drawing (GD'11)*, volume 7034 of *LNCS*, pages 226–237. Springer, 2012.

8 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer, 3rd edition edition, 2008.

9 David Lichtenstein. Planar formulae and their uses. *SIAM J Comput*, 11(2):329–343, 1982.

Time-Space Trade-Offs for Computing Euclidean Minimum Spanning Trees*

Bahareh Banyassady[1], Luis Barba[2], and Wolfgang Mulzer[1]

1 **Freie Universität Berlin, Berlin, Germany**
[bahareh, mulzer]@inf.fu-berlin.de
2 **ETH Zurich, Zurich, Switzerland**
luis.barba@inf.ethz.ch

Abstract

Given n sites in the plane, their *Euclidean minimum spanning tree* (EMST), is the minimum spanning tree with the sites as vertices, where the weight of the edge between two sites is their Euclidean distance. In this paper, we revisit this problem, and design algorithms to compute the EMST in a limited-workspace model. In this model the input of size n lies in a random access read-only memory. The output has to be reported sequentially, and it cannot be accessed or modified. In addition, there is a read-write *workspace* of $O(s)$ words, where $s \in \{1, \ldots, n\}$ is a given parameter. We present an algorithm that computes EMST using $O(n^3 \log s/s^2)$ time and $O(s)$ words of workspace. Using the fact that EMST is a subgraph of the bounded-degree *relative neighborhood graph* (RNG), we apply Kruskal's MST algorithm on RNG. To achieve this with limited workspace, we introduce a compact representation of planar graphs, called an *s-net* which allows us to manipulate RNG's component structure during the execution of the algorithm.

1 Introduction

A significant amount of research was focused on the design of algorithms using few variables. Many of them dating from the 1970s, when memory used to be an expensive commodity. While in recent days the cost has been substantially reduced, the amount of data has increased, and the size of some devices has been dramatically reduced. Sensors and small devices, where larger memories are neither possible nor desirable, have proliferated in recent years. Moreover, even if a device is procured with a large memory, it might still be preferable to limit the number of write operations, since they are slow and costly. Therefore, while many memory-constrained models exist, the general scheme is the following: the input resides in a read-only memory where data cannot be modified by the algorithm. The algorithm is allowed to store a few variables to solve the problem. These variables reside in a local memory and can be modified as needed (usually called *workspace*). Since the output may also not fit in our local memory, the model provides us with a write-only memory where the desired output is sequentially reported by the algorithm.

In general, one might consider algorithms that are allowed to use a workspace of $O(s)$ *words* for some parameter s, where a word is a collection of $\theta(\log n)$ bits. The goal is then to design algorithms whose running time decreases as s increases, and that provides a nice trade-off between workspace size and running time.

Asano et al. [1] proposed an algorithm to compute the EMST of a set of n given sites in $O(n^3)$ time using a workspace of $O(1)$ words. In this paper, we provide an algorithm that computes the EMST in $O(n^3 \log s/s^2)$ time using $O(s)$ words of workspace. This algorithm provides a smooth transition between the $O(n^3)$ time algorithm [1] with constant words of workspace and the $O(n \log n)$ time algorithm [2] using a workspace of $O(n)$ words.

* Supported in part by DFG project MU/3501/2 and by the ETH Postdoctoral Fellowship.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

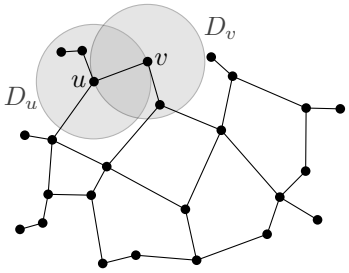


Figure 1 The graph RNG for a set of sites. The disk D_u (resp. D_v) is centered at u (resp. v) and passes through v (resp. u). The edge uv is in RNG, since there is no site in the lens $D_u \cap D_v$.

2 Preliminaries and Definitions

Let V be a set of n points (sites) in the plane. The *Euclidean minimum spanning tree* of V, EMST(V), is the minimum spanning tree of the complete graph G on V, where the edges are weighted by the Euclidean distance between their endpoints. We assume that V is in general position, i.e., the edge lengths in G are pairwise distinct, thus EMST(V) is unique. Given V, we can compute EMST(V) in $O(n \log n)$ time using $O(n)$ words of workspace [2].

The *relative neighborhood graph* of V, RNG(V), is the undirected graph with vertex set V obtained by add an edge between any two sites $u, v \in V$ if and only if the intersection of the two disks centered at u or v and passing through the other one, which is called the *lens* of u and v, is empty of sites in V [7]; see Figure 1. A plane embedding of RNG(V) is obtained by the straight line drawing of the edges. Furthermore, the maximum degree of RNG(V) is six and so, the number of edges of RNG(V), which is denoted by m, is $O(n)$. It is well-known that EMST(V) is a subgraph of RNG(V). This implies that RNG(V) is connected. Given V, we can compute RNG(V) in $O(n \log n)$ time using $O(n)$ words of workspace [5–7].

Recall the classic algorithm by Kruskal to find EMST(V) [4]: start with an empty forest T, and consider the $m = O(n)$ edges of RNG(V) one by one, by increasing weight. In each step, insert the current edge $e = vw$ into T iff there is no path between v and w in T. In the end, T is EMST(V). This takes $O(n \log n)$ total time and $O(n)$ words of workspace.

Let $s \in \{1, \ldots, n\}$ be a parameter, and let V be a set of n sites in general position (as above) in a read-only array. The goal is to find EMST(V), with $O(s)$ words of workspace. We use RNG(V) in order to compute EMST(V). By general position, the edge lengths in RNG(V) are pairwise distinct. Thus, we define $E_R = e_1, \ldots, e_m$ to be the sorted sequence of the edges in RNG(V), in increasing order of length. For $i \in \{1, \ldots, m\}$, we define RNG_i to be the subgraph of RNG(V) with vertex set V and edge set $\{e_1, \ldots, e_{i-1}\}$.

In the limited workspace model, we cannot store RNG_i explicitly. Instead, we resort to the *computing instead of storing* paradigm [1]. That is, we completely compute the next batch of edges in E_R whenever we need new edges of RNG(V) in Kruskal's algorithm. To check whether a new edge $e_i \in E_R$ belongs to EMST(V), we need to check if e_i connects two distinct components of RNG_i. To do this with $O(s)$ words of workspace, we will use a succinct representation of its component structure; see below. In our algorithm, we represent each edge $e_i \in E_R$ by two directed *half-edges*. The two half-edges are oriented in opposite directions such that the face incident a half-edge lies to the left of it. Obviously, each half-edge in RNG_i has an opposing partner. However, in our succinct representation, we will rely on individual half-edges. We denote directed half-edges as $\overrightarrow{e}$, and undirected edges as e. For a half-edge $\overrightarrow{e} = \overrightarrow{uv}$ with $u, v \in V$, we call v the *head* of $\overrightarrow{e}$, and u the *tail* of $\overrightarrow{e}$.

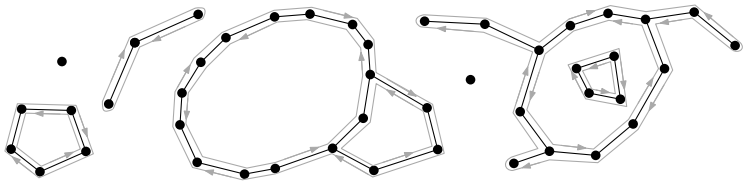


Figure 2 A schematic drawing of RNG_i is shown in black. The face-cycles of RNG_i are shown in gray. All the half-edges of a face-cycle are directed according to the arrows.

3 The Algorithm

In Lemma 3.1 we compute batches of edges of $\mathrm{RNG}(V)$ using $O(s)$ words of workspace. Then using this lemma we enumerate the edges of $\mathrm{RNG}(V)$ by increasing lengths, in Lemma 3.2.

▶ **Lemma 3.1.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. Given a set $Q \subseteq V$ of s sites, we can compute for each $u \in Q$ the at most six neighbors of u in $\mathrm{RNG}(V)$ in total time $O(n \log s)$, using $O(s)$ words of workspace.*

Proof. Let $V_j \subseteq V$, $j = 1, \ldots \lceil n/s \rceil$, be the j-th *batch* of s sites of V. In the first step, we compute $\mathrm{RNG}(Q \cup V_1)$ with standard algorithms in $O(s \log s)$ time using $O(s)$ words of workspace. We store N_1, the set of all neighbors in $\mathrm{RNG}(Q \cup V_1)$ of all sites in Q. Then, in each step $j \neq 1$, we compute $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ in $O(s \log s)$ time using $O(s)$ words of workspace. We store N_j, the set of all neighbors in $\mathrm{RNG}(Q \cup V_j \cup N_{j-1})$ of all sites in Q. Since the degree of sites in Q is at most six, $|N_j| = O(s)$. Notice that for a pair $u \in Q, v \in V$, if v is not among the neighbors of u in $N_{\lceil n/s \rceil}$, at some step there was a site in the lens of u and v. Thus, only the sites in $N_{\lceil n/s \rceil}$ define edges of $\mathrm{RNG}(V)$. However, all of them are not necessarily the neighbors of sites of Q in $\mathrm{RNG}(V)$. To filter these neighbors, we again scan V in batches of size s: for each $u \in Q$, we test if the lens between u and each of its neighbors in $N_{\lceil n/s \rceil}$ is empty of sites of V. After scanning V, the candidates with empty lens define neighbors of u in $\mathrm{RNG}(V)$. Since we use $O(s \log s)$ time per step, the claim follows. ◀

▶ **Lemma 3.2.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. Let $E_R = e_1, \ldots, e_m$ be the sequence of edges in $\mathrm{RNG}(V)$, by increasing length. Let $i \geq 1$. Given e_{i-1} (or null, if $i = 1$), we can find $e_i, \ldots, e_{i+s-1}$ (or $e_i, \ldots, e_m$, if $i + s - 1 > m$), in $O(n^2 \log s/s)$ time using $O(s)$ words of workspace.*

Proof. We generate all the edges of $\mathrm{RNG}(V)$ by applying $O(n/s)$ times Lemma 3.1. Since, we obtain the edges in batches of size $O(s)$, each taking $O(n \log s)$ time, the total time amounts to $O(n^2 \log s/s)$. During this process, we find $e_i, \ldots, e_{i+s-1}$ of E_R with a trick by Chan and Chen [3]. More precisely, whenever we produce new edges of $\mathrm{RNG}(V)$, we store the edges that are longer than e_{i-1} in an array A of size $O(s)$. Whenever A contains more than $2s$ elements, using a linear time selection procedure, we find the edge with rank s, and we remove all edges longer than that [4]. This needs $O(s)$ operations per step, repeating for $O(n/s)$ steps, giving total time $O(n)$ for selecting the edges. In the end, we have $e_i, \ldots, e_{i+s-1}$ in A, albeit not in sorted order. Thus, we sort the final A in $O(s \log s)$ time. The running time is dominated by the time needed to compute the edges of $\mathrm{RNG}(V)$, so the claim follows. ◀

For $i \in \{1, \ldots, m\}$, a *face-cycle* in RNG_i is the circular sequence of half-edges that bounds a face in RNG_i. All half-edges in a face-cycle are oriented in the same direction, and RNG_i can be represented as a collection of face-cycles; see Figure 2. Asano et al. [1] observe that to run Kruskal's algorithm on $\mathrm{RNG}(V)$, it suffices to know the structure of the face-cycles.

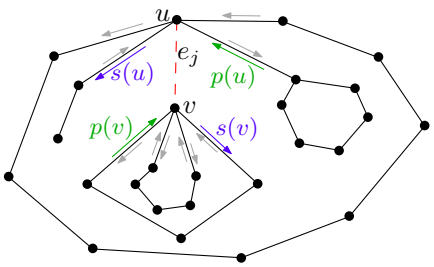


Figure 3 A schematic drawing of RNG_i. The endpoints u and v of e_j identify the predecessors of e_j, shown by $p(u)$ and $p(v)$ in green, and the successors of e_j, shown by $s(u)$ and $s(v)$ in blue.

▶ **Observation 3.3.** *Let $i \in \{1, \ldots, m\}$. The edge $e_i \in E_R$ belongs to $\mathrm{EMST}(V)$ if and only if there is no face-cycle C in RNG_i such that both endpoints of e_i lie on C.*

For $j \geq i \geq 1$, we define *predecessor* (*successor*) of e_j in RNG_i, regarding each endpoint w of e_j, as the half-edge in RNG_i which has w as its head (tail) and is the first edge encountered in a counterclockwise (clockwise) sweep from e_j around w; see Figure 3. If there is no edge incident to w in RNG_i, we set null to the predecessor $p(w)$, and successor $s(w)$, of e_j. Here, we can already derive a simple time-space trade-off for computing $\mathrm{EMST}(V)$.

▶ **Theorem 3.4.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. We can output all the edges of $\mathrm{EMST}(V)$, in sorted order, in $O(n^3 \log s/s)$ time using $O(s)$ words of workspace.*

Proof. Let $E_R = e_1, \ldots, e_m$ be the edges of $\mathrm{RNG}(V)$, sorted by length. We simulate Kruskal's algorithm on E_R: take batches of s edges of E_R and report the ones which are in $\mathrm{EMST}(V)$. More precisely, we use Lemma 3.2 to find a batch of s edges $e_i, \ldots, e_{i+s-1}$, in $O(n^2 \log s/s)$ time. For each such edge e_j, we pick an endpoint $u_j \in V$ and we find first its incident edges in $\mathrm{RNG}(V)$ (Lemma 3.1), and then its incident edges in RNG_j (compare the edges from $\mathrm{RNG}(V)$ with e_j). Then, we identify the successor $s(u_j)$ of each e_j in RNG_j (if it exists), and we perform s parallel walks, where walk j takes place in RNG_j. In each step, we have s current half-edges and we advance each half-edge along its face-cycle, using Lemma 3.1 in $O(n \log s)$ time. A walk j continues until either it encounters the other endpoint of e_j or until it arrives at the predecessor $p(u_j)$ of e_j in RNG_j. Only in the latter case, e_j is in $\mathrm{EMST}(V)$, and we report it. Since there are $O(n)$ half-edges in $\mathrm{RNG}(V)$, it takes $O(n)$ steps to conclude all the walks. Thus, we can process a single batch of edges in $O(n^2 \log s)$ time, using $O(s)$ words of workspace. Since we have $O(n/s)$ many batches, the claim follows. ◀

For the case of linear space $s = n$, the running time of Theorem 3.4 is $O(n^2 \log n)$, while the classic algorithm takes $O(n \log n)$ time to find $\mathrm{EMST}(V)$. The bottleneck in Theorem 3.4 is performing the walks in RNG_j, that might take up to $\Omega(n)$ steps, leading to a running time of $\Omega(n^2 \log s)$ for processing a single batch. To avoid this, we maintain a compressed representation of RNG_j that allows us to reduce the number of steps in each walk to $O(n/s)$.

An *s-net* N for RNG_i, $i \in \{1, \ldots, m\}$, is a collection of half-edges, called *net-edges*, in RNG_i such that: (i) each face-cycle in RNG_i with at least $\lfloor n/s \rfloor + 1$ half-edges contains at least one net-edge; and (ii) for any net-edge $\overrightarrow{e} \in N$, let C be the face-cycle of $\overrightarrow{e}$ in RNG_i. Then, between the head of $\overrightarrow{e}$ and the tail of the next net-edge on C, there are at least $\lfloor n/s \rfloor$ and at most $2\lfloor n/s \rfloor$ other half-edges on C. Note that the next net-edge on C after $\overrightarrow{e}$ could be possibly $\overrightarrow{e}$ itself. This implies that face-cycles with less than $\lfloor n/s \rfloor$ edges contain no net-edges. The following observation records two important properties of s-nets.

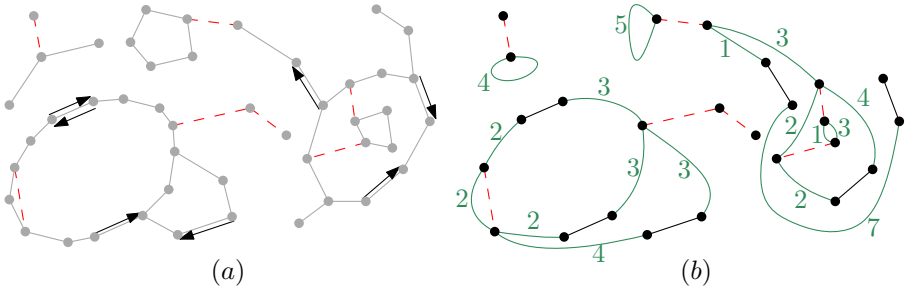


Figure 4 (*a*) A schematic drawing of RNG_i is shown in gray. The half-edges of N are in black and the edges of the next batch $E_{i,s}$ are dashed red segments. (*b*) The auxiliary graph H including the batch-edges (in red). The graph H contains the net-edges (in black), and the successors of the batch-edges and the compressed edges (which are combined in green paths in this picture).

▶ **Observation 3.5.** *Let $i \in \{1, \ldots, m\}$, and N be an s-net for RNG_i. Then, (N1) $|N| = O(s)$; (N2) let $\overrightarrow{f}$ be a half-edge of RNG_i, and C be the face-cycle that contains it. Then, it takes at most $2\lfloor n/s \rfloor$ steps along C from the head of $\overrightarrow{f}$ until either a net-edge or the tail of $\overrightarrow{f}$.*

Proof. Property (ii) implies that only face-cycles of RNG_i with at least $\lfloor n/s \rfloor + 1$ half-edges contain net-edges. Furthermore, on these face-cycles, we can uniquely charge $\Theta(n/s)$ half-edges to each net-edge, again by (ii). Thus, since there are $O(n)$ half-edges in total, we have the first statement. For *(N2)*, note that if C contains less than $2\lfloor n/s \rfloor$ half-edges, the claim holds trivially. Otherwise, C contains at least one net-edge, by property (i). Now, property (ii) shows that we reach a net-edge in at most $2\lfloor n/s \rfloor$ steps from $\overrightarrow{f}$. ◀

Now, we show how to use the s-net in order to speed up the processing of a single batch.

▶ **Lemma 3.6.** *Let $i \in \{1, \ldots, m\}$, and let $E_{i,s} = e_i, \ldots, e_{i+s-1}$ be a batch of s edges of E_R. Suppose we have an s-net N for RNG_i in our workspace. Then, we can determine which edges from $E_{i,s}$ belong to $\mathrm{EMST}(V)$, using $O(n^2 \log s/s)$ time and $O(s)$ words of workspace.*

Proof. Let F be the set of half-edges that contains all net-edges from N, as well as, for each *batch-edge* $e_j \in E_{i,s}$, the two successors of e_j in RNG_i, one for each endpoint of e_j. By definition, we have $|F| = O(s)$, and it takes $O(n \log s)$ time to compute F, using Lemma 3.1. Now, we perform parallel walks through the face-cycles of RNG_i, as in Theorem 3.4. We have one walk for each half-edge in F, and each walk proceeds until it encounters the tail of a half-edge from F (including the starting half-edge itself). In each step of these parallel walks we need $O(n \log s)$ time to find the next edge on the face-cycle and then we need $O(s \log s)$ time to check whether these new edges are in F. Since F contains N, by property *(N2)*, each walk finishes after $O(n/s)$ steps. Thus, the total time for this procedure is $O(n^2 \log s/s)$.

Next, we build an auxiliary *undirected* graph H as follows: the vertices of H are the endpoints of the half-edges in F. Furthermore, H contains undirected edges for all the half-edges in F and additional *compressed edges* representing the outcomes of the walks: if a walk started from the head u of a half-edge in F and ended at the tail v of a half-edge in F, we add an edge from u to v in H, and we label it with the number of steps during the walk. Thus, H contains *F-edges* and *compressed edges*; see Figure 4. After all the walks have been performed, we can construct H in $O(s)$ time, using $O(s)$ words of workspace.

Next, using Kruskal's algorithm we insert the batch-edges of $E_{i,s}$ into H: we determine the connected components of H, in $O(s)$ time using depth-first search. Then, we insert the batch-edges into H, one after another, in sorted order and we keep track of how the

connected components of H change, using a union-find data structure [4]. Whenever a batch-edge connects two different connected components, we output it as an edge of EMST(V). Otherwise, we do nothing. Note that even though H may have a lot more components than RNG_i, the algorithm is still correct, by Observation 3.3. This execution of Kruskal's algorithm, and updating the structure of connected components of H takes $O(s \log s)$ time, which is dominated by the running time of $O(n^2 \log s/s)$ from the first phase of the algorithm. ◀

The following lemma shows how to compute an s-net for RNG_{i+s}, having an s-net for RNG_i and the graph H described in the proof of Lemma 3.6, for each $i \in \{1, \ldots, m\}$.

▶ **Lemma 3.7.** *Let $i \in \{1, \ldots, m\}$, and suppose we have the graph H derived from RNG_i as above, such that all batch-edges have been inserted into H. Then, we can compute an s-net N for RNG_{i+s} in time $O(n^2 \log s/s)$, using $O(s)$ words of workspace.*

Proof. By construction, all *big* face-cycles of RNG_{i+s}, which are the faces with at least $\lfloor n/s \rfloor + 1$ half-edges appear as faces in H. Thus, by walking along all faces in H, and taking into account the labels of the compressed edges, we can determine these big face-cycles in $O(s)$ time. The big face-cycles are represented through sequences of F-edges, compressed edges, and batch-edges. For each such sequence, we determine the positions of the half-edges for the new s-net N, by spreading the half-edges equally at distance $\lfloor n/s \rfloor$ along the sequence, again taking the labels of the compressed edges into account. Since the compressed edges have length $O(n/s)$, for each of them, we create at most $O(1)$ new net-edges. Now that we have determined the positions of the new net-edges on the face-cycles of RNG_{i+s}, we perform $O(s)$ parallel walks in RNG_{i+s} to actually find them. As it was explained in Theorem 3.4, this can be done in $O(n^2 \log s/s)$ time using Lemma 3.1. ◀

The following theorem provides a smooth trade-off between the cubic time constant workspace algorithm and the classical $O(n \log n)$ time algorithm with $O(n)$ words of workspace.

▶ **Theorem 3.8.** *Let V be a set of n sites in the plane, in general position. Let $s \in \{1, \ldots, n\}$ be a parameter. We can output all the edges of* EMST(V)*, in sorted order of length, in $O(n^3 \log s/s^2)$ time using $O(s)$ words of workspace.*

Proof. This follows immediately from Lemma 3.6 and Lemma 3.7, because we need to process $O(n/s)$ batches of edges from E_R. ◀

References

1 T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.

2 M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and applications*. Springer-Verlag, third edition, 2008.

3 T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007.

4 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, third edition, 2009.

5 J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80:1502–1517, 1992.

6 J. S. B. Mitchell and W. Mulzer. Proximity algorithms. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, page to appear. CRC Press, third edition, 2017.

7 G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

Mitered offsets and straight skeletons for circular arc polygons

Bastian Weiß[1], Bert Jüttler[1], and Franz Aurenhammer[2]

1 **Johannes Kepler University Linz**
`bastian.weiss@jku.at`
`bert.juettler@jku.at`
2 **Graz University of Technology**
`franz.aurenhammer@igi.tugraz.at`

Abstract

We generalize the offsetting process that defines straight skeletons of polygons to circular arc polygons. The offsets and the associated skeleton are obtained by applying an evolution process to the boundary and tracing the paths of vertices. These paths define the associated patch decomposition. While the skeleton is a forest, the patches of the decomposition possess a radial monotonicity property. Analyzing the events that occur during the evolution process is non-trivial. This leads us to an event-driven algorithm for offset and skeleton computation. Several examples (both manually created ones and approximations of planar free-form shapes by arc polygons) are presented and used to analyze the performance of our algorithm.

1 Introduction

We define mitered offsets and straight skeletons for planar free-form shapes represented as circular arc polygons, see Fig. 1, which also provides a comparison with classical offsets and polygons.

Offsets and skeletons of planar shapes are widely used in shape analysis, shape design, motion planning, image processing and tool path generation. Besides classical offsets, various generalizations have been considered.

The singularities and self-intersections of offset curves are closely related to the medial axis [6], which is a particular skeleton, i.e., a structural shape descriptor. Algorithms for computing the medial axes of planar shapes are studied in several publications, see [1, 7].

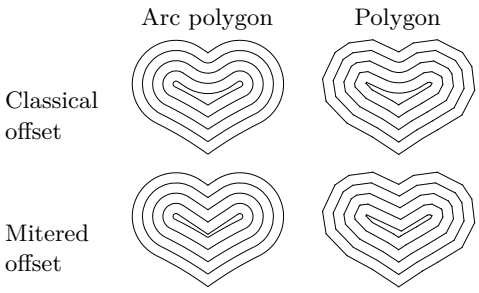


Figure 1 Comparison of classical and mitered offsets.

In the case of piecewise linear shapes, mitered offsets provide an alternative to classical offsets with enhanced shape-preserving properties around reflex vertices [9]. They are defined procedurally, by specifying the evolution of the boundary of a shape as the offset distance increases. Special attention has to be paid to topological changes of the offsets, which can be classified into events.

An evolution is used in [3, 4, 8] to define the straight skeleton of simple polygons and planar straight line graphs.

Circular arc polygons are potentially piecewise G^1 smooth and possess a better approximation order than piecewise linear boundaries [5]. Algorithms for the approximate conversion of general shapes into arc polygons are also well understood. The mitered offsets

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

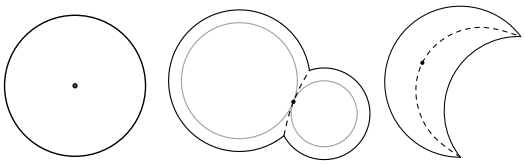
are again defined by specifying the evolution of the boundary, where the use of arcs leads to a wider variety of events.

2 Definitions

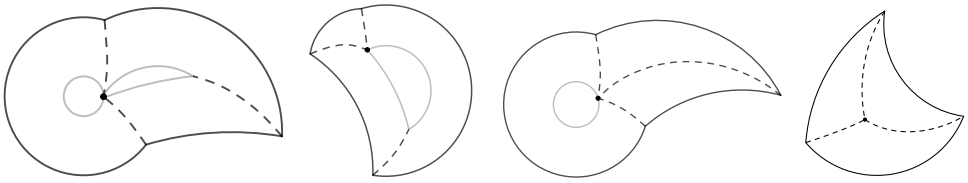
We define offsets, and consequently the skeleton, for arc polygons P as the result of an evolution of its boundary. Offsetting P means that the edges shrink or expand in radial direction (parallel for straight line segments) with constant speed towards the interior of the polygon. Simultaneously, the edges' endpoints travel on certain paths, which are determined by the evolution of the adjacent edges. This process is well-defined until we encounter (self-) intersections or the boundary becomes disconnected, see *splice event* below. We introduce events in order to obtain a globally consistent definition of trimmed offsets. More precisely, we distinguish between four classes of events.

Vanish event An edge e of P vanishes. This happens if the two endpoint vertices of e become coincident and the length of e shrinks to zero, see Fig. 4.

Terminal event These events occur if a connected component of P has three or fewer edges. Depending on the situation, these events can be grouped into seven possible types, see Fig. 2. The label $k\mathrm{Tv}\ell$ of each type depends on the number k of involved edges and ℓ of vanishing edges.



(a) Events 1Tv1, 2Tv0 and 2Tv2 (from left to right).



(b) Events 3Tv0, 3Tv1, 3Tv2 and 3Tv3 (from left to right).

Figure 2 Terminal events.

Contact event The arc polygon touches itself in its interior. There are three events of this type (Fig. 3): A *split event* occurs when a reflex vertex hits an edge of P. Two touching edges create a *squeeze event*. Finally, a *bubble event* happens when the endpoints of a shrinking edge e meet while the length of that edge is not zero.

Circular edges can become disconnected during the offsetting process. Since this situation was not encountered for straight skeletons, we need to introduce a new event:

Splice event A reflex vertex splices at the moment when the adjacent edges become tangential. The continuation of the evolution is not unique. We propose to close the gap in P by inserting a semicircle that starts to expand, thereby creating two smooth vertices, see Fig. 4. This solution ensures the property of local invertibility, which will be discussed in the next section.

Terminal events play a role in the later stages of the evolution. In these situations, it is common that three endpoints meet in a single point. While this might happen also in

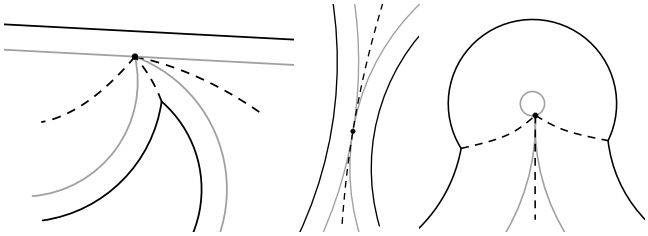


Figure 3 Split, squeeze and bubble event.

earlier stages of the evolution, we do not consider it as it has zero probability, i.e., it occurs in non-generic cases only.

Most events are of local nature, since they involve only a small number of adjacent edges. The only exceptions are split and squeeze events, which entail global modifications of the polygon's topology. Consequently we will distinguish between *local* and *global events*.

3 Properties and computation

The arc polygon P consists of N *vertices* v_i (with indices modulo N) connected by *edges* $e_i = \overline{v_i v_{i+1}}$ in counterclockwise order. Each edge has a center m_i (possibly at infinity) and a radius r_i (non-zero, possibly infinite). Positive and negative values of the radius correspond to arcs with counterclockwise and clockwise orientation, respectively. Vertices with an inner angle of π are called *smooth*. The path of a vertex v_i is a conic section and is determined by its neighbor edges. Note that all edges can be straight line segments, hence we include straight skeletons [3] as a special case.

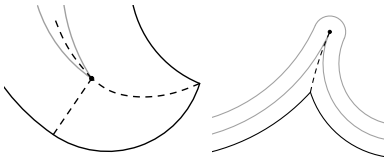


Figure 4 Vanish event (left) and splice event (right). The gray curve is the offset. The dashed lines are part of the skeleton.

Due to the fact that our offsets are defined by an evolution process there is a local invertibility: It is possible to reverse the evolution between two consecutive events. In addition, it is possible to reverse the process when contact or splice events occur. However, this does not hold for vanished edges. The local invertibility is not valid for classic offsets where we would loose convex corners as shown in Fig. 5.

▶ **Definition 3.1.** The skeleton $\mathcal{S}$ of a circular arc polygon P is defined as the path of all non-smooth vertices.

Figure 6 shows an arc polygon P (black) and its skeleton (blue). The extended skeleton is the union of $\mathcal{S}$ with the paths of all smooth vertices (green). This extended skeleton together with P separates the domain into disjoint patches, the so called *patch decomposition* $\mathcal{P}$. In Fig. 6 there is a hyperbolic part of $\mathcal{S}$ emerging from the reflex vertex. The upper part of $\mathcal{S}$ from left to right, consists of a hyperbolic arc, an elliptic arc and another hyperbolic arc, joined smoothly. The skeleton $\mathcal{S}$ is a *forest* and has, in general, several connected components. It can be shown that $\mathcal{S}$ has *linear complexity*. While this is rather obvious for straight skeletons of polygons [3], the analysis is slightly more involved for circular arc polygons, due to the presence of splice events.

Figure 5 Local invertibility.

The straight skeleton possesses a monotonicity perpendicular to the defining edge, while the patch decomposition $\mathcal{P}$ has *radial monotonicity*. That means, every radial line intersected with its patch is connected (Fig. 6, red segments).

The skeleton $\mathcal{S}$ is a generalization of straight skeletons. It is not obvious how to apply well known principles like sweep-line and divide & conquer since straight skeletons do not have a Voronoi structure [4]. The intuitive reason for this is that the velocity of a vertex is determined by its inner angle. In fact, the vertex velocity tends to infinity as the angle approaches zero. Instead, our approach is based on a simulation of the event driven evolution that defines mitered offsets.

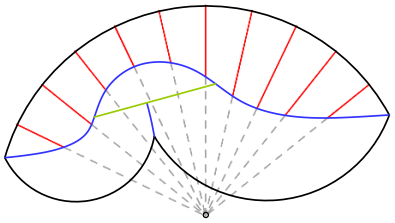


Figure 6 shows the skeleton $\mathcal{S}$ (blue) and the paths of smooth vertices in green. The radial red lines show the radial monotonicity property.

Our main data structures consists of a list E containing edges and vertices which represents P and an *event queue* Q. Events are interlinked to edges and vertices and stored in Q, ordered consecutively. The arc polygon P and its offsets are represented by patches on right circular cones in space-time, where the time has been added as third (vertical) coordinate. The patch associated with the edge e_i is denoted by c_i. A patch is represented by its boundary curves. Splice events create new edges and associated patches. Vanish and terminal events trigger the deletion of edges from the arc polygon. At this stage, the associated patch is complete.

Our algorithm takes P given as an edge list E and computes the skeleton and patch structure. Initially, the event queue Q is populated by all local events that can be computed from P. Now we loop over E as long as it is not empty. Each loop performs three steps: First, computation of the next possible global event and comparison with the next local event give us the next event e. Second, grow $\mathcal{S}$ and $\mathcal{P}$ until we reach the event. Third, the event is handled (see details below). This may trigger the insertion of new local events into Q.

The computation of events is done differently for local and global ones. All local events are computed by intersecting cones in space-time, which represent the edges, and certain planes. For instance, a splice event of v_i is computed by analyzing the intersection $c_{i-1} \cap c_i \cap p_i$, where p_i is the vertical plane defined by the points m_{i-1}, m_i.

While computing local events of edges or vertices requires knowledge about their local neighborhood only, the computation of global events is more expensive since the entire arc polygon P has to be considered. To speed up this computation, we use a sweep-line algorithm over axis–aligned bounding boxes of edges and vertices of P.

The procedure for handling an event operates on E which represents P, the event queue Q, the event e itself and the data structures holding the skeleton $\mathcal{S}$ and patches $\mathcal{P}$.

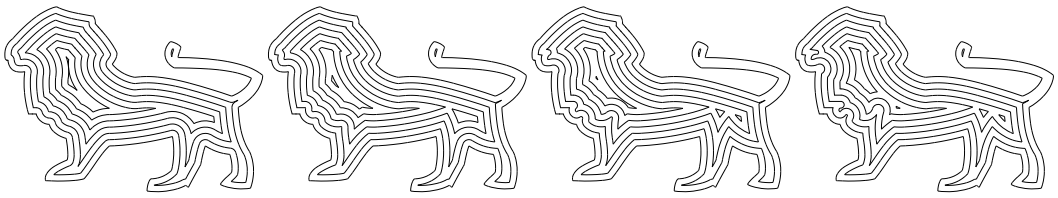
1. Add/delete/split edges.
2. Reconnect edges properly such that there is no gap in P.
3. Update S and $\mathcal{P}$.
4. Delete e from Q.
5. $Q \leftarrow Q \cup \{\text{local events for edges having a new neighbour}\}$

The splice event is necessary to close the gap of the polygon boundary if it breaks. We suggested in Section 2 to do this at the latest possible point. However, an earlier splice is achievable: The introduction of an angle σ, which is the inner angle of a reflex vertex v,

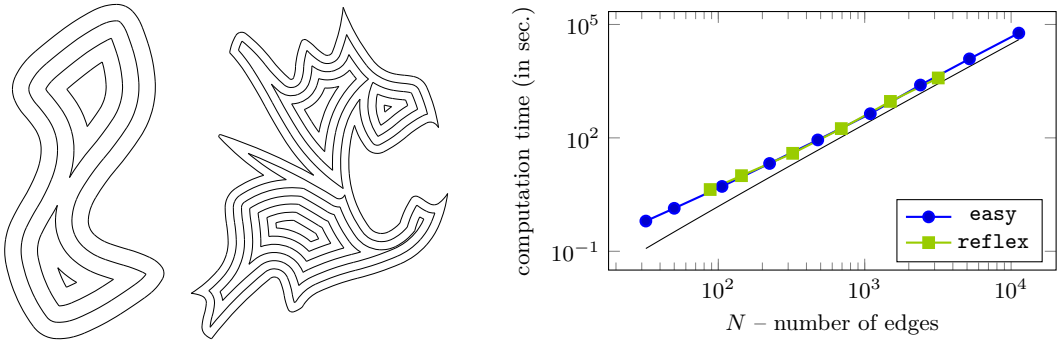
allows us to control the splice behavior of P. Choosing $\sigma = 2\pi$ results in the known splice event. An angle of $\pi < \sigma < 2\pi$ forces v to splice earlier and $\sigma = \pi$ cause each reflex vertex to splice immediately which results in classical offsets and the obtained skeleton is the medial axis. The analysis of the algorithms' complexity is still ongoing work. Experimental results are provided in the next section.

4 Examples

We perform experiments with manually designed arc polygons as well as approximations of planar free form shapes by circular arc polygons. These arc splines are created by spiral biarcs following the approach described in [2]. The results are depicted in Fig. 7.



(a) The `lion` shape for $\sigma = 180°$, $280°$, $320°$ and $360°$.



(b) The examples `easy`, `reflex` and computation times.

Figure 7 Offsets of examples `easy` and `reflex` (left and center). Computation times (right) for arc splines of different size representing these examples. As a reference, the black line indicates $\mathcal{O}(N^2 \log N)$.

Figure 7a shows offsets of the lion shape, which is represented by spiral biarcs, for various values of the splicing parameter σ. The leftmost picture visualizes the classical offset. Increasing σ delays the occurring splice events, thereby preserving the reflex vertices.

The examples `easy` and `reflex` (Fig. 7b) are used to analyze the complexity of the algorithm. Finally, we also include Fig. 8, containing straight line segments (arcs with infinite radii) on the left. The right of Fig. 8 shows the skeleton (blue) and the paths of smooth vertices (green).

The proposed algorithm has been implemented in Python 3.6 on an Intel i7-6700 CPU machine with 8 GB RAM. We demonstrate experimentally that our algorithms' complexity does not exceed $\mathcal{O}(N^2 \log N)$ in practice: Figure 7b (right) shows the computation time in seconds over the number of edges N for the `easy` and `reflex` examples. Both axes use logarithmic scales. Since the graphs depicting the computation times for the two examples seem to become tangential to the reference line (black) Fig. 7b supports the claimed complexity.

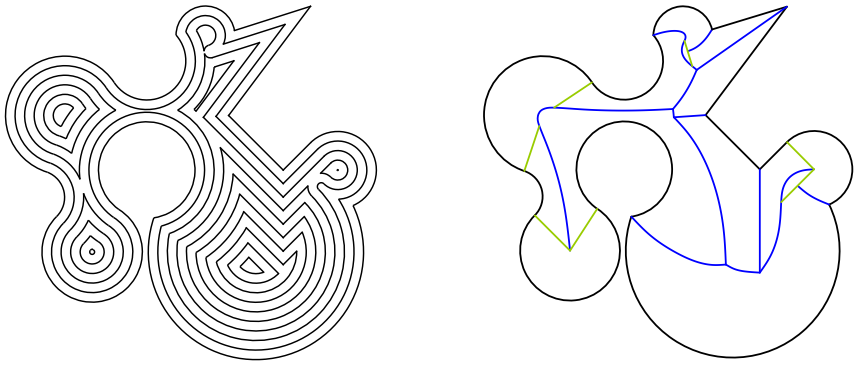


Figure 8 Mitered offsets (left) and skeleton (right, blue).

5 Conclusion

We presented an extension of straight skeletons to shapes bounded by circular arc polygons. Experimental results indicate that the proposed algorithm computes offsets and its skeleton for N edges in $\mathcal{O}(N^2 \log N)$ time in practice. The size of the resulting skeleton is linear, the patch decomposition possesses a radial monotonicity property and the offsets are locally invertible. The introduction of the parameter σ allows us to control the inner angle at which reflex vertices splice, thereby influencing the structure of the skeleton. Consequently, our algorithm can be adapted to compute either the medial axis or the straight skeleton (for polygons with only straight edges) as special cases. Finally we note that the algorithm can be extended to general circular arc figures in the plane.

References

1 O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, E. Pilgerstorfer, and M. Rabl. Divide-and-conquer for voronoi diagrams revisited. *Computational Geometry: Theory and Applications*, 8(43):688–699, 2010.

2 O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, and M. Rabl. Medial axis computation for planar free–form shapes. *Computer-Aided Design*, 41(5):339–349, 2009.

3 O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *International Computing and Combinatorics Conference*, pages 117–126. Springer, 1996.

4 O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. In *The Journal of Universal Computer Science*, pages 752–761. Springer, 1996.

5 O. Aichholzer, F. Aurenhammer, T. Hackl, B. Jüttler, M. Rabl, and Z. Šír. Computational and structural advantages of circular boundary representation. *Int. J. Comput. Geom. Appl.*, 21(1):47–69, 2011.

6 H. Blum. A transformation for extracting new descriptors of shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

7 D.T. Lee. Medial axis transformation of a planar shape. *IEEE Transactions on pattern analysis and machine intelligence*, pages 363–369, 1982.

8 Peter Palfrader and Martin Held. Computing mitered offset curves based on straight skeletons. *Computer-Aided Design and Applications*, 12(4):414–424, 2015.

9 S.C. Park and Y.C. Chung. Mitered offset for profile machining. *Computer-Aided Design*, 35(5):501–505, 2003.

The Partition Spanning Forest Problem*

Philipp Kindermann[1], Boris Klemz[2], Ignaz Rutter[3], Patrick Schnider[4], and André Schulz[5]

1 **FernUniversität in Hagen, Germany**
philipp.kindermann@fernuni-hagen.de
2 **Freie Universität Berlin, Germany**
klemz@inf.fu-berlin.de
3 **TU Eindhoven, The Netherlands**
I.Rutter@tue.nl
4 **ETH Zürich, Switzerland**
patrick.schnider@inf.ethz.ch
5 **FernUniversität in Hagen, Germany**
andre.schulz@fernuni-hagen.de

Abstract

Given a set of colored points in the plane, we ask if there exists a crossing-free straight-line drawing of a spanning forest, such that every tree in the forest contains exactly the points of one color class. We show that the problem is NP-complete, even if every color class contains at most five points, but it is solvable in $O(n^2)$ time when each color class contains at most three points. If we require that the spanning forest is a linear forest, then the problem becomes NP-complete even if every color class contains at most four points.

1 Introduction

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane and let $C = \{C_1, \ldots, C_k\}$ be a partition of P into k sets of points, called *color classes*, such that every point belongs to exactly one color class. We study the *partition spanning forest problem* which is defined as follows: Is there a crossing-free straight-line drawing of a spanning forest F that consists of k trees $T_1, \ldots, T_k$ such that each tree T_i, $1 \leq i \leq k$, contains exactly the points of the color class C_i? Figure 1 shows an example with three color classes.

For $k = 1$, the problem is equivalent to finding a geometric spanning tree of P which trivially always exists. Hence, several optimization versions of this problem have been studied in the past; see Eppstein [4] for a survey. Bereg et al. [3] showed how to solve the problem in $O(n \log n)$ time in the case of $k = 2$. Hiu and Schaefer [5] proved that it is NP-complete to decide for two color classes $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ whether there exists an ordering π such that the geometric paths $a_{\pi_1}, \ldots, a_{\pi_n}$ and $b_{\pi_1}, \ldots, b_{\pi_n}$ are crossing-free. Bereg et al. [2] asked for not necessarily straight-line Steiner trees for each color class of minimum total length and gave a PTAS for $k = 2$ and a $(k + \varepsilon)$-approximation for $k > 2$.

In this paper, we analyze the complexity of the partition spanning forest problem for color classes of bounded size. We give an $O(n^2)$-time algorithm when each color class contains at most three points (Sec. 2) and show that the problem is NP-complete for up to five points per color class (Sec. 3); the complexity for four points remains open. In Section 4, we show that the *partition spanning linear forest problem*, where each tree is required to be a path, is NP-complete, even if every color class contains at most four points.

* This work started at the 14th European Research Week on Geometric Graph (GGWeek'17) in Vierhouten, The Netherlands. This research was funded in part by Humility & Conviction in Public Life, a project of the University Connecticut sponsored by the John Templeton Foundation.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

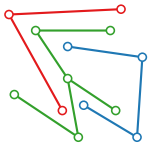


Figure 1 A solution to a problem instance with three color classes.

2 Color classes with at most three points

In the case where each color class of the input instance contains of at most three points, the partition spanning forest problem can be solved in polynomial time. In fact, with this restriction the problem can be formulated as a 2-SAT problem.

Assume that our point set $P = \{p_1, \ldots, p_n\}$ consists of n points. In the following we will understand the color classes as subsets $I \subseteq [n] := \{1, \ldots, n\}$ of indices. For a point p_i we denote its color class by $I(p_i)$. We refer to the edges (p_i, p_j) where p_i and p_j are in the same color class as the *potential edges* of the instance. Observe that an arbitrary choice of the potential edges forms a solution to the problem (with at most three points per color class) if and only if it satisfies the following conditions: (i) For each point p_i, if $|I(p_i)| > 1$, then at least one potential edge incident to p_i must be chosen. (ii) For any pair of potential edges p_ip_j and p_kp_l that intersect in the interior, at most one of them is chosen. (iii) For any color class I with $|I| = 3$ one of the potential edges of that color is not chosen.

Observe that condition (iii) can be skipped, as any choice of potential edges satisfying conditions (i) and (ii) can be extended to also satisfy (iii).

We model the possible choices of potential edges that satisfy conditions (i) and (ii) by a 2-SAT formula as follows. For each potential edge (p_i, p_j) there is a variable x_{ij} with the interpretation that if x_{ij} is true, then the edge connecting p_i to p_j is *not* chosen as part of the solution, and otherwise it is.

Conditions (i) and (ii) can be expressed as 2-SAT formulas using the variables x_{ij} as follows. For condition (i), we create for each point p_i the (sub)formula $\bigvee_{j \in I(p_i) \setminus \{i\}} \neg x_{ij}$. Note that this is a 2-SAT formula since $|I(p_i) \setminus \{i\}| \leq 2$ by the assumption that each color class has size at most three. For any two potential edges (p_i, p_j) and (p_k, p_l) that cross, we add the clause $x_{ij} \vee x_{kl}$, thus enforcing condition (ii). It follows that the resulting 2-SAT formula φ is satisfiable if and only if the original instance of the partition spanning forest problem admits a solution. The formula has length at most $O(n^2)$ and can be constructed in $O(n^2)$ time as well. By using an efficient algorithm for 2-SAT [1], we get the desired algorithm. We summarize our construction in the following theorem.

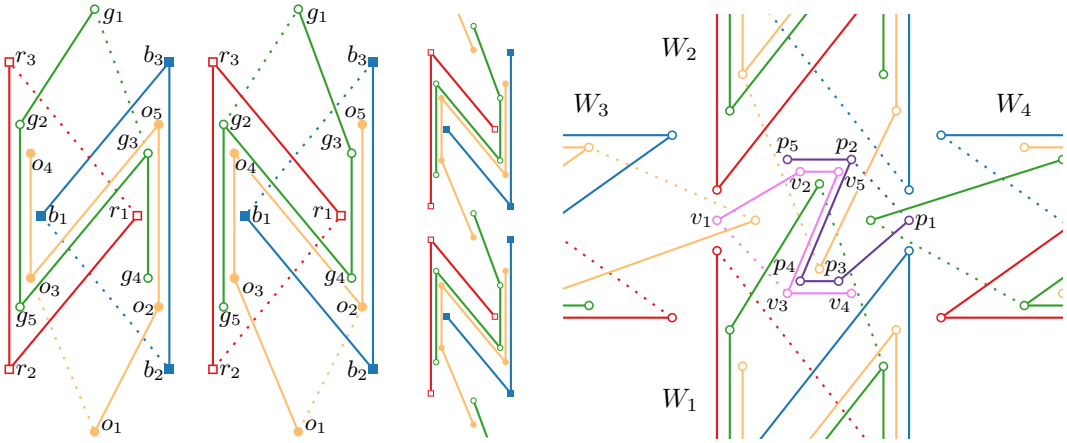
▶ **Theorem 1.** *The partition spanning forest problem for n points can be solved in $O(n^2)$ time if every color class contains at most three points.*

3 Color classes with at most five points

In this section we prove the following theorem:

▶ **Theorem 2.** *The partition spanning forest problem is* NP*-complete, even if every color class contains at most five points.*

The problem is obviously contained in NP. In order to show the NP-hardness, we perform a polynomial-time reduction from Planar 3-Satisfiability. In this NP-hard [7] special case of 3SAT the input is a 3SAT formula φ whose variable–clause graph is planar. We can



(a) State (i) **(b)** State (ii) **(c)** A chain **(d)** Splitting gadget

Figure 2 (a–c) The configurations of the wire gadget and (d) the splitting gadget.

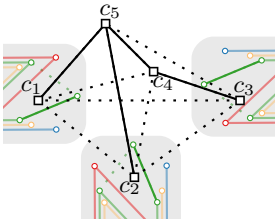
assume that such a formula is given together with a contact representation $\mathcal{R}$ of φ [6]. Thus, all variables are represented as horizontal line segments arranged on one line. Each clause c is represented as an E-shape turned by 90° such that the three vertical *legs* of the E-shape touch precisely the variables contained in c. For our reduction, we construct a set of colored points that admits a partition drawing if and only if φ is satisfiable.

Overview. We introduce five types of gadgets. For each variable u we create a *variable gadget* which admits exactly two distinct partition drawings. These drawings correspond to the two truth states of u. *Wire gadgets* are used to propagate these states to the *clause gadgets*, one of which is created for every clause c. The clause gadget of c ensures that gadget configurations of the variables contained in c correspond to a truth assignment in which at least one of the literals of c is satisfied. In order to connect our gadgets appropriately we also require a *splitting gadget*, which splits one wire into two wires, and we require a gadget that flips the state transported along a wire. We proceed by describing our gadgets in detail. Note that different gadgets always use different color classes, even if we might give them the same name in the construction (so there are many *red* color classes in an instance).

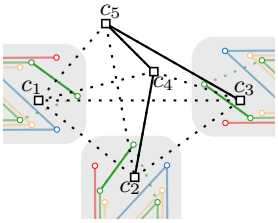
The wire gadget. The wire gadget consists of four color classes; see Figure 2. The points of the *red* color class $R = \{r_1, r_2, r_3\}$ and the *blue* color class $B = \{b_1, b_2, b_3\}$ are arranged such that the convex hulls of R and B intersect in the two points $b_1b_2 \cap r_1r_2$ and $b_1b_3 \cap r_1r_3$. As a consequence, there are exactly two possible configurations for the red and blue spannings trees which can be used in a partition drawing, see Figure 2a and Figure 2b. Either choice uniquely determines the spanning tree of both the *green* color class $G = (g_1, \dots, g_5)$ and the *orange* color class $O = (o_1, \dots, o_5)$, as the edges of the red and blue spanning trees obstruct all other possible green and orange edges. Thus, there are exactly two possible partition drawings of the wire gadget. In particular, these two drawings satisfy the following.

▶ **Observation 3.** *Any partition drawing of the wire gadget either contains (i) the edges g_1g_2 and o_1o_2, but not the edges g_1g_3 and o_1o_3, see Figure 2a; or (ii) the edges g_1g_3 and o_1o_3, but not the edges g_1g_2 and o_1o_2, see Figure 2b.*

These two states (i) and (ii) may be propagated by creating *chains* of wire gadgets in which the convex hulls of consecutive gadgets intersect in two points as illustrated in Figure 2c. Consider two consecutive wire gadgets in a chain. By Observation 3, either both gadgets



(a) $\ell_1 = \ell_2 = \ell_3 =$ TRUE



(b) $\ell_1 = \ell_2 = \ell_3 =$ FALSE

Figure 3 The clause gadget between literals ℓ_1, ℓ_2, ℓ_3.

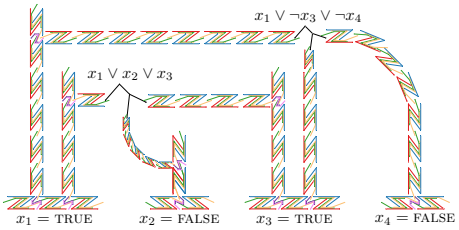


Figure 4 A full example.

are in state (i) or both gadgets are in state (ii) due to the way their convex hulls intersect. As a consequence, the first gadget of the chain is in state (i) if and only if the last one is in state (i) as well. Chains are flexible structures and turns can easily be implemented by curving a chain. Further, the length of a chain may be adjusted by increasing or decreasing the distance between consecutive wire gadgets.

Splitting and inverting. The splitting gadget consists of two color classes $V = \{v_1, \ldots, v_5\}$ (*violet*) and $P = \{p_1, \ldots, p_5\}$ (*purple*) whose points are placed between two consecutive wires W_1, W_2 in a chain, see Figure 2d. The functionality of these two color classes is similar to the one of the color classes green and orange in the wire gadget: the state of W_1 and W_2 uniquely determines the spanning tree of both the violet and the purple color class. We may now attach one or two additional wires perpendicular to the chain such that their convex hulls intersect the convex hull of the splitting gadget, see W_3 and W_4 in Figure 2d. The edges incident to p_1 and v_1 in the purple and violet spanning trees allow precisely one state for both W_3 and W_4.

▶ **Observation 4.** *In any drawing of the splitting gadget, the state of the wires W_3 and W_4 differs from the state of W_1 and W_2.*

In this sense, the splitting gadget does not only split a wire into two wires, it can also be used to flip the state propagated along a chain.

The variable gadget. The variable gadget is a horizontal chain to which we attach multiple wires using splitters. The number of wires attached from the top (bottom) matches the number of E-shape legs touching the variable from the top (bottom) in the contact representation $\mathcal{R}$ of φ.

The clause gadget. The clause gadget for a clause of three literals ℓ_1, ℓ_2, ℓ_3 consists of one color class with exactly five vertices $c_1, \ldots, c_5$. We place c_1, c_2, and c_3 inside a wire gadget representing ℓ_1, ℓ_2, and ℓ_3, respectively, and we place c_4 and c_5 between those as depicted in Figure 3. We will now show that the gadget is drawable if and only if at least one of ℓ_1, ℓ_2, ℓ_3 is TRUE. In particular, we can always use an edge to connect c_4 and c_5. We can connect c_3 to c_4 if ℓ_3 is TRUE and we can connect c_3 to c_5 otherwise; similarly, we can connect c_2 to c_5 if $\ell_2 =$ TRUE and we can connect c_2 to c_4 otherwise. If $\ell_1 =$ TRUE, then we can always connect c_1 to c_5. However, if $\ell_1 =$ FALSE, then we cannot connect c_1 to c_4 or c_5, and we can connect it to c_2 or c_3 only if ℓ_2 or ℓ_3 is TRUE, respectively. Hence, the gadget is not drawable if $\ell_1 = \ell_2 = \ell_3 =$FALSE. Note that the connection from c_1 to c_3 might intersect the connection from c_2 to c_4. However, we only have to use it if $\ell_1 = \ell_2 =$ FALSE and $\ell_3 =$ TRUE; in this case, we can connect c_2 to c_3 instead of c_4. Thus, the gadget is drawable if and only if at least one of ℓ_1, ℓ_2, and ℓ_3 is TRUE.

Layout and correctness. The wires that are attached to the variable gadgets are vertical and, by Observation 4, their state is inverted, so they propagate the negated variable. Hence,

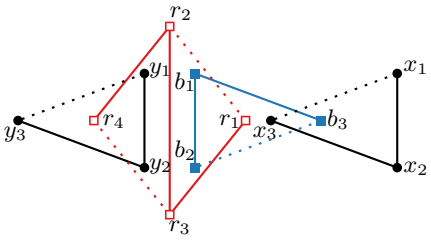


Figure 5 The wire gadget.

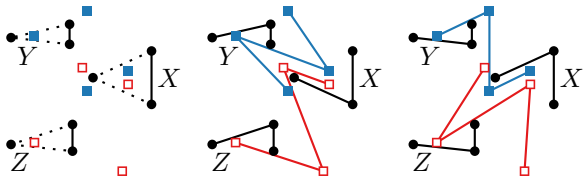


Figure 6 The splitting gadget and its assignments.

if a literal is positive, we have to invert the state of the wire again. Two of the wires are supposed to enter the clause horizontally; for these two, if they correspond to a positive literal, we simply use another splitting gadget to make the wire horizontal. Otherwise, the wire makes a 90° degree turn to become horizontal and to propagate the negated variable. The third wire is supposed to enter the clause gadget vertically, so if its literal is negative, the vertical wire can directly connect to the clause. Otherwise, we use another splitting gadget followed by a 90° degree turn. See Figure 4 for an example of that shows all cases. Since the clause gadgets are drawable if and only if one of their literals is TRUE and since the wires propagate the states of the variable gadgets, the resulting instance is drawable if and only if the planar 3SAT formula φ is satisfiable, which proves the correctness of Theorem 2.

4 Linear forests for color classes with at most four points

In this section we consider the additional restriction that the spanning forest is a linear forest, that is, each connected component is a path. Note that, if every color class contains at most three points, then every spanning forest is linear, so in this case we can solve the problem in polynomial time. On the other hand, we show that under this additional restriction, the problem is NP-complete already if every color class contains at most four points.

▶ **Theorem 5.** *The partition spanning linear forest problem is* NP*-complete, even if every color class contains at most four points.*

Again, the problem is clearly contained in NP. In order to show the NP-hardness, we again perform a polynomial-time reduction from Planar 3-Satisfiability, but using different gadgets. As before, we construct a variable gadget, a splitting gadget, a wire gadget, and an inverter gadget. Instead of directly constructing a clause gadget, we will however construct an OR-gadget. The clause gadget can then be built by concatenating two OR-gadgets and enforcing the resulting variable gadget to be set to TRUE by crossing the appropriate edge with a new color class consisting of two points.

The variable, wire, and inverter gadgets. The variable gadget consists of one color class, the *black* color class $X = \{x_1, x_2, x_3\}$. Using a second color class, the *blue* color class $B = \{b_1, b_2, b_3\}$, we can enforce that the edge x_1x_2 must be drawn in any partition drawing. The classes B and X are placed in such a way that their convex hulls intersect in two points. In particular, there are two distinct partition drawings for B and X, corresponding to two truth states and x_1x_2 is present in both of them.

The wire gadget consists of four color classes, the *red* color class $R = \{r_1, r_2, r_3, r_4\}$ and the *blue* color class $B = \{b_1, b_2, b_3\}$, and two *black* color classes $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$, see Figure 5. Classes B and X are placed as in the variable gadget. Class Y is a copy of X, placed outside the convex hull of X and B. The point r_1 is placed inside the convex hull of B but outside the convex hull of X. The point r_4 is placed inside the convex

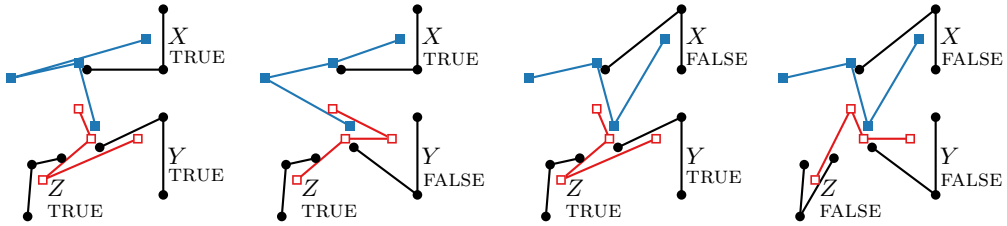


Figure 7 Assignments of the OR-gadget.

hull of Y and r_2 and r_3 are placed such that the line through them separates the convex hulls of B and Y. Then, either partition drawing on X and B induces a unique partition drawing of R and Y, where the drawing on Y is the same as the drawing on X.

Placing Y as a copy of B instead of X, i.e., with only one point in the convex hull of R, we can also turn this gadget into an inverter gadget.

The splitting gadget. The splitting gadget consists of three variable gadgets X, Y, and Z, and two additional color classes, the *red* color class R and the *blue* color class B, see Figure 6. The truth assignment on X enforces some edges in R and B to be present, which then uniquely determines the partition drawing on the whole gadget. Note that the truth assignments on Y and Z are enforced as the negated truth assignment on X, so an additional inverter gadget might be needed depending on the required literal.

The OR-gadget. The OR-gadget consists of three variable gadgets X, Y, and Z, and two additional color classes, the *red* color class R and the *blue* color class B, see Figure 7. The truth assignments on X and Y enforce some edges in R and B to be present. It can be seen that the drawing of Z corresponding to the value TRUE can only be drawn if X or Y are also drawn corresponding to the value TRUE. In some of these cases, Z could also be drawn according to the value FALSE, but this does not affect the proof as it is still true that the constructed point set admits a partition drawing if and only if the planar 3SAT formula φ is satisfiable.

References

1 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979.

2 Sergey Bereg, Krzysztof Fleszar, Philipp Kindermann, Sergey Pupyrev, Joachim Spoerhase, and Alexander Wolff. Colored non-crossing euclidean steiner forest. In Khaled M. Elbassioni and Kazuhisa Makino, editors, *Proc. 26th Int. Symp. Alg. Comput. (ISAAC'15)*, volume 9472 of *Lecture Notes Comput. Sci.*, pages 429–441. Springer, 2015.

3 Sergey Bereg, Minghui Jiang, Boting Yang, and Binhai Zhu. On the red/blue spanning tree problem. *Theor. Comput. Sci.*, 412(23):2459–2467, 2011.

4 David Eppstein. Spanning trees and spanners. *Handbook Comput. Geom.*, pages 425–461, 1999.

5 Peter Hui and Marcus Schaefer. Paired pointset traversal. In Rudolf Fleischer and Gerhard Trippen, editors, *Proc. 15th Int. Symp. Alg. Comput. (ISAAC'04)*, volume 3341 of *Lecture Notes Comput. Sci.*, pages 534–544. Springer, 2004.

6 Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. Disc. Math.*, 5(3):422–427, 1992.

7 David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

Protecting a highway from fire*†

Rolf Klein[1], David Kübel[1], Elmar Langetepe[1], and Barbara Schwarzwald[1]

1 Department of Computer Science, University of Bonn

Abstract

Suppose a fire spreads at speed 1 along a highway given by a horizontal line in the L_1 plane. A fighter is tasked to protect the highway by building barriers along or perpendicular to the highway with a building speed v. The fighter can move without delay or additional costs between different construction sites. We show that $v > 1.5$ is necessary and $v > \frac{2+\sqrt{5}}{\sqrt{5}} = 1.8944\ldots$ is sufficient.

1 Introduction and problem statement

In fire fighting theory and practice, different tasks are important: quenching or enclosing a fire, and protecting objects from fire; see [6]. Results in continuous and discrete models differ significantly; see [1–5, 7].

In our scenario, a horizontal highway of infinite length in the L_1 plane must be protected from a fire originating from $(0, 0)$ at speed 1 in all directions. The highway is considered as protected if no point on the highway is ever touched by the fire. For protection, a fire fighter may build barriers at speed v while the fire is spreading. Barriers are impassable for the fire, but can only be built at locations where the fire has not arrived, yet.

One way to protect the highway would be to enclose the fire by barriers in some fashion. This model has been investigated e. g. by Bressan [2]. He considers a fighter that can build arbitrary curves and is able to fly between construction sites without delay and free of costs. For the L_2 plane, he showed that a building speed of $v > 1$ is necessary and $v > 2$ is sufficient. Despite serious effort, this gap is still open.

We discuss a different model and obtain better bounds, where $v > 1.5$ is necessary and $v > 1.8944\ldots$ suffices. In our model the flying fighter is tasked to protect the highway by building barriers along, or perpendicular to, the highway in the L_1 plane. Note that in this model the highway is protected if we can guarantee that the barriers along the highway can always be built before the fire arrives there. Thus, the purpose of the perpendicular barriers is only to slow down the fire. Namely, the fire has to move up and down to overcome the vertical barrier, while the fighter can set it up in one go; see e. g. barrier d_2 in Figure 1. This involves that d_2 must be ready where the fire crawls upwards. However, it need not be built a second time, when the fire crawls downwards along the backside. The question is, whether these savings allow to improve on the naive approach, which only builds horizontal barriers along the highway and obviously requires $v \geq 2$.

2 Model

In our model, the highway is the x-axis and the fire originates from the point $(0, 0)$ and continuously expands over time equally in all directions with speed 1 according to the L_1 metric. Since we allow the fire to start at the highway, we allow an arbitrarily small head-start of barrier of length s into both directions along the highway.

* This work has been supported by DFG grant Kl 655/19 as part of a DACH project.
† We thank all anonymous reviewers of EuroCG for their valuable comments.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

A *barrier set* is described by two series of pairs (a_i, b_i) and (c_i, d_i), one for each direction. While a_i and c_i denote the distance to the previous barrier in the series (or to $(0, 0)$ for $i = 1$), b_i and d_i denote the length of the vertical barrier; see Figure 1 for an example of a general barrier set. Note that this barrier set also includes horizontal barriers along the highway.

A necessary condition for a barrier set solution with speed v can be obtained as follows. Assume that the final system of barriers has been built and then the fire spreads. At any moment of time t, the total amount of barrier *consumed* by the fire must not exceed $v \cdot t$. Here we consider a piece of the barriers as consumed, as soon as the fire has reached this piece for the first time; see Figure 1.

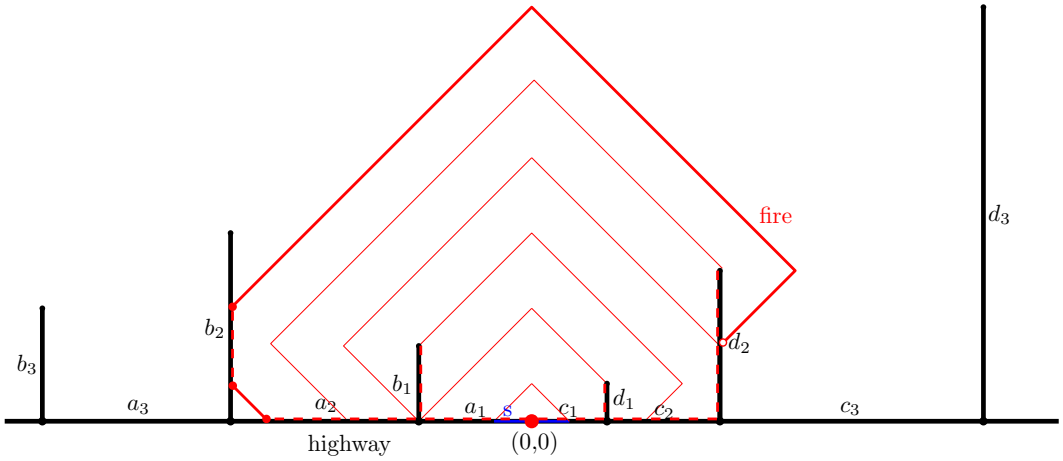


Figure 1 A snapshot of the fire burning along a barrier set. Dashed lines mark the consumed pieces of the barriers. The fire burns along barriers at 4 places, namely twice at b_2 and once at a_2 and d_2. However, the current consumption is only 3, since the fire burning along the back of d_2, which had already been reached, does not count.

Thus, we define the *total consumption* and *consumption-ratio* for a barrier set $\mathcal{S}$:

$$\begin{aligned} \mathcal{C}_{\mathcal{S}}(t) &:= \text{length of all pieces of barriers} \in \mathcal{S} \text{ consumed by the fire until time t} \\ \mathcal{Q}_{\mathcal{S}}(t) &:= \tfrac{\mathcal{C}_{\mathcal{S}}(t)}{t}\,. \end{aligned}$$

As the fire spreads over the barrier set, it represents a geodesic L_1 circle. At some points in time, the structure of its boundary changes. Between such events, the derivative of $\mathcal{C}_{\mathcal{S}}(t)$ is an integer given by the number of places, where the fire burns along a part of the barrier set for the first time. We call this value k the *current consumption* and the time interval between the events a *k-interval*. In Figure 1, there are four boundary points, but only three of them count, thus the current consumption is 3.

Note that all these definitions can easily be restricted to either side of the barrier set, e. g. denoted by $\mathcal{Q}^l_{\mathcal{S}}(t)$ and $\mathcal{Q}^r_{\mathcal{S}}(t)$. Obviously, $\mathcal{Q}_{\mathcal{S}}(t) = \mathcal{Q}^l_{\mathcal{S}}(t) + \mathcal{Q}^r_{\mathcal{S}}(t)$.

A barrier set is then called *viable* for any speed v such that $v > \max_t \mathcal{Q}_S(t)$. Viability is necessary and sufficient, too, as Lemma 2.1 shows. Due to space limitations, we omit the proof of this technical lemma.

▶ **Lemma 2.1.** *For any barrier set that is viable for a speed v, there exists a building plan for this barrier set with speed $v + \delta$, where $\delta > 0$ can be arbitrarily small.*

The extra speed δ is required if we assume that the fighter cannot build single points and has to build pieces of positive length one at a time instead.

The question then obviously is: What is the minimum speed v for which a viable barrier set exists?

3 A lower bound of $v > 1.5$

First, we show that not all possible barrier sets have to be considered.

▶ **Lemma 3.1** (Increasing barrier lengths). *For any barrier set $\mathcal{S}$ viable for a speed v, there exists a barrier set consisting of vertical barriers of strictly increasing length in each direction which is also viable for v. That is, $b_{i+1} > b_i$ and $d_{i+1} > d_i$, $\forall i \geq 1$.*

Proof. Assume $\mathcal{S}$ contains a vertical barrier w_2 that is at most as long as the previous one w_1 in that direction. Then consider the barrier set $\mathcal{S}' = \mathcal{S} \setminus w_2$. Removing w_2 does not decrease the shortest L_1 path from the fire origin $(0, 0)$ to any point in the plane including points on all other barriers, as those paths are prolonged by w_1. However, w_2 is no longer consumed and hence the total consumption $\mathcal{C}_{\mathcal{S}'}(t) \leq \mathcal{C}_{\mathcal{S}}(t)$ for all t and therefore $\mathcal{Q}_{\mathcal{S}'}(t) \leq \mathcal{Q}_{\mathcal{S}}(t)$. ◀

▶ **Theorem 3.2** (Lower Bound). *There exists no viable barrier set for any speed $v \leq 1.5$.*

Proof. The idea of the proof is to show that the ratios $\mathcal{Q}^{l,r}_{\mathcal{S}}(t)$ are always larger than 0.5, for t greater than some threshold and sometimes exceed 1 for arbitrarily large t. We start with the latter.

Assume there exists a barrier set $\mathcal{S}^*$ which is viable for $v < 1.5$. For every time $t_0 > 0$ there must exist some point in time $t_1 > t_0$, where the current consumption is at most 1. Because the current consumption takes on integer values, it would otherwise be greater or equal than 2 all the time, requiring speed $v \geq 2$. Hence, at t_1 at least for one of the directions the current consumption is 0. This can only happen if the fire burns along the back of a vertical barrier and has fully consumed all barriers previously touched in this direction. For example, such a 0-interval can be seen at d_2 in Figure 1.

W. l. o. g. assume this happens on the left-hand side. By Lemma 3.1, we can also assume that barriers are strictly increasing in length in both directions. Consider the time t and total consumption $\mathcal{C}^l_{\mathcal{S}^*}(t)$ in the left direction at the beginning of a 0-interval.

$$t = \sum_{k=1}^{i} a_k + b_i + \max(0, 2b_{i-1} - b_i) \qquad \mathcal{C}^l_{\mathcal{S}^*}(t) = \sum_{k=1}^{i} a_k + \sum_{k=1}^{i} b_k - s$$

For an explanation of the $\max(0, 2b_{i-1} - b_i)$ summand, see Figure 2.

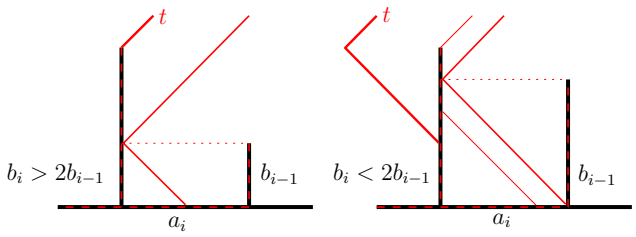


Figure 2 The two possible configurations at the beginning of a 0-interval at time t. In the first case, the fire reached the lower end of the b_i barrier first, so the 0-interval starts exactly when the upper end is reached. In the second case, the upper end was reached first, so the 0-interval begins when the lower end is reached after additional time $2b_{i-1} - b_i$.

Assume $\mathcal{Q}^l_{\mathcal{S}^*}(t) \leq 1$, then $\frac{\mathcal{C}^l_{\mathcal{S}^*}(t)}{t} \leq 1$ and $\mathcal{C}^l_{\mathcal{S}^*}(t) \leq t$

$$\begin{aligned} \Rightarrow \quad \sum_{k=1}^{i} a_k + \sum_{k=1}^{i} b_k - s &\leq \sum_{k=1}^{i} a_k + b_i + \max(0, 2b_{i-1} - b_i) \\ \Rightarrow \quad \sum_{k=1}^{i-2} b_k + b_{i-1} + b_i &\leq \max(b_i, 2b_{i-1}) + s \end{aligned}$$

Due to Lemma 3.1 $b_{i-1} + b_i > \max(b_i, 2b_{i-1})$ and $\sum_{k=1}^{i-2} b_k > (i-2) \cdot b_1 > s$ for i larger than some threshold i_0. As the left-hand current consumption is 0 for arbitrarily large t, we can assume $i > i_0$. But this is a contradiction. Hence $\mathcal{Q}^l_{\mathcal{S}^*}(t) > 1$ at the beginnings of all 0-intervals after some threshold t_0 in the left-hand direction.

Hence for the whole barrier set $\mathcal{S}^*$ to be viable for $v < 1.5$, the consumption-ratio $\mathcal{Q}^r_{\mathcal{S}^*}(t)$ in the right-hand direction must be below 0.5 at exactly these moments. Hence for the right-hand direction 0-intervals exist for arbitrarily large t.

The ends of those 0-intervals are local minima of that directions consumption-ratio. Consider the time t and total consumption $\mathcal{C}^r_{\mathcal{S}^*}(t)$ in the right-hand direction at the end of such an interval.

$$t = \sum_{k=1}^{i} c_k + d_i + \min(c_{i+1}, d_i) \qquad \mathcal{C}^r_{\mathcal{S}^*}(t) = \sum_{k=1}^{i} c_k + \sum_{k=1}^{i} d_k - s$$

For an explanation of the $\min(c_{i+1}, d_i)$ summand, see Figure 3.

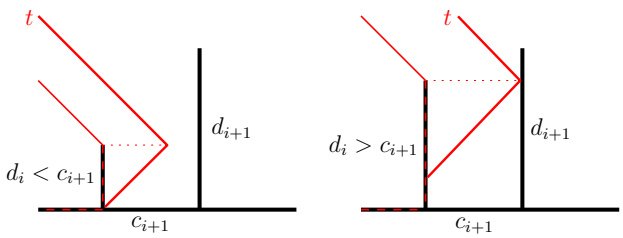


Figure 3 The two possible configurations at the end of a 0-interval at time t. In the first case, the 0-interval ends, when the fire reaches the highway, d_i time units after passing the top of the d_i barrier. In the second case, the next barrier is reached before the highway, so the 0-interval ends, c_{i+1} time units after passing the top of the d_i barrier.

Assume $\mathcal{Q}^r_{\mathcal{S}^*}(t) < 0.5$, then $\frac{\mathcal{C}^r_{\mathcal{S}^*}(t)}{t} < 0.5$ and $2 \cdot \mathcal{C}^r_{\mathcal{S}^*}(t) < t$

$$\begin{aligned} \Rightarrow \quad 2 \cdot \sum_{k=1}^{i} c_k + 2 \cdot \sum_{k=1}^{i} d_k - 2s &< \sum_{k=1}^{i} c_k + d_i + \min(c_{i+1}, d_i) \\ \Rightarrow \quad \sum_{k=1}^{i} c_k + 2 \cdot \sum_{k=1}^{i} d_k &< 2d_i + 2s \Rightarrow \sum_{k=1}^{i} c_k + 2 \cdot \sum_{k=1}^{i-1} d_k < 2s \end{aligned}$$

As above $\sum_{k=1}^{i-1} d_k > (i-1)d_1 > 2s$ for i larger than some threshold i_0. Hence after some threshold time t_0, even at the local minima the consumption-ratio of the right-hand direction is above 0.5, hence it is always above 0.5. Therefore $\mathcal{Q}_{\mathcal{S}^*}(t)$ exceeds 1.5 for arbitrarily large t, which makes $\mathcal{S}^*$ not viable for $v \leq 1.5$. ◀

4 A viable barrier set for $v = 1.8944\ldots$

Assume we want to construct a barrier set $\mathcal{S}$ improving on the naive strategy, which is viable for $v = 2$. The consumption-ratio only decreases when the current consumption is below

the current ratio. As the current consumption is an integer value, this only happens when it is at most 1, which means we have a 0-interval in at least one direction. Hence we would want these intervals to be as long as possible, which results in the following conditions:

$$\begin{aligned} a_{i+1} \geq b_i \ \wedge \ b_{i+1} \geq 2b_i \qquad & \forall i \geq 1 \\ \text{equivalently } c_{i+1} \geq d_i \ \wedge \ d_{i+1} \geq 2d_i \qquad & \forall i \geq 1 \end{aligned}$$

This forces the 0-intervals to always start exactly when the fire reaches the upper end of a vertical wall and end when the fire reaches the highway again after burning along the full length along the back of the already consumed vertical wall; see Figure 2 and 3. For a single direction this results in a repeating cycle of time intervals as seen in Figure 4. From our reasoning about the lower bound, we know that 0-intervals are necessary for both

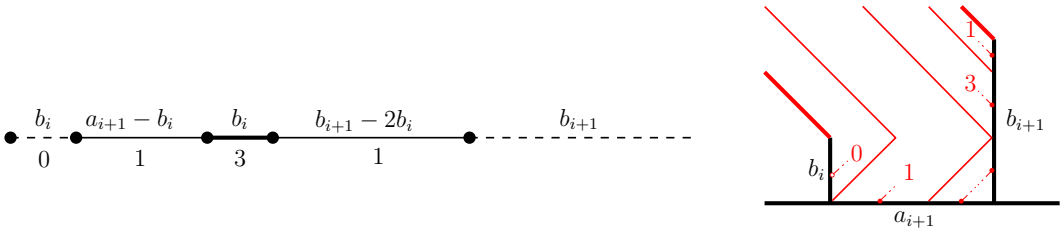


Figure 4 A repeating time interval cycle. The values $0, 1$ and 3 denote the current consumption in this direction.

directions in order to achieve $v < 2$. The idea is to construct the barrier set in such a way that the 0-intervals always appear in an alternating fashion, so the local maxima in the consumption-ratio of one direction can be countered by the 0-intervals of the other direction.

Consider the periodic interlacing of time intervals as illustrated in Figure 5, where the end point of the 0-interval from one direction coincides with the beginning of the 3-interval of the other direction.

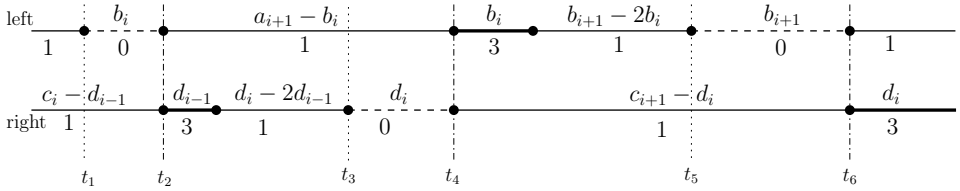


Figure 5 The periodic interlacing of time intervals.

The current consumption is always greater than 1, since the 0-intervals do not overlap. As we are trying to construct a barrier set viable for some $v < 2$, $\mathcal{Q}_{\mathcal{S}}(t)$ must be smaller than 2 at all times. Then, t_1, t_3 and t_5 have to be local maxima, while t_2, t_4 and t_6 have to be local minima of $\mathcal{Q}_{\mathcal{S}}(t)$. The idea to ensure that the global maximum of $\mathcal{Q}_{\mathcal{S}}(t)$ stays below 2 is to make all local maxima attain the same value μ and hope that it stays below 2.

So let $\mu := \mathcal{Q}_{\mathcal{S}}(t_1)$. Then $\mathcal{Q}_{\mathcal{S}}(t_3) = \mu$ if and only if, between t_1 and t_3, the ratio of consumption over time interval length is also μ. Thus,

$$\begin{aligned} & \tfrac{b_i \cdot (0+1) + d_{i-1} \cdot (1+3) + (d_i - 2d_{i-1}) \cdot (1+1)}{b_i + d_{i-1} + d_i - 2d_{i-1}} && = \mu \\ \Leftrightarrow \quad & b_i + 2d_i && = \mu(b_i + d_i - d_{i-1}) \\ \Leftrightarrow \quad & d_i && = \tfrac{\mu - 1}{2 - \mu} b_i - \tfrac{\mu}{2 - \mu} d_{i-1} \,. \end{aligned}$$

Similarly, by applying the same reasoning to t_3 and t_5, we obtain a second recursive formula $b_{i+1} = \frac{\mu-1}{2-\mu} d_i - \frac{\mu}{2-\mu} b_i$. Together these recursions can be represented by a matrix and solved by applying standard techniques. The values for a_i (and c_i) can be expressed as linear combinations of d_i and b_i – see the interval between t_2 and t_4 (t_4 and t_6) in Figure 5.

Analysing the zeros of the characteristic polynomial indicates that this recursion has real solutions for b_i and d_i as long as $\mu \geq (2+\sqrt{5})/\sqrt{5}$. For the limiting case $\mu = (2+\sqrt{5})/\sqrt{5} = 1.8944\ldots$, we get $b_i = \left(2+\sqrt{5}\right) d_{i-1} = \left(2+\sqrt{5}\right)^2 b_{i-1}$.

To prove the final theorem, it remains to find initial values to get the recursion started, while maintaining $\mathcal{Q}_{\mathcal{S}}(t) \leq \mu$. Suitable values are

$$b_i := s \cdot \left(2+\sqrt{5}\right)^{2i} \qquad \mu := \tfrac{2+\sqrt{5}}{\sqrt{5}} \approx 1.8944 \qquad c_1 := \tfrac{(\mu-2)d_1+2s+b_1}{2-\mu}$$
$$d_i := s \cdot \left(2+\sqrt{5}\right)^{2i+1} \qquad a_1 := s \qquad a_2 := 2d_1 + c_1 - a_1 - b_1,$$

which results in intervals given in Figure 6. Note that all barrier lengths only depend on s.

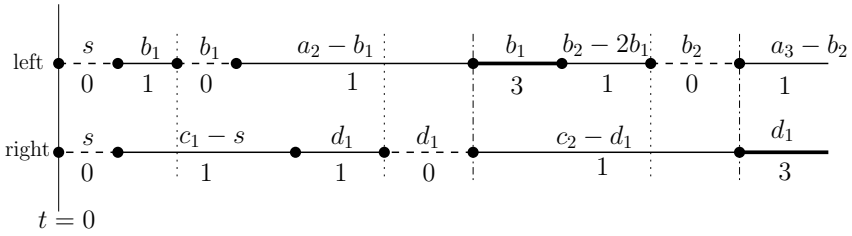


Figure 6 Illustration of time intervals for a suitable starting situation.

▶ **Theorem 4.1.** *The highway can be protected with speed* $v > \mu = \frac{2+\sqrt{5}}{\sqrt{5}} = 1.8944\ldots$

5 Conclusion

We have shown the first non-trivial bounds for the highway protection problem. One future goal is to examine whether these results can be extended to more general barrier systems and/or the L_2 plane. We expect the analysis in the L_2 plane to be more difficult, especially finding the beginning and end times of the 0-intervals.

References

1 P. Brass, K. D. Kim, H.-S. Na, and C.-S. Shin. Escaping offline searchers and isoperimetric theorems. *Computational Geometry*, 42(2):119 – 126, 2009.

2 A. Bressan. Differential inclusions and the control of forest fires. *Journal of Differential Equations*, 243(2):179 – 207, 2007.

3 A. Bressan, M. Burago, A. Friend, and J. Jou. Blocking strategies for a fire control problem. *Analysis and Applications*, 6(3):229–246, 2008.

4 S. Finbow and G. MacGillivray. The firefighter problem: A survey of results, directions and questions. Technical report, 2007.

5 F. V. Fomin, P. Heggernes, and E. J. van Leeuwen. The firefighter problem on graph classes. *Theor. Comput. Sci.*, 613(C):38–50, 2016.

6 R. Klein and E. Langetepe. Computational Geometry Column 63. *SIGACT News*, 47(2):34–39, 2016.

7 R. Klein, E. Langetepe, and C. Levcopoulos. A fire-fighter's problem. In *Proceedings 31st Symposium on Computational Geometry (SoCG'15)*, 2015.

A Fully Polynomial Time Approximation Scheme for the Smallest Diameter of Imprecise Points

Vahideh Keikha[1], Maarten Löffler[2], and Ali Mohades[3]

1 **Department of Mathematics and Computer Science, University of Sistan and Baluchestan, Zahrdan, Iran,** va.keikha@aut.ac.ir.com
2 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands,** m.loffler@uu.nl
3 **Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran,** mohades@aut.ac.ir

Abstract

Given a set $D = \{d_1, ..., d_n\}$ of imprecise points modeled as disks, the minimum diameter problem is to locate a set $P = \{p_1, ..., p_n\}$ of fixed points, where $p_i \in d_i$, such that the furthest distance between any pair of points in P is as small as possible. This introduces a tight lower bound on the size of the diameter of any instance P. In this paper, we present a fully polynomial time approximation scheme (FPTAS) for this problem that runs in $O(n^3\epsilon^{-2})$ time, where the input is a set of disjoint disks.

1 Introduction

One of the most extensively quantitative techniques used to deal with uncertainty of the input is the *Region-based model*, where the input is a set $R = \{r_1, r_2, \ldots, r_n\}$ of regions and each region represents an imprecise point. In this model, the minimum diameter problem was studied by Löffler and van Kreveld [1], who presented a PTAS in $O(n^{3\pi/\sqrt{\epsilon}})$ time, where the uncertainty of the input was modeled by arbitrary disks. In the same paper, the authors also presented an exact algorithm that runs in $O(n \log n)$ time and computes an upper bound on the diameter (maximum diameter problem). Recently, a new approximation algorithm was presented for this problem, where the uncertainty of the input is modeled by convex objects in d-dimensional space [5]. The presented algorithm runs in $O(2^{\epsilon^{-d}}\epsilon^{-2d}n^3)$ time. The minimum diameter problem is also studied in other models of uncertainty [2–4, 6, 9].

Contribution. We formulate our problem as follows. We are given a set $D = \{d_1, d_2, \ldots, d_n\}$ of imprecise points modeled as disjoint disks; choose a set $P = \{p_1, ..., p_n\}$ of points, where $p_i \in d_i$, such that the size of the diameter of P is as small as possible among all choices for each p_i.

As for the result, we present an FPTAS for this problem which runs in $O(n^3\epsilon^{-2})$ time (Section 3).

2 Preliminaries

In terms of definitions and terminology, we will follow Löffler and van Kreveld [1]. For a given set $D = \{d_1, d_2, \ldots, d_n\}$ of imprecise points modeled as disks, an *extreme disk* $d_i \in D$ has a line ℓ tangent to some point on the boundary of d_i, where no other disk can have its interior completely on the same side of d_i, unless it is tangent to ℓ, as illustrated in Figure 1. The *critical sequence* Δ_D is the set of all extreme disks of D. Without loss of generality, suppose the elements of Δ_D are ordered clock-wise. The ordered set Δ_D can be found in $O(n \log n)$ time [8].

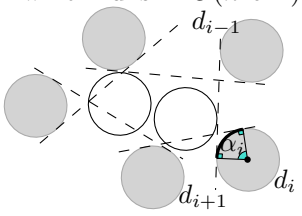


Figure 1 Critical sequence (gray disks).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

In the minimum diameter problem it is possible that the diameter occurs at several pairs at the same time, and many points are involved in the diameter, such that moving any of them would increase the distance between at least one pair of them. This property makes the problem difficult (see Figure 3(right)). In this situation, the diameter constructs a graph, the *star graph*. A star graph G is defined as $G = (P^*, E)$, where $P^* = \{b_1, ..., b_m\}$ $(m \leq n)$ is a collection of points, such that $b_i \in d_i$ for some i, and all the elements of E have equal length, which is the optimal diameter and denoted by d^*. Since this is the diameter, no two points can be more than $|d^*|$ away from each other. It follows that all the elements of E must intersect each other, and the path makes an angle of at most $60°$ at each vertex. Thus the degree of each vertex of G can be at most two, and each element of P^* (with degree 2) is in *balance* between its neighbors, that is, it could move closer to one neighbor but only by moving further from the other neighbor. Note that we can remove any vertex in this graph that can come closer to some element without moving further from another element. Also notice that all n points could be involved in such a construction, where none of the points can be moved without increasing the diameter. The elements of P^* are called *bends*. Indeed, computing the exact positions of the bends is a difficult problem. Any two adjacent disks on Δ_D introduce a common tangent line (see Figure 1). The *extreme arc* α_i of an extreme disk d_i is defined by two disks d_{i-1} and d_{i+1} that are neighbors of d_i on Δ_D, such that the endpoints of α_i are the touching points of the common tangent lines, as illustrated in Figure 1. Note that α_i is the part of the boundary of disk d_i that must contain b_i. Let r_i and c_i denote the radius and the center of d_i, respectively. The distance from a point x on disk d_i to a disk d_j is the minimum distance between x and any point on d_j. The distance between two arcs α_i and α_j (resp. two disks d_i and d_j) is the minimum distance from any point on α_i (resp. d_i) to any point on α_j (resp. d_j).

▶ **Observation 2.1.** *For a given set D of n unit disks with $n \geq 3$, the smallest diameter $|d^*|$ is at least* 0.28 *and for $n \geq 7$, $|d^*| \geq 2$.*

The observation can be easily proved by considering Figure 2.

▶ **Lemma 2.2.** *Let $D = \{d_1, ..., d_n\}$ be a set of disjoint disks, and let $|\alpha_i|$ denote the size of the constructed angle where the two lines through the endpoints of the extreme arc α_i of extreme disk d_i meet the center. Then $\sum_{i=1}^{n} |\alpha_i| = 2\pi$.*

Cherry disks. For any disk d_i that shares bend b_i on the star graph, there always exist two other disks $d_j, d_k \in \Delta_D$ with $j \neq k$, such that bend b_i is in balance between them, that is b_i could move closer to one of them but only by moving further from the other one. We call d_j and d_k the *cherry disks* of d_i. An example is illustrated in Figure 3. If a disk $d_i \in \Delta_D$ does not have two cherry disks, d_i cannot introduce a bend on the star graph. It is an interesting open question to compute all the possible cherry disks efficiently, since then the minimum diameter problem can be formulated as a second order cone program, and its optimal solution can be computed in quadratic time [7]. This results in an $O(n^2 n!)$ time exact algorithm.[1]

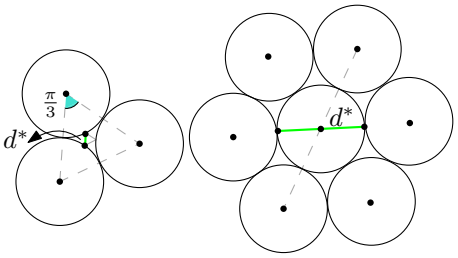


Figure 2 (left) For $n \geq 3$, $|d^*| \geq 0.28$. (right) For $n \geq 7$, $|d^*| \geq 2$.

For any disk $d_i \in \Delta_D$, let e_{ij} and e_{ik} denote the intersection of disk d_i by line segments $c_i c_j$ and $c_i c_k$, respectively (see Figure 3(right)), such that d_j and d_k are the cherry disks of

[1] In a model of computation that we can exactly compute the roots of any constant degree polynomials.

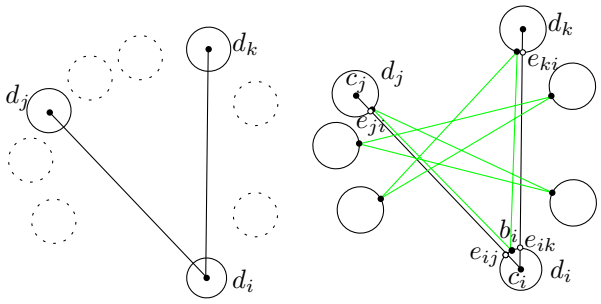


Figure 3 (left) Extreme disk $d_i \in D$ has two cherry disks d_j and d_k. (right) All the cherry disks of D; the constructed star graph on them is shown in green. Bend b_i is located in the interval $[e_{ij}, e_{ik}]$.

d_i. The points e_{ij} and e_{ik} are the startpoint and the endpoint of the arc on which bend b_i on α_i is in balance between the cherry disks of d_i (which we will prove later). We denote this interval by $[e_{ij}, e_{ik}]$. Obviously d_i is also a cherry disk for both d_j and d_k.

▶ **Lemma 2.3.** *Let d_j and d_k with $j < k$ denote the cherry disks of d_i. Then b_i is located in the interval $[e_{ij}, e_{ik}]$.*

Proof. Suppose this is false. Then bend b_i is strictly located either before e_{ij} or after e_{ik} (on the boundary of d_i). Consider the case where b_i is strictly located before the position e_{ij} (the other case is similar). In this case, at least b_j is strictly located after e_{ji} (on the cw ordering of the boundary of d_j). But then the two disks (or even one) which determine the position of b_j on d_j must be located between d_i and d_j. Let d_p and d_q denote these disks. Since b_j is strictly located after e_{ji}, at least one of d_p and d_q is different from d_i. Clearly, b_ib_k, b_jb_p and b_jb_q are some edges of the star graph. But b_jb_p and b_jb_q never intersect the edge b_ib_k. This gives a contradiction with the fact that all the edges of the star graph are pairwise intersecting. ◀

3 Minimum diameter problem

If the disks are not unit but still disjoint, Observation 2.1 holds if the smallest disk is a unit disk.

3.1 Unit disks

From Lemma 2.2 we know that if D consists of disjoint disks, the total sum of the angles of extreme arcs equals 2π. First, suppose D consists of disjoint unit disks. We proceed by *covering* the boundary of a unit disk U by all the extreme arcs of set Δ_D, such that they just intersect at the endpoints, as illustrated in Figure 4(a). This covering is indeed a translation transformation. We decompose the boundary of U into smaller, equal-length *sub-arcs* by regularly inserting $2\pi/\epsilon$ points. Then, for any disk d_i, the added points on the boundary of U which is covered by α_i will be transferred to the boundary of d_i, as illustrated in Figure 4(a). Consequently the extreme arcs get divided into sub-arcs of length at most ϵ.

Recall that P^* denote the optimal point set. Let $P' = \{p_1, ..., p_m\}$ denote the optimal point set restricted to the endpoints of the sub-arcs. The set P' minimizes the furthest distance between any pair of points on P' among all possible choices for P' [2]. Let d denote the diameter of P'. As said before, d^* denote the optimal diameter of P^*. We will show that d approximates d^* within a factor $(1 + \epsilon)$.

For any disk $d_i \in \Delta_D$, we define the *optimal sub-arc* α_i^* that includes (if any) the bend b_i. Also, α_i^* minimizes the difference of distances of the endpoints of α_i^* to the approximated cherry disks of

[2] We use the name P for the set of all the candidate points of set D, where $P' \subseteq P$. It is easy to observe that the diameter d' of P' equals the diameter d of P.

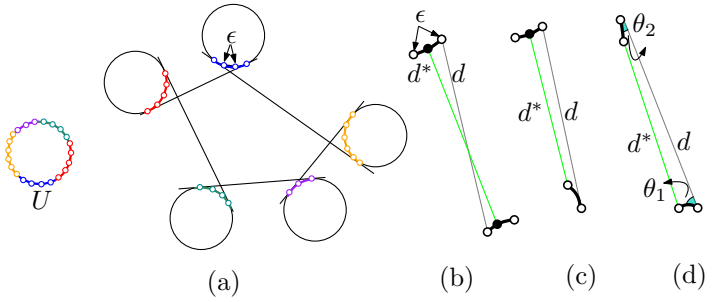


Figure 4 (a) Subdivisions of the extreme arcs into sub-arcs of length ϵ. (b-d) The configuration of the approximated diameter (gray) and the optimal diameter (green).

d_i, where the diameter which realizes by this selection of α_i^*, is as small as possible. Note that in the optimal solution, the length of each α_i^* equals 0.

We will postpone the discussion of computing the optimal sub-arcs, and we first consider how set P' approximates the minimum diameter. Note that the optimal diameter d^* at least equals the largest distance between any two optimal sub-arcs (where the distance between two arcs α_i and α_j is the minimum distance from any point on α_i to any point on α_j). Thus, we show that for any two optimal sub-arcs which include the vertices of the potential minimum diameter, the ratio of the smallest distance to the furthest distance equals $(1+\epsilon)$. There exist two configurations to consider the ratio of the smallest distance to the furthest distance of a pair of optimal sub-arcs.

- The case where d and d^* intersect each other (see Figure 4(b)). Let d_1 and d_2 (resp. d_1^* and d_2^*) denote the two segments which are determined on d (resp. d^*) by intersection with d^* (resp. d), such that d_1 and d_1^* form a triangle, where the endpoints of its base are located on an optimal sub-arc.

  Since the length of the optimal sub-arc is at most equal to ϵ, by the triangle inequality we have $|d_1^*| + \epsilon > |d_1|$ and $|d_2^*| + \epsilon > |d_2|$, and since $|d^*| \geq 2$, $|d| \leq |d^*|(1+\epsilon)$.
- The case where d and d^* do not intersect each other, in which case d^* selects its two vertices at the endpoints of its optimal sub-arcs, or d^* selects one vertex at the middle of one of its optimal sub-arcs (as illustrated in Figure 4(c,d)). Let θ_1 and θ_2 denote the angles between d and (tangents of) the optimal sub-arcs, then $|d| \leq \epsilon(\cos\theta_1 + \cos\theta_2) + |d^*|$. This again gives us $|d| \leq |d^*|(1+\epsilon)$.

Computing the optimal sub-arcs. Let m denote the number of extreme disks of D. We show that for any disk $d_i \in \Delta_D$, we can find α_i^* efficiently. For any disk $d_i \in \Delta_D$ we first select point p_i which is chosen to be one of the endpoints of the sub-arcs of α_i. This is the initialization of set P'. Then, during the algorithm, we try to move each element of P' to its best position, so that the final set P' minimizes the diameter among all possible choices for P'. Indeed, for any disk d_i we look for the optimal sub-arc α_i^*, where one of the endpoints of α_i^* determines one element of P'.

In each step of the algorithm we start by computing the diameter of P'. Let d' denote the diameter of P' with p_i and p_j as the vertices. If p_i (or p_j) is not yet in balance, we move it forward among the endpoints of the sub-arcs of α_i in the direction that the size of d' is decreasing. In each possible movement, we update the size of d', and stop moving p_i, when in the next movement, the distance of p_i to any other point p_k will be greater than the current size of d'. Let $d'' < d'$ denote the diameter with a vertex at p_j. Then we move p_j forward in the direction that the size of d'' is decreasing, and also we update the size of d'' in each movement, until in the next movement, the distance of p_j to any other point p_l is greater than the current size of d''. We also repeat this procedure for p_k and p_l, respectively, by computing the corresponding diameter with a vertex at p_k and p_l, respectively. We stop this step when we have checked/corrected the position of all the elements of P', each of which one time.

In the second step, we again start by computing the diameter of P'. We continue above procedure, until we check the position of all the elements of P'. Since the vertices of the diameter may already be in balance, it is not always possible to move them to reduce the diameter. In the following we prove that it is always possible to reduce the value of the diameter after $O(m\epsilon^{-1})$ consecutive steps of the algorithm.

In the last step of the algorithm, we only can check the position of all the elements of P', while no other movement is possible. This way we have approximated the cherry disks of any disk d_i, and

also one endpoint of the optimal sub-arc α_i^*. The other endpoint is the one which is closer to both cherry disks of d_i.

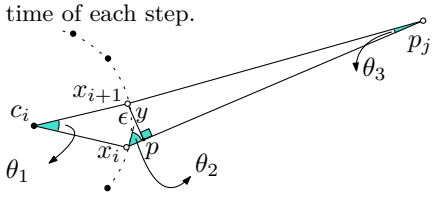
▶ **Lemma 3.1.** *After at most $O(m\epsilon^{-1})$ steps of the algorithm, the size of d' reduces by a factor of $\sqrt{2}/2$.*

Proof. In the worst case, in each step, we only could move one point p_a to its balanced position. Then in the next step, at least another point p_b, with $b \neq a$ can be moved (otherwise the algorithm will be terminated) which could not be moved in the previous step. This point can only be the point which previously was in balance between p_a and another point p_c. Then we may go back to move p_b, and then p_a and p_c, if they make a bend with its two cherry disks. Since the distances of the bend from its cherry disks are only decreasing, in at most $O(\epsilon^{-1})$ consecutive steps of the algorithm, either the algorithm stops, or we can move a new point which is distinct from p_a, p_b and p_c. Consequently, after at most $O(m\epsilon^{1-})$ steps, we have changed the position of all the elements of P', and since we have reduced all the furthest distances of the bends from the corresponding cherry disks, the size of the diameter is decreased. Now we clarify the changes on the size of the diameter that occur during the algorithm. Let p_i and p_j denote the vertices of the diameter d' which we have reduced its value. Thus we at least move one vertex of the diameter from a position x_i to x_{i+1}.

Let θ_1 denote the angle subtended by the arc with length ϵ at the center of d_i, and let θ_2 denote the determined angle by the intersection of p_jx_i and the tangent line of d_i at x_i, and let θ_3 denote the angle between p_jx_i and p_jx_{i+1}, as illustrated in the below Figure. Notice that the size of the angles θ_2 and θ_3 change during the algorithm. Also let y denote the height of triangle $x_ix_{i+1}p_j$ from the triangle's vertex x_{i+1} to the base p_jx_i, and let p denote the intersection point of y and p_jx_i. Since $|y| < |x_ix_{i+1}|$ and, $|x_ix_{i+1}| < 2$ and $|p_jx_i| = |d'| \geq 2$, the angle $|\theta_3| < 45^\circ$. Consequently $\frac{|y|}{|d'|} < \frac{\sqrt{2}}{2}$. Also since $|p_jx_{i+1}| = |d''| \geq 2$, $|x_ix_{i+1}| < 2$ and $|\theta_3| < 45^\circ$, the angle $|\theta_2| > 45^\circ$. Then we have $\frac{|y|}{|d''|} < \frac{\sqrt{2}}{2}$ and $\frac{|x_ip|}{|x_ix_{i+1}|} < \frac{\sqrt{2}}{2}$, and thus $\frac{|x_ip|\cdot|y|}{|x_ix_{i+1}|\cdot|d''|} < \frac{2}{4}$. Since $\frac{|y|}{|x_ix_{i+1}|} < \frac{\sqrt{2}}{2}$, $|x_ip| < \frac{1}{\sqrt{2}}|d''|$ and since $|d''| < |d'|$ we have $|x_ip| < \frac{1}{\sqrt{2}}|d'|$ [3]. ◀

The importance of the reduced value from the diameter is on the convergence of the iterative process. Also $2 \leq |d^*| < |d_{max}|$, where d_{max} denote the maximum diameter of D (it can be computed in $O(n \log n)$ time [1]). Obviously the same bound also holds for d'. Consequently, the algorithm will be terminated after $O(m\epsilon^{-1}(\log_{\sqrt{2}}|d_{max}|)$ steps. Since $\log_{\sqrt{2}}|d_{max}|$ is a constant, we omit it from the total running time. Now we consider the running time of each step.

Since we know the cw order of the elements of P', the diameter of P' can be computed in linear time in each step of the algorithm. In each movement of any element p_i of P', we should be careful for not increasing the size of the diameter with a vertex at p_i. Thus we costs $O(m + m\epsilon^{-1})$ for each element in one step. The later m is the time costs to check whether the corresponding element gets in balance or not. Thus the algorithm takes $O(m^3\epsilon^{-1}(1 + \epsilon^{-1}))$ time.



▶ **Lemma 3.2.** *For any disk $d_i \in \Delta_D$, computed α_i^* includes possible bend b_i.*

Proof. Suppose this is false. Then b_i is located on a sub-arc α_i' which is distinct from α_i^*. Then either we have passed over this sub-arc during the algorithm, or we did not find it and we stopped. Let d_j and d_k denote the approximated cherry disks of d_i by the algorithm.

In the first case, at both endpoints of α_i', the computed distance of d_i to both d_j and d_k must be greater than our current diameter, while we have found a solution with strictly a smaller size. This contradicts the optimality of the computed minimum diameter.

In the second case, since α_i' and α_i^* are distinct, at least one computed cherry disk for α_i' has to be distinct from d_j or d_k (if not; $\alpha_i' = \alpha_i^*$, and we are done). But then we could move p_i to

[3] Notice that this lemma holds for a set of arbitrary disks where the smallest disk is a unit disk, since we do not let the length of the sub-arcs (which is ϵ) on the smallest disk to be as large as π, $|y|$ must always be smaller than 2.

reduce the distance of d_i to at least one of d_j and d_k. This contradicts the stop criterion of the algorithm. ◀

3.2 Disks with different size

In the case where D consists of arbitrary disjoint disks, the total sum of the angles of the extreme arcs still equals 2π, but with the idea we used on unit disks, the optimal sub-arcs will not necessarily have the same length. In this case, we first apply the presented constant factor approximation algorithm [1] for the minimum diameter problem on a set of disks. The presented algorithm approximates the smallest diameter within a constant factor c in linear time. Let d' denote the approximated smallest diameter of D within factor c. We define $h = \epsilon \cdot c \cdot |d'|$ as the new length of the sub-arcs. Note that if h is greater than the extreme arc of a disk d_i, we consider α_i instead of a sub-arc of length h. In this case, the total sum of the lengths of the extreme arcs is bounded by $|2\pi(cd'/2)|$, since any circle whose radius is greater than $|cd'/2|$ will share an extreme arc with less curvature (and thus with less arc length), also any circle whose radius is less than $|cd'/2|$ will share an extreme arc with a shorter arc length (see Figure 5). Thus the maximum number of the points that approximates the extreme arcs is bounded by $\frac{2\pi(cd'/2)}{h}$, which is in $O(\epsilon^{-1})$. Since computed sub-arcs admit the same length h, the considered ratio of the furthest distance to the smallest distance between the optimal sub-arcs (in Section 3.1) still holds, and the presented algorithm works in $O(n^3\epsilon^{-2})$ time.

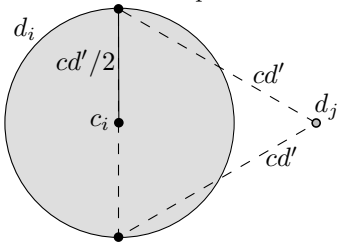


Figure 5 The maximum possible length for an extreme arc appears between two disks d_i and d_j with $|r_i| = |cd'|/2$ and $|r_j| \approx 0$. Thus the total sum of the extreme arcs is bounded by $|2\pi(cd'/2)|$. Note that the subtended angle of a sub-arc cannot equals π, it is supposed so to compute the upper bound.

▶ **Theorem 3.3.** *Given a set of n disjoint disks, the problem of choosing a point on the boundary of each disk such that the diameter of the resulting point set is as small as possible can be approximated within a factor $(1+\epsilon)$ in $O(n^3\epsilon^{-2})$ time.*

References

1 Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419 – 433, 2010.

2 P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

3 M. E. Consuegra and G. Narasimhan. Geometric avatar problems. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 24, 2013.

4 R. Fleischer and X. Xu. Computing minimum diameter color-spanning sets is hard. *Information Processing Letters*, 111(21):1054–1056, 2011.

5 M. Ghodsi, H. Homapour, and M. Seddighin. *Approximate Minimum Diameter*, pages 237–249. Springer International Publishing, Cham, 2017.

6 W. Ju, C. Fan, J. Luo, B. Zhu, and O. Daescu. On some geometric problems of color-spanning sets. *Journal of Combinatorial Optimization*, 26(2):266–283, 2013.

7 M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998.

8 T. Nagai, S. Yasutome, and N. Tokura. Convex hull problem with imprecise input and its solution. *Systems and Computers in Japan*, 30(3):31–42, 1999.

9 D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *Data Engineering*, pages 688–699, 2009.

A new algorithm for finding polygonal voids in Delaunay triangulations and its parallelization

José Ojeda[1], Rodrigo Alonso[1], and Nancy Hitschfeld-Kahler[1]

1 **Department of Computer Science, FCFM, University of Chile**
jojeda@dcc.uchile.cl, ralonso@dcc.uchile.cl, nancy@dcc.uchile.cl*

Abstract

A known geometrical problem is to find low density zones (voids) in planar point sets and to represent them as polygons. In this paper we recall the concept of terminal-edge region to identify subvoid candidates over a triangulation, present a linear algorithm to find subvoids taking as input a Delaunay triangulation, and show that this new strategy can be naturally parallelized using GPU computing. We also show preliminary experimental results.

1 Introduction

A real geometrical problem is to find underdense zones in planar point sets and represent their shapes by polygons. This problem is particularly relevant in astronomy, where the regions almost empty of bright galaxies are known as cosmological voids. In computational geometry, a similar and well studied problem is to find large convex holes in a planar point set P. A convex hole is represented by a convex polygon that contains no point of P in its interior. The key question is the expected size (number of vertices) of the existent convex polygons [2, 8]. Convex polygons solutions would not generally apply to the problem in astronomy, where the cosmological voids are usually not convex and may contain a few galaxies in its interior. The computation of the α-shape of a set of points [3], which is a generalization of the convex hull of it can also be seen as a related problem. The original goal of the α-shape algorithm is to represent the shape of the set of points and not to compute the empty spaces inside the convex hull of the point set but it has been also used to study the topology of the cosmic web [11].
In the last years, a new algorithm was developed to detect and build polygons representing underdense regions or voids on a planar point set [1, 4]. The algorithm starts from terminal-edges (local longest-edges) in a Delaunay triangulation and builds the largest possible low density terminal-edge regions around them. A terminal-edge region can represent either an entire void or part of a void (subvoid). The algorithm is divided in two main steps: (1) *Building subvoids* and (2) *Joining subvoids*. In the first step, each triangle is attached to the terminal-edge region (a triangle set) it belongs to. In the second step, fragmented voids, represented by several terminal-edge regions, are joined to build whole voids. Each void is described by a polygon defined by the edges which appear only once in the region.
In this paper, we (a) summarize the foundations of the *Building subvoids* step introduced in [1], (b) present the design of a linear algorithm to find terminal-edge regions starting from a Delaunay triangulation, and (c) show that this new solution can be naturally parallelized using GPU computing. We also show preliminary experimental results.

2 Problem Statement and Solution

In this section we summarize the concepts and some theoretical foundations of the algorithm to find underdense regions (voids) published in [1]. These concepts are required to understand

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

the new algorithm presented in § 3. It is worth to mention that a region whose interior point density is lower than the background density is considered a void candidate. Each application must define the corresponding threshold density value.

2.1 Main Approach

A Delaunay triangulation is geometric structure that allows one to find naturally the input points (sites) that are (locally) close or far from each other. By definition, an edge is a Delaunay edge, if there exists an empty circle that passes through its endpoints. If the empty circles are large enough, they are candidates to be part of a void. That is why a Delaunay triangulation (in time $O(n \log n)$) over the input points is generated first and used later as input to look for subvoids. As basic strategy, the algorithm described in [1] builds empty regions around local longest-edges. Each two triangles that share "large enough" local longest-edge will be part of a subvoid and so define the boundaries of initial empty polygons. The question is now which other triangles should be added to these two initial triangles to form a larger empty polygon. The strategy includes triangles that are adjacent to the two initial triangles and that share their longest-edge with these two initial triangles. Then the empty polygon will be now defined by the two initial triangles plus the neighbor triangles that fulfill this criterion. This process is repeated for the newly added triangles until no other triangle can be added. Notice that this criterion avoids going from a large empty region to another large empty region trough a zone like a tunnel or a region with higher point density [4].

2.2 Basic Definitions

Let T be a conforming triangulation of n points and m triangles. Let t_i be a triangle, $i = 0, ..., m-1$. Each triangle has a unique *longest-edge* and a *shortest-edge* except for isosceles and equilateral triangles. We assume that there are neither isosceles nor equilateral triangles but if they would exist, the equal length edges are arbitrarily ordered. The terms *Longest-edge propagation path* and *terminal-edge* were first introduced by Rivara in the context of new algorithms for the refinement/improvement of triangulations [9]. In [1] these terms are used to define and characterize the *terminal-edge regions*, that is, the subvoid candidates. That is why they are also recalled here.

▶ **Definition 2.1.** Terminal-triangles and -edges [10]: Two triangles are called *terminal triangles* if they share their longest-edge. This shared longest-edge is called *terminal-edge*.

▶ **Definition 2.2.** Longest-edge propagation path [10]: For any triangle t_0 in any conforming triangulation T, the *Longest-Edge Propagation Path* of t_0 (Lepp(t_0)) is the ordered list of all the triangles t_0 , t_1, t_2, ..., t_{l-1}, t_l, such that t_i is the neighbor triangle of t_{i-1} by the longest-edge of t_{i-1}, for $i = 1, 2, \ldots, l$. The longest-edge shared by t_{l-1} and t_l is a terminal-edge and t_{l-1} and t_l are terminal-triangles.

▶ **Definition 2.3.** Boundary edge and boundary terminal-edge: The edges that belong to only one triangle of a triangulation are called *boundary edges*. If a boundary edge is the longest-edge of the triangle, it will be called *boundary terminal-edge*.

2.3 Algorithm Foundations

In this section we recall the definitions and theorems published in [1] required to understand the new sequential algorithm proposed in § 3.1 and the parallel algorithm presented in § 3.2. Theorem 2.9 was extended to show that terminal-edges regions cover the whole space.

▶ **Definition 2.4.** Terminal-edge region: A *terminal-edge region* R is a region formed by the union of all triangles t_i such that Lepp(t_i) has the same terminal-edge. In case the terminal-edge is a boundary-edge the region will be called *boundary terminal-edge region*. As illustration see Figure 1.

▶ **Definition 2.5.** Frontier-edge: A *frontier-edge* is an edge that is shared by two triangles, each one belonging to a different terminal-edge region.

▶ **Lemma 2.6.** *Let t_i and t_j be two triangles that share the edge e. If the edge e is a frontier-edge, then e is neither the longest-edge of t_i nor of t_j.*

Proof: See [1].
Note that the opposite is false. It is possible that two triangles t_i, t_j whose shared edge is not the longest-edge of any of them and these two triangles belong to the same terminal-edge region R_i. e is still a particular case frontier-edge in the sense that the region R_i can not grow through it. Since e does not separate two different regions, it was called a *barrier-edge*.

▶ **Definition 2.7.** Barrier-edge: A *barrier-edge* of a region R is an edge shared by two triangles of R which is not the longest-edge of any of them.

▶ **Definition 2.8.** Internal-edge: An *internal-edge* of a region R is an edge that is neither a terminal-edge, a frontier-edge, a barrier-edge nor a boundary-edge.

▶ **Theorem 2.9.** *Let T be a conforming triangulation of any set of points P and let CH be the convex hull of P. Then T can be partitioned into a set of terminal-edge regions covering the whole convex hull area as shown in Figure 1, and without overlapping.*

Proof: In [1] was demonstrated that terminal-edge regions do not overlap, then we only need to show that terminal-edge regions cover P. Let us assume that the triangulation T has n triangles and m terminal-edges $e_i, i = 0, \ldots, m-1$. e_i can also be a boundary terminal-edge. By definition 2.4, each terminal-edge e_i has associated a terminal-edge region R_i. Let assume that there exists a triangle $t_j, j = 0, \ldots, n-1$ which does not belong to any terminal-edge region $R_i, i = 0, \ldots, m-1$. By definition 2.2 each triangle t_j has a Lepp(t_j) and so an associated terminal-edge e. Then t_j must be included in the terminal-edge region associated to e and this fact contradicts our assumption. □

3 The New Algorithm

The algorithms presented in this section decrease the computational cost of the *Building subvoids* step published in [1]. The original algorithm sorts the triangles of the Delaunay triangulation by their longest-edge, and so computes the terminal-edge regions associated to the largest-terminal edges first. In this way, the *Building subvoids* step is done in $O(n \log n)$, where n is the number of points. The new algorithm works over a graph representation $G = (V, E)$ of a triangulation [7] and builds the terminal-edge regions in $O(n)$. In G each node $v \in V$ represents a triangle and E contains the adjacency relations. More specifically, arc $(u, v) \in E$ if and only if the triangles represented by the nodes u and v are adjacent in the actual triangulation.

3.1 Sequential Algorithm

Algorithm 1 partitions the adjacency graph $G = (V, E)$ into terminal-edge regions by removing arcs of E associated with adjacencies between triangles sharing frontier-edges. The graph is

divided into terminal-edge regions, each one a subvoid candidate. The regions are classified into boundary subvoids or subvoids. Algorithm 2 uses a simple navigation strategy to compute the area and to check if it is a boundary component or not. The *Building subvoids* step is now in $O(n)$.

Input: A adjacency graph $G = (V, E)$ of the Delaunay triangulation
Input: A *threshold_value*
foreach $v \in V$ **do**
  Label the longest-edge in triangle related to v
foreach $(u, v) \in E$ **do**
  if the triangles related to u and v share a frontier edge **then**
    $E \leftarrow E \setminus \{(u, v)\}$
foreach $v \in V$ **do**
  if v is not visited yet **then**
    $total\_area, touches\_border \leftarrow$ **Visit**(G, v)
    if $total\_area \geq threshold\_value$ **then**
      if *touches_border* **then**
        Component type of $v \leftarrow$ Boundary subvoid candidate
      else
        Component type of $v \leftarrow$ Subvoid candidate
return $G = (V, E)$

Algorithm 1: *Building subvoids* algorithm

3.2 Parallel Algorithm

The sequential algorithm is naturally parallelizable on GPU because the computation of terminal-edge regions can be partitioned into $O(n)$ smaller tasks and solved by threads per arc and threads per nodes using similar data structures as the ones described in [5, 6]. Currently, the tasks per thread are $O(1)$ except for the area computation (Algorithm 2), which in the worst case is $O(n)$. The steps of the kernels are:

1. Kernel Initialization
   - Create a thread per node v (triangle)
   - Each thread labels its triangle longest-edge
   - Each thread computes its triangle area
2. Kernel Generation of terminal-edge regions
   - Create a thread per arc e
   - Each thread eliminates arc e if the shared edge is a frontier-edge
   - Each thread labels the shared edge as terminal-edge if corresponds
3. Kernel Terminal-edge regions classification
   - Create a thread per terminal-edge region
   - Each thread computes the area of its terminal-edge region
   - Each thread labels its region as subvoid, boundary subvoid or null according to a threshold value.

Input: $G = (V, E)$ the graph
Input: A node $v \in V$ belonging to a terminal-edge region
1 Mark v as visited
2 $total\_area \leftarrow$ Area of the triangle related to v
3 $touches\_border \leftarrow$ Triangle related to v is a boundary triangle
4 **foreach** $(u, v) \in E$ **do**
5 **if** u is not visited yet **then**
6 $u\_total\_area, u\_touches\_border \leftarrow$ **visit**(G, u)
7 $total\_area \leftarrow total\_area + u\_total\_area$
8 $touches\_border \leftarrow touches\_border \vee u\_touches\_border$
9 **return** *total_area*,*touches_border*

Algorithm 2: The **Visit** subroutine

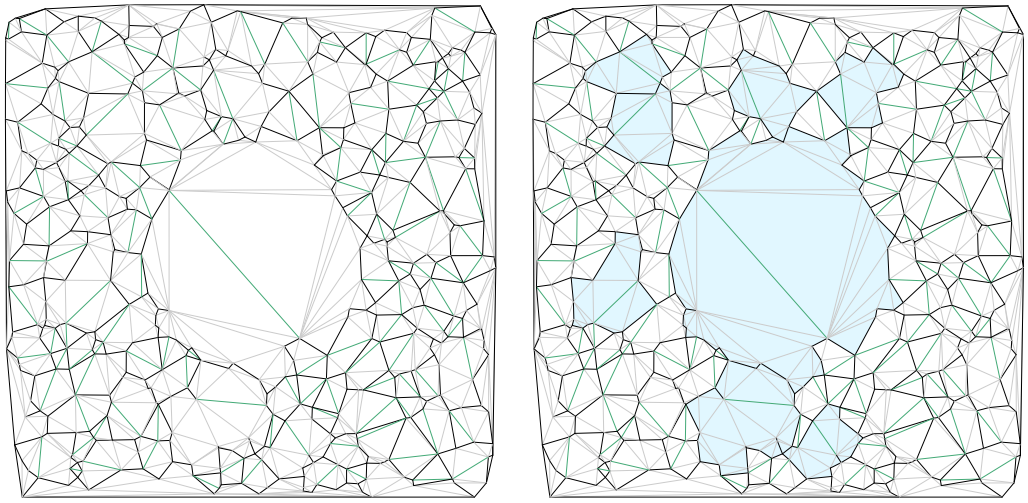


Figure 1 Terminal-edge regions in a Delaunay triangulation. The green edges are terminal-edges, the light blue are internal edges and the black edges are frontier edges. At the left, each terminal-edge region is delimited by the black edges, and at the right, the light blue regions are the ones with area greater than a threshold value. These are subvoids candidates.

4 Empirical Evaluation

Figure 2 shows a comparison of the three implementations of the *Building subvoids* step. The x-axis shows the number of points and the y-axis the classification time. The green line represents the performance of the original solution written in Python [1], the blue line the time of Algorithm 1 in § 3.1 and the red line, the performance of the parallel implementation described in § 3.2. Both new implementations are written in C and Opencl.

5 Conclusions and Ongoing work

We have presented a new algorithm to find terminal-edge regions (subvoids candidates) in planar points sets. This new strategy allowed us to propose sequential and parallel algorithms which have a lower time complexity than the approach published in [1]. The parallel strategy is still in the worst case $O(n)$ due to the linear computation of each terminal-edge region area, but this computation can also be improved by parallelizing inside each terminal-edge

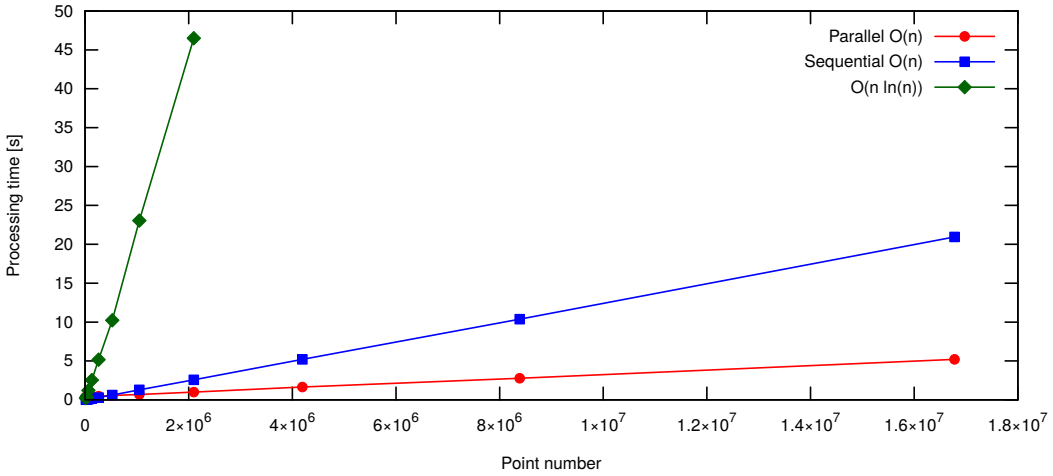


Figure 2 Performance comparison on a multicore architecture.

region. We also plan to compare our approach with prior work in related problems [11].

References

1 R. Alonso, J. Ojeda, N. Hitschfeld, C. Hervías, and L.E. Campusano. Delaunay based algorithm for finding polygonal voids in planar point sets. *Astronomy and Computing*, 22:48 – 62, 2018.

2 József Balogh, Hernán González-Aguilar, and Gelasio Salazar. Large convex holes in random point sets. *Comput. Geom.*, 46(6):725–733, 2013.

3 Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–558, 1983.

4 Carlos Hervías, Nancy Hitschfeld-Kahler, Luis E. Campusano, and Giselle Font. On finding large polygonal voids using Delaunay triangulation: The case of planar point sets. In *Proceedings of the 22nd International Meshing Roundtable*, pages 275–292, 2013.

5 C. A. Navarro, N. Hitschfeld-Kahler, and E. Scheihing. A GPU-based method for generating quasi-Delaunay triangulations based on edge-flips. In *GRAPP/IVAPP*, pages 27–34, 2013.

6 Cristobal Navarro, Nancy Hitschfeld-Kahler, and Eliana Scheihing. A parallel GPU-based algorithm for Delaunay edge-flips. In *Abstracts from 27th European Workshop on Computational Geometry (EUROCG2011)*, pages 75–78. Switzerland, March 28-30, 2011.

7 Joseph O'Rourke. *Computational Geometry in C.* Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

8 Rom Pinchasi, Rados Radoicic, and Micha Sharir. On empty convex polygons in a planar point set. *J. Comb. Theory, Ser. A*, 113(3):385–419, 2006.

9 M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *Int. Jour. for Num. Meth. in Eng.*, 40:3313–3324, 1997.

10 M. C. Rivara, N. Hitschfeld, and R. B. Simpson. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. *CAD journal*, 33:263–277, 2001.

11 Rien van de Weygaert, Gert Vegter, Herbert Edelsbrunner, Bernard J. T. Jones, Pratyush Pranav, Changbom Park, Wojciech A. Hellwing, Bob Eldering, Nico Kruithof, E. G. P. (Patrick) Bos, Johan Hidding, Job Feldbrugge, Eline ten Have, Matti van Engelen, Manuel Caroli, and Monique Teillaud. Alpha, betti and the megaparsec universe: On the topology of the cosmic web. *Trans. Computational Science*, 14:60–101, 2011.

$L(2,1)$-labeling of disk intersection graphs

Konstanty Junosza-Szaniawski and Joanna Sokół*

Faculty of Mathematics and Information Science,
Warsaw University of Technology, Poland
{k.szaniawski, j.sokol}@mini.pw.edu.pl

Abstract

In this paper we study the problem of $L(2,1)$-labeling of intersection graphs of disks. An $L(2,1)$-labeling is a mapping from the vertex set of the graph to non-negative integers, in which labels assigned to adjacent vertices differ by at least 2, and labels assigned to vertices at distance 2 are different. The span of an $L(2,1)$-labeling is the difference between the maximum and the minimum label used, and the span $\lambda(G)$ of a graph G is the minimum span of an $L(2,1)$-labeling of G. We show that if G is an intersection graph of disks, then $\lambda(G) \leq \frac{4}{5}\Delta(G)^2 + 25\Delta(G) + 22$, where $\Delta(G)$ denotes the maximum degree in graph G.

1 Introduction

Problems arising in frequency assignment in radio networks gave raise to many interesting graph-theoretical questions, especially concerning various variants of graph coloring. Notable and well-studied members of a big family of such problems are channel assignment problem [14], T-coloring [19], distance-constrained labeling [9, 14, 19], and $L(p,q)$-labeling [8, 2]. They are interesting for their potential applications [23], and purely theoretical properties.

In this paper, we consider one of such problems, i.e. the $L(2,1)$-labeling problem. It asks for a vertex labeling with non-negative integers, in which adjacent vertices get labels that differ by at least two, and vertices at distance two get different labels. The chromatic parameter related to this problem is the $L(2,1)$-*span* of a graph G, denoted by $\lambda(G)$, which is the minimum possible difference between the largest and the smallest label used by an $L(2,1)$-labeling of G. Griggs and Yeh [8] showed that for every G it holds that $\lambda(G) \leq \Delta(G)^2 + 2\Delta(G)$, where $\Delta(G)$ denotes the maximum vertex degree in G. Moreover, they conjectured that $\lambda(G) \leq \Delta(G)^2$ for every graph G with $\Delta(G) \geq 2$. This conjecture attracted a considerable attention and upper bounds were successfully improved, e.g. Gonçalves [6] showed an algorithm finding and $L(2,1)$-labeling of any graph with $\Delta(G) \geq 3$, whose span is at most $\Delta(G)^2 + \Delta(G) - 2$. Using a non-constructive method, Havet *et al.* [10] settled the "delta-square conjecture" in affirmative for graphs with $\Delta(G) \geq 10^{69}$. For graphs with smaller maximum degree the problem remains open. Another open direction is finding a constructive proof of the conjecture.

What makes the delta-square conjecture even more interesting is the fact that we know only two graphs G, which satisfy the equality $\lambda(G) = \Delta(G)^2$: they are the Petersen graph and the Singleton-Hoffman graph. Recently Lu [15] presented an infinite family of graphs G with $\lambda(G) = \Delta(G)^2 - \Delta(G) + 1$, which is the largest value for any known infinite family.

Besides the results for general graphs, also restricted graph classes received a considerable attention. For example it is known that $\lambda(G) \leq \Delta(G)+2$ if G is a tree [8], $\lambda(G) \leq 2\Delta(G)+23$ if G is planar [22], and $\lambda(G) \leq \frac{d-2}{d-1}\Delta(G)^2+2\Delta(G)$, if G is $K_{1,d}$-free [20]. We refer the reader to the survey by Calamoneri [2] for more information about $L(2,1)$-labeling and related problems.

* Joanna Sokół was partially supported by the National Science Center of Poland under grant no. 2016/23/N/ST1/03181.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

In this paper we focus on the class of intersection graphs of disks in the Euclidean plane called disk graphs or DG in short. This class with its subclass where all disks are required to have equal diameter (shortly denoted as UDG) are among the most extensively studied classes of geometric intersection graphs, both from the combinatorial and the algorithmic point of view [3, 5, 7, 18]. They are also especially interesting and natural in the context of $L(2,1)$-labeling, since they are the simplest class used for modeling radio networks [16, 17, 23]. Since unit disk graphs are $K_{1,6}$-free (see Fig.1), the bound by Shao *et al.* [20] implies that $\lambda(G) \leq \frac{4}{5}\Delta(G)^2 + 2\Delta(G)$. Fiala *et al.* [4] considered offline and online algorithms for $L(2,1)$-labeling of disk and unit disk graphs. Among other results, they have shown that $\lambda(G) \leq 18\omega(G)$ for $G \in UDG$, where $\omega(G)$ denotes the cardinality of the largest clique in G. This clearly implies that $\lambda(G) \leq 18\Delta(G) + 18$ for such graphs.

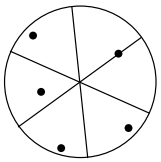


Figure 1 A maximal independent set in the neighborhood of a vertex of UDG. (Points represent the centers of the disks. Disks with centers in each region form a clique. For any point we can rotate the partition so that it is on the boundary of two regions, and then we can only add 4 other points to the independent set.)

In this paper we continue the line of research started by Fiala *et al.* [4], Shao *et al.* [20] and Junosza-Szaniawski *et al.* [12]. We solve the delta-square conjecture for disk graphs with maximum degree at least 126, by proving the following theorem.

▶ **Theorem 1.1.** *For any disk graph G with maximum degree $\Delta(G)$ we have $\lambda(G) \leq \frac{4}{5}\Delta(G)^2 + 25\Delta(G) + 22$.*

To the best of our knowledge, this is the first non-trivial upper bound for $\lambda(G)$ if G is a disk intersection graph, without any further assumption on the radii of the disk in a geometric representation.

Throughout the paper, we assume that the input disk intersection graph is given along with its geometric representation. Note that this assumption is important, as the problems of recognizing unit disk graphs [1] and disk graphs [11] are NP-hard. Actually, the problem of recognizing unit disk graphs is known to be $\exists\mathbb{R}$-complete [13], which is a strong evidence that it may not even be in NP.

2 Preliminaries

For a graph $G = (V, E)$, by $\Delta(G)$ and $\omega(G)$ we denote, respectively, the maximum degree and the size of the maximum clique in G. By $\bar{G}$ we denote the *complement* of G, i.e. a graph with the vertex set V and the edge set $\binom{V}{2} \setminus E$. For vertices u, v of G, by $\mathrm{d}_G(u,v)$ we denote the number of edges on the shortest u-v–path in G (i.e., the distance between u and v in the graph G). By $N(v)$ we denote the *neighborhood* of the vertex v, i.e. the set of vertices u with $\mathrm{d}_G(u,v) = 1$. By $N^2(v)$ we denote the set of vertices u with $\mathrm{d}_G(u,v) = 2$.

A function $c\colon V \to \mathbb{N}_0$ is called an $L(2,1)$-*labeling* of $G = (V, E)$, if

1. $|c(v) - c(w)| \geq 1$ for all $v, w \in V$ such that $\mathrm{d}_G(u,w) = 2$,
2. $|c(v) - c(w)| \geq 2$ for all $v, w \in V$ such that $\mathrm{d}_G(v,w) = 1$.

A *span* of an $L(2,1)$-labeling c of G is the difference between the maximum and the minimum label used by c (note that some labels may not be used at all). An $L(2,1)$-*span* of G, denoted by $\lambda(G)$, is the minimum possible span in an $L(2,1)$-labeling of G. Note that the number of labels that might be used in an $L(2,1)$-labeling with the minimum span is $\lambda(G) + 1$.

For $u, v \in \mathbb{R}^2$, by $\mathrm{dist}(v,u)$ we denote the euclidean distance between u and v. For $r \in \mathbb{R}$ and $v \in \mathbb{R}^2$ by $D(v,r)$ we denote the set $\{p \in \mathbb{R}^2 : \mathrm{dist}(v,p) \leq r\}$, i.e., the disk with a center

in v and the radius r. For a set $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ of disks in the plane, we define their *intersection graph* $G = (V, E)$, whose vertex set is $\{v_1, v_2, \ldots, v_n\}$, and the vertices v_iv_j are adjacent if and only if $D_i \cap D_j \neq \emptyset$. We will often identify the vertices of G with the centers of the disks in $\mathcal{D}$. Notice that $v_iv_j \in E$ if and only if $\text{dist}(v_i, v_j)$ is at most the sum of the radii of D_i and D_j.

For such a graph G, the set $\mathcal{D}$ is called *geometric representation by intersection disks*, or a *representation* in short. A graph G is called a *disk intersection graph*, or a *disk graph* in short, if it has a geometric representation by intersecting disks. If a graph G admits a geometric representation, where all disks have the same radius, we say that G is a *unit disk intersection graph*, or *unit disk graph*. The classes of disks intersection graphs and unit disks intersection graphs are denoted by DG and UDG, respectively.

We will also use the celebrated Turán Theorem.

▶ **Theorem 2.1** (Turán [21]). *For integers $d \geq p$, every K_p-free graph with d vertices has at most $\frac{p-2}{p-1}\frac{d^2}{2}$ edges.*

3 General disk intersection graphs

In this section we prove Theorem 1.1. Consider a disk graph $G = (V, E)$ along with its geometric representation by the collection of intersecting disks $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Let v_i be the center and r_i be the radius of D_i. We will identify points $v_1, v_2, \ldots, v_n$ with the corresponding vertices of G. Moreover, we assume that disks are ordered by non-increasing radius, i.e. $r_1 \geq r_2 \geq \ldots \geq r_n$.

Consider any vertex $v_i \in V$. We start with defining two special types of vertices in $N^2(v_i)$. We say that v_j is an LL-neighbor of v_i, or, equivalently, $v_j \in N^2_{LL}(v_i)$, if $v_j \in N^2(v_i)$, $r_j \geq r_i$ and there exists a disk D_k intersecting both D_i and D_j such that $r_k \geq r_i$ ("LL" stands for Large-Large, as both D_j and D_k are larger than D_i). Analogously, we say that v_j is an SL-neighbor of v_i, or $v_j \in N^2_{SL}(v_i)$, if $v_j \in N^2(v_i)$, v_j is not an LL-neighbor of v_i, $r_j \geq r_i$, and there exists a disk D_k intersecting both D_i and D_j such that $r_k < r_i$ (here "SL" stands for Small-Large, as D_k is smaller than D_i and D_j is larger than D_i).

Now we want to bound the cardinalities of $N^2_{SL}(v_i)$ (in Lemma 3.1) and $N^2_{LL}(v_i)$ (in Lemma 3.2) of each vertex v_i of G.

▶ **Lemma 3.1.** *$|N^2_{SL}(v_i)| \leq 22\omega(G)$, for any vertex v_i in a disk graph G.*

Proof. Let $v_j \in N^2_{SL}(v_i)$. Since D_i and D_j do not intersect, we have $dist(v_i, v_j) > r_i + r_j \geq 2r_i$. On the other hand there exists a disk D_k intersecting both D_i and D_j, such that $r_k \leq r_i$. Thus we obtain

$$\begin{aligned} dist(v_i, v_j) \leq & dist(v_i, v_k) + dist(v_k, v_j) \leq (r_i + r_k) + (r_k + r_j) \\ \leq & r_i + 2r_k + r_j \leq 3r_i + r_j. \end{aligned} \tag{$\star$}$$

We partition $\mathbb{R}^2 - D(v_i, 2r_i)$ into 22 regions, which will form cliques, in the following manner. First we divide $\mathbb{R}^2 - D(v_i, 2r_i)$ by a circle with the center in v_i and radius $2t \cdot r_i$, where $t = \frac{1}{\sqrt{2-\sqrt{2}}} \approx 1.3$. Then we partition the ring $D(v_i, 2tr_i) - D(v_i, 2r_i)$ into 8 congruent bounded regions, and $\mathbb{R}^2 - D(v_i, 2tr_i)$ into 14 congruent unbounded regions, as presented on Figure 2. In the remainder of the proof, we will show that disks with the radius at least r_i, whose centers lie inside one region, form a clique. Clearly this will imply the lemma. It

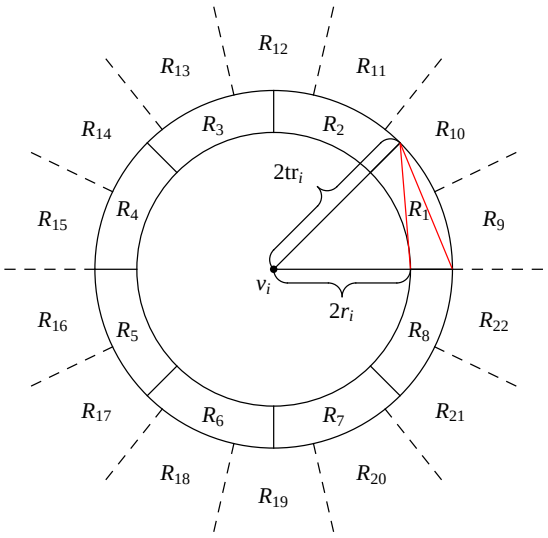


Figure 2 Partition of $\mathbb{R}^2 - D(v_i, 2r_i)$ into 22 regions: $R_1, R_2, \ldots, R_{22}$, $(t = \frac{1}{\sqrt{2-\sqrt{2}}})$.

is straightforward to verify that the diameter of each bounded region is at most

$$\begin{aligned}\text{diam} &= \max\left\{2r_i\sqrt{\left(t\cos\frac{\pi}{4}-1\right)^2+\left(t\sin\frac{\pi}{4}\right)^2}, 2t_i\sqrt{\left(t\cos\frac{\pi}{4}-t\right)^2+\left(t\sin\frac{\pi}{4}\right)^2}\right\}\\ &= \max\left\{2r_i\sqrt{t^2-t\sqrt{2}+1}, 2r_i\sqrt{(2-\sqrt{2})t^2}\right\}.\end{aligned}$$

Since $t = \frac{1}{\sqrt{2-\sqrt{2}}}$, we obtain $diam = \max\left\{2r_i\sqrt{\frac{1}{2-\sqrt{2}} - \frac{\sqrt{2}}{\sqrt{2-\sqrt{2}}}+1}, 2r_i\right\} = 2r_i$. Thus any two disks with centers in one bounded region and radii at least r_i must intersect.

Now consider two disks D_p, D_q, whose centers lie in an unbounded region and $v_p, v_q \in N^2_{SL}(v_i)$. We consider three cases. First, suppose that both v_p and v_q are at distance at most $4r_i$ from v_i. Then the distance between v_p and v_q is at most

$$\max\left\{2r_i\sqrt{(2\cos\frac{\pi}{7}-t)^2+(2\sin\frac{\pi}{7})^2}, 2r_i\sqrt{(2\cos\frac{\pi}{7}-2)^2+(2\sin\frac{\pi}{7})^2}\right\} < 2r_i.$$

Since $r_p, r_q \geq r_i$, we know that $r_p + r_q \geq 2r_i$, and thus D_p and D_q intersect.

Now consider the second case where v_p is at distance at most $4r_i$ from v_i, but the distance between v_q and v_i is greater than $4r_i$ (see Figure 3). Let Q be the intersection point of the line containing points v_i and v_q, and the circle with radius $4r_i$, centered at v_i. Let $D_Q := D(Q, r_q - dist(Q, v_q))$ be the disk with the center Q, contained in and tangent to D_q. Notice that the radius of D_Q is at least r_i (since, by $(\star)$, we obtain $r_q - \text{dist}(Q, v_q) \geq \text{dist}(v_i, v_q) - 3r_i - \text{dist}(Q, v_q) = \text{dist}(v_i, Q) - 3r_i = 4r_i - 3r_i = r_i$). Hence, from the previous case, we know that D_Q and D_p intersect. Since $D_Q \subseteq D_q$, disks D_q and D_p also intersect.

For the last case, suppose that v_p and v_q both at distance greater than $4r_i$ from v_i. As in the previous case we define the disk $D_Q \subseteq D_q$ and analogously $D_P \subseteq D_p$. The radii of both D_Q and D_P are at least r_i, so the disks intersect. Consequently D_q and D_p intersect. ◀

Now we consider the cardinality of $N^2_{LL}(v_i)$.

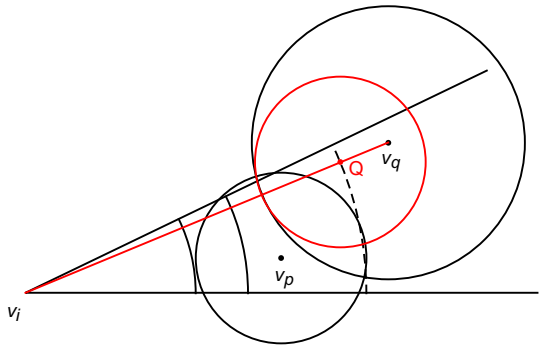


Figure 3 Case 2. Location of the point Q.

▶ **Lemma 3.2.** *$|N^2_{LL}(v_i)| \leq \frac{4}{5}\Delta(G)^2$, for any vertex v_i in a disk graph G.*

Proof. Let v_i be a vertex of graph G. Let H be a graph induced by the neighbors of v_i corresponding to disks with radius at least r_i. We define $\Delta := \Delta(G)$ and $d := |V(H)| \leq \Delta$. Notice that v_i cannot have 6 independent neighbors - recall the argument showing that UDG are $K_{1,6}$-free. Thus the graph H is $\overline{K_6}$-free, and hence $\overline{H}$ is K_6-free. By Theorem 2.1, the maximum number of edges in $\overline{H}$ is $\frac{4}{5}\frac{d^2}{2}$. Therefore the number of edges in H is at least $\binom{d}{2} - \frac{4}{5}\frac{d^2}{2} = \frac{d^2}{10} - \frac{d}{2}$.

The obvious upper bound on the number of all possible vertices in $N^2_{LL}(v_i)$ is $d(\Delta - 1)$. Each edge in H reduces this number by two. Thus we obtain the following upper bound on $|N^2_{LL}(v_i)|$:

$$d(\Delta - 1) - 2\left(\frac{d^2}{10} - \frac{d}{2}\right) = d\Delta - \frac{d^2}{5}.$$

One can easily verify this expression is maximized for $d = \Delta$. Hence $|N^2_{LL}(v_i)| \leq \frac{4}{5}\Delta^2$. ◀

Now Theorem 1.1 is an easy consequence of Lemmas 3.1 and 3.2.

Proof of Theorem 1.1. Consider a greedy algorithm labeling vertices of G, ordered by non-increasing radii of disks in the geometric representation. Let v_i be a vertex of G and let $V' = \{v_1, v_2, \ldots, v_{i-1}\}$ be the set of vertices that are already labeled. We will compute the maximum possible number of labels that cannot be used to label a vertex v_i. Each neighbor of v_i that belongs to V' blocks at most 3 labels, which gives at most $3\Delta(G)$ labels in total. Each vertex in $V' \cap N^2(v_i)$ blocks 1 label. Recall that we can partition the set $V' \cap N^2(v_i)$ into two subsets – $N^2_{SL}(v_i)$ and $N^2_{LL}(v_i)$. By Lemma 3.1, $N^2_{SL}(v_i)$ blocks at most $22\omega(G) \leq 22(\Delta(G) + 1)$ labels in total. By Lemma 3.2, $N^2_{LL}(v_i)$ blocks at most $\frac{4}{5}\Delta(G)^2$ labels in total. Hence the number of labels that cannot be used for v_i is at most $\frac{4}{5}\Delta(G)^2 + 25\Delta(G) + 22$. ◀

4 Conclusion

A very natural question to ask is whether the upper bounds presented in this paper are tight. It is interesting to look for non-trivial constructions of families of (unit) disk graphs with large $L(2,1)$-span, compared to their maximum degree. It is even more interesting, since very little is known about the topic. To the best of our knowledge the largest lower bound is equal to $2\Delta(G)$ and is obtained by a $2k$-th power of a cycle of length $4k+1$, so by the unit disk graph. Clearly this bound is very far from the known upper bounds. It is also very interesting if we can actually force a quadratic span in general disk intersection graphs.

References

1. Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1):3 – 24, 1998.
2. Tiziana Calamonerki. The $L(h, k)$-labelling problem: An updated survey and annotated bibliography. *Comput. J.*, 54(8):1344–1371, 2011.
3. Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.
4. Jirí Fiala, Aleksei V. Fishkin, and Fedor V. Fomin. On distance constrained labeling of disk graphs. *Theor. Comput. Sci.*, 326(1-3):261–292, 2004.
5. Aleksei V. Fishkin. Disk graphs: A short survey. In Roberto Solis-Oba and Klaus Jansen, editors, *Approximation and Online Algorithms: First International Workshop, WAOA 2003, Revised Papers*, pages 260–264, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
6. Daniel Gonçalves. On the $L(p, 1)$-labelling of graphs. *Discrete Mathematics*, 308(8):1405 – 1414, 2008.
7. A. Gräf, M. Stumpf, and Gerhard Weißenfels. On coloring unit disk graphs. *Algorithmica*, 20(3):277–293, 1998.
8. Jerrold R. Griggs and Roger K. Yeh. Labelling graphs with a condition at distance 2. *SIAM J. Discrete Math.*, 5(4):586–595, 1992.
9. William K. Hale. Frequency assignment: Theory and applications. *Proceedings of IEEE*, 68:1497–1514, 1980.
10. Frederic Havet, Bruce Reed, and Jean-Sébastien Sereni. Griggs and Yeh's Conjecture and $L(p, 1)$-labelings. *SIAM Journal on Discrete Mathematics*, 26:145 – 168, 2012.
11. Petr Hliněný and Jan Kratochvíl. Representing graphs by disks and balls (a survey of recognition-complexity results). *Discrete Mathematics*, 229(1):101 – 124, 2001.
12. Konstanty Junosza-Szaniawski, Paweł Rzążewski, Joanna Sokół, and Krzysztof Węsek. Coloring and $L(2, 1)$-labeling of unit disk intersection graphs. In *EuroCG'16 - The European Workshop on Computational Geometry, Book of abstracts*, pages 83–86, 2016.
13. Ross J. Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discrete & Computational Geometry*, 47(3):548–568, 2012.
14. Daniel Král and Riste Škrekovski. A theorem about the channel assignment problem. *SIAM J. Discrete Math.*, 16(3):426–437, 2003.
15. L. Lu. Graph Labeling with Distance Conditions and the Delta Squared Conjecture. Senior Thesis, University of South Carolina.
16. Ewa Malesińska. *Graph theoretical models for frequency assignment problems*. PhD thesis, Technische Universität Berlin, 1997.
17. Ewa Malesińska, Steffen Piskorz, and Gerhard Weißenfels. On the chromatic number of disk graphs. *Networks*, 32(1):13–22, 1998.
18. René Peeters. On coloring j-unit sphere graph. *Technical Report, Department of Economics, Tilburg University*, 1991.
19. Fred S. Roberts. T-colorings of graphs: recent results and open problems. *Discrete Mathematics*, 93(2-3):229–245, 1991.
20. Zhendong Shao, Roger K. Yeh, Kin Keung Poon, and Wai Chee Shiu. The $L(2, 1)$-labeling of $K_{1,n}$-free graphs and its applications. *Appl. Math. Lett.*, 21(11):1188–1193, 2008.
21. Paul Turán. On an extremal problem in graph theory. *Matematikai és Fizikai Lapok*, 48:436–452, 1941.
22. Jan van den Heuvel and Sean McGuinness. Coloring the square of a planar graph. *Journal of Graph Theory*, 42(2):110–124, 2003.
23. Zbigniew Walczak and Jacek Wojciechowski. Transmission scheduling in packet radio networks using graph coloring algorithm. In *Proceedings of the Second International Conference on Wireless and Mobile Communications (ICWMC'06)*, page 46, 2006.

A Polynomial Algorithm for Balanced Clustering via Graph Partitioning*

Luis Evaristo Caraballo†[1], José-Miguel Díaz-Báñez[1], and Nadine Kroher[1]

1 **Department of Applied Mathematics II, University of Seville**
{lcaraballo,dbanez,nkroher}@us.es

Abstract

The objective of clustering is to discover natural groups in datasets and to identify geometrical structures which might reside there, without assuming any prior knowledge on the characteristics of the data.The problem can be seen as detecting the inherent separations between groups of a given point set in a metric space governed by a similarity function. The pairwise similarities between all data objects form a weighted graph adjacency matrix which contains all necessary information for the clustering process, which can consequently be formulated as a graph partitioning problem. In this context, we propose a new cluster quality measure which uses the maximum spanning tree and allows to compute the optimal clustering under the min-max principle in polynomial time. Our algorithm can be applied when a load-balanced clustering is required.

1 Introduction

The objective of clustering is to divide a given dataset into groups of similar objects in an unsupervised manner. Clustering techniques find frequent application in various areas, including computational biology, computer vision, data mining, gene expression analysis, text mining, social network analysis, VLSI design, and web indexing, to name just a few. Commonly, a metric is used to compute pair-wise similarities between all items and the clustering task is formulated as a graph partitioning problem, where a complete graph is generated from the similarity matrix. In fact, many graph-theoretical methods have been developed in the context of detecting and describing inherent cluster structures in arbitrary point sets using a distance function [2].

Here, we propose a novel clustering algorithm based on a quality measure that uses the maximum spanning tree of the underlying weighted graph and addresses a balanced grouping with the min-max principle. More specifically, we aim to detect clusters which are balanced with respect to their ratio of intra-cluster variance to their distance to other data instances. In other words, we allow clusters with weaker inner edges to be formed, if they are located at large distance of other clusters (Figure 1). We prove that an optimal clustering under this measure can be computed in polynomial time using dynamic programming.

Such cluster properties are typically desired when grouping sensors in Wireless Sensor Networks [1] and multi-robot task allocation in Cooperative Robotics where the goal is to allocate tasks to cooperative robots while minimizing costs [5]. Another application area arises from the field of Music Information Retrieval, where several applications rely on the

* This research has received funding from the projects COFLA2 (Junta de Andalucía, P12-TIC-1362) and GALGO (Spanish Ministry of Economy and Competitiveness and MTM2016-76272-R AEI/FEDER,UE) and from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922. The problems studied here were introduced and partially solved during a visit to Havana University, Cuba in October 2016.

† L.E. Caraballo is supported by the Spanish Government under the FPU grant agreement FPU14/04705.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

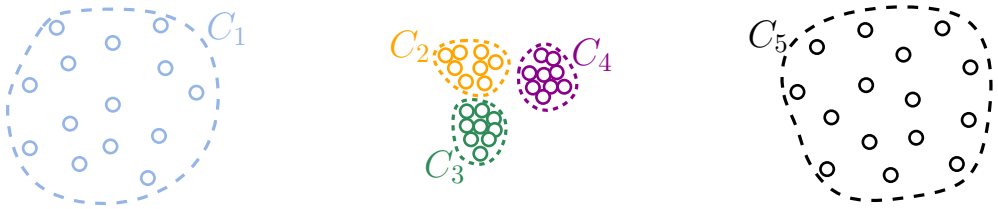


Figure 1 Illustration of the desired cluster properties: The ratios of inner variance to distance to other clusters is balanced among groups. Clusters C_1 and C_5 exhibit a higher variance but are also further apart from the other clusters.

unsupervised discovery of similar (but not identical) melodies or melodic fragments. In this context, clustering methods can be used to explore large music collections with respect to melodic similarity, or to detect repeated melodic patterns within a composition [4].

2 Problem statement

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of points or nodes in a metric space and suppose that there exists a function to estimate the similarity between two nodes. Let A be the matrix holding similarity values computed for every pair of elements in V. The value $A[i, j]$ is the similarity between the nodes v_i and v_j. If $A[i, j] > A[i, l]$ then the node v_i is more similar to v_j than to v_l. Our goal is to create groups such that similar nodes are located in the same cluster and dissimilar nodes are in separate clusters.

Let $G = (V, E, w)$ be a weighted and undirected graph induced by A. In this paper, such graphs are simply referred to as "graph". E is the set of edges and contains an edge for every unordered pair of nodes, and w is a weight function $w : E \longrightarrow (0, 1)$ such that $w(e)$ is the similarity between the nodes connected by e (i.e. if $e = \{v_i, v_j\}$, then $w(e) = A[i, j]$).

Let $C \subseteq V$ be a cluster. The *outgoing edge set* of C, denoted by $Out(C)$, is the set of edges connecting C with $V \setminus C$. Let $MST(C)$ be the maximum spanning tree of C. Let $\max(Out(C))$ and $\min(MST(C))$ be the weights of the heaviest and lightest edges of $Out(C)$ and $MST(C)$, respectively. We define the following function $\Phi(C)$ as the quality measure of a cluster C:

$$\Phi(C) = \begin{cases} 0 & \text{if} \quad C = V, \\ \max(Out(C)) & \text{if} \quad |C| = 1, \\ \dfrac{\max(Out(C))}{\min(MST(C))} & \text{otherwise} \end{cases}$$

Note that higher values of $\Phi(\cdot)$ correspond to worse clusters. Let $\Pi = \{C_1 \ldots, C_k\}$ be a k-clustering (clustering formed by k clusters) of G. To evaluate the quality of Π we use the quality of the worst cluster, $\Phi(\Pi) = \max_{i=1}^{k}\{\Phi(C_i)\}$. Denoting the set of all possible k-clusterings on G by $\mathcal{P}(k, G)$, we state the following optimization problems:

▶ Problem 2.1.

$$\min \quad \Phi(\Pi) \qquad \text{subject to: } \Pi \in \mathcal{P}(k, G).$$

When the value of k is unknown, the problem can be stated as follows:

▶ Problem 2.2.

$$\min \quad \Phi(\Pi) \qquad \text{subject to: } \Pi \in \bigcup_{k=2}^{n} \mathcal{P}(k, G).$$

3 Properties of the optimal clustering

Note that the problems stated above can be generalized to connected (not necessarily complete) graphs, by simply setting $\mathcal{P}(k, G)$ as the set of all the possible partitions of G in k connected components. In this extended abstract most of the proofs are omitted. A full version of the paper can be found in [3].

▶ **Lemma 3.1.** *Let G be a graph and let Π^* be an optimal clustering of G for Problem 2.1 in $\mathcal{P}(k, G)$. Then, $\Phi(\Pi^*) \leq 1$.*

▶ **Lemma 3.2.** *Let G be a graph and let Π^* be an optimal clustering of G for Problem 2.1. Let C be a cluster in Π^*. If $|C| > 1$, then every bipartition of C has a crossing edge in a maximum spanning tree of G.*

▶ **Theorem 3.3.** *Let G be a graph and let $\Pi^* \in \mathcal{P}(k, G)$ be an optimal clustering of G for Problem 2.1. For every cluster $C \in \Pi^*$, the maximum spanning tree of C is a subtree of a maximum spanning tree of G and the heaviest outgoing edge of C is in a maximum spanning tree of G.*

Proof. (Sketch) Let C be a cluster of Π^*. If $|C| = 1$ then, obviously, $MST(C) \subset MST(G)$. If $|C| > 1$ then $MST(C) \subseteq MST(G)$ by Lemma 3.2. The second part of the theorem, claiming that the heaviest edge in $Out(C)$ is in $MST(G)$, is deduced from the properties of the MST. ◀

▶ **Corollary 3.4.** *Let G be a graph and let $\Pi^* \in \bigcup_{k=2}^{n} \mathcal{P}(k, G)$ be an optimal clustering of G for Problem 2.2. For every cluster $C \in \Pi^*$, the maximum spanning tree of C is a subtree of a maximum spanning tree of G and the heaviest outgoing edge of C is in a $MST(G)$.*

We introduce the following notion: Let T be a spanning tree of a graph G. Let $\Pi \in \mathcal{P}(k, T)$ be a clustering of T. The evaluation function $\Phi_T(\Pi)$ operates as usual, but it is restricted to the set of edges forming T. Therefore, the optimal solution for Problem 2.1 on T is $\Pi^\dagger \in \mathcal{P}(k, T)$ such that $\Phi_T(\Pi^\dagger) \leq \Phi_T(\Pi)$ for every other clustering $\Pi \in \mathcal{P}(k, T)$.

▶ **Theorem 3.5.** *Let G be a graph and let T be a maximum spanning tree of G. If $\Pi^* \in \mathcal{P}(k, G)$ and $\Pi^\dagger \in \mathcal{P}(k, T)$ are the optimal clusterings for Problem 2.1 on G and T, respectively; then $\Phi(\Pi^*) = \Phi_T(\Pi^\dagger)$.*

▶ **Corollary 3.6.** *Let G be a graph and let T be a maximum spanning tree of G. If $\Pi^* \in \bigcup_{k=2}^{n} \mathcal{P}(k, G)$ and $\Pi^\dagger \in \bigcup_{k=2}^{n} \mathcal{P}(k, T)$ are the optimal clusterings for Problem 2.2 on G and T, respectively; then $\Phi(\Pi^*) = \Phi_T(\Pi^\dagger)$.*

4 The algorithm

First, recall that Theorem 3.5 and Corollary 3.6 provide a nice property, which allows us to reduce Problems 2.1 and 2.2 from a graph to its maximum spanning tree. Consequently, given a similarity graph, we can operate on its maximum spanning tree $T = (V, E, w)$. From now on, we will use E to denote the set of edges in the maximum spanning tree. Observe that every cluster C in T determines only one subtree of T. Then, using $MST(C)$ to denote the maximum spanning tree in C, may be confusing or redundant. Therefore, instead of using $MST(C)$ we will use $E(C)$ (set of edges connecting nodes in C).

The proposed algorithm is based on *dynamic programming*. We show that the stated problems have an *optimal substructure* which allows us to build the optimal solution in T

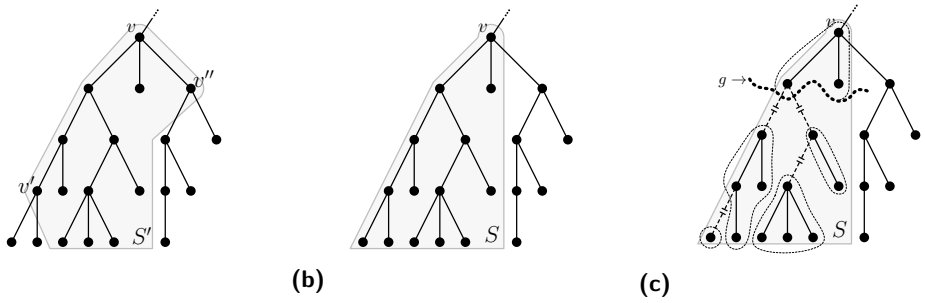


Figure 2 (a) A subtree S' which is not considered. (b) A considered subtree S. (c) Representation of a clustering Π of S. The *head cluster* is above the curve g. The edges of $Out_S(h(\Pi))$ are the edges in S stabbed by g. The clusters of Π that are below g constitute the headless clustering $\Pi \setminus \{h(\Pi)\}$.

from local solutions for subtrees of T. From here on, we consider that the tree T is rooted at $r \in V$. Let $p(v)$ be the parent of v and $c(v)$ be the set of children of v. Given a tree T, let S be a subtree of T and let v be the node with minimum depth in S. Then we say that S is rooted at v. In the rest of this paper, we only consider subtrees S rooted at v that contain all the descendants of vertices $v' \in S \setminus \{v\}$ (see Figures 2a and 2b). We say $S = T_v$ if S contains all the descendants of v.

The main idea of our algorithm is to operate on (local) clusterings of a subtree and perform a bottom-up dynamic programming strategy with two basic operations:

- `UpToParent`: knowing an optimal clustering of a subtree $S = T_v$ such that $T_v \neq T$, compute an optimal clustering of the subtree $\overline{S}$ formed by adding $p(v)$ to S.
- `AddChildTree`: knowing an optimal clustering of a subtree S rooted at $p(v)$, $v \notin S$; and knowing the optimal clustering of $Q = T_v$; compute an optimal clustering of the subtree resultant of joining S and Q.

Now, we elaborate on a (local) clustering Π of a subtree S rooted at v. We call the cluster containing the node v (root of the subtree) the *head cluster* of Π and we denote it $h(\Pi)$ (see Figure 2c). Let $Out_S(C)$ be the set of outgoing edges of C in S. In Figure 2c, $Out_S(h(\Pi))$ is formed by the edges stabbed by the curve g.

Given a clustering Π of a subtree S, let M denote the weight of the heaviest edge in $Out_S(h(\Pi))$, that is, $M = \max(Out_S(h(\Pi)))$. If $h(\Pi)$ contains all the nodes in S then we set $M = 0$. On the other hand, let μ denote the weight of the lightest edge in $E(h(\Pi))$, that is $\mu = \min(E(h(\Pi)))$. If $h(\Pi)$ is formed by single node we set $\mu = 1$. For convenience we introduce the functions $\Phi_S(\cdot)$ and $\Phi_S(\cdot)$ as restricted quality measures of a cluster and a clustering, respectively. They work as defined above, but are restricted to the edges of the subtree S, thus:

$$\Phi_S(h(\Pi)) = \frac{M}{\mu}. \tag{1}$$

Note that, if $S = T$, then $\Phi_S(h(\Pi)) = \Phi(h(\Pi))$. If $S = T_v \neq T$, then: $\Phi(h(\Pi)) = \frac{\max\{M, w(\{v,p(v)\})\}}{\mu}$. For every other cluster $C \in \Pi$, such that C is not the head cluster, the usual evaluation and the restricted one have the same value, $\Phi(C) = \Phi_S(C)$. Consequently, the restricted evaluation of the "*headless*" clustering $\Pi \setminus \{h(\Pi)\}$ is:

$$\Phi_S(\Pi \setminus \{h(\Pi)\}) = \Phi(\Pi \setminus \{h(\Pi)\}) = \max\{\, \Phi(C) \mid C \in \Pi \setminus \{h(\Pi)\} \,\}, \tag{2}$$

and the restricted evaluation of the clustering Π is:

$$\Phi_S(\Pi) = \max\{\Phi_S(h(\Pi)),\ \Phi(\Pi \setminus \{h(\Pi)\})\}. \tag{3}$$

Let S be a subtree of T, and let $\mathcal{H}(l, S, \mu)$ denote the set of l-clustering of S in which μ is the weight of the lightest edge in the head cluster. That is:

$$\mathcal{H}(l, S, \mu) = \{\ \Pi\ \mid\ \Pi \in \mathcal{P}(l, S) \quad \text{and} \quad \mu = \min(E(h(\Pi)))\ \}.$$

Now we are ready to state an encoding of a local solution and the invariant that allows us to apply dynamic programming:

▶ Notation 4.1. Suppose $\mathcal{H}(l, S, \mu)$ is not empty, then a clustering Π in $\mathcal{H}(l, S, \mu)$ is encoded by the ordered pair $O_S(l, \mu) = (M, b)$ if the following properties are fulfilled:

1. $b = \Phi_S(\Pi) = \min\{\ \Phi_S(\Pi')\ \mid\ \Pi' \in \mathcal{H}(l, S, \mu)\}$, and
2. $M = \max(Out_S(h(\Pi))) = \min\left\{\ \max(Out_S(h(\Pi')))\ \middle|\ \begin{array}{l}\Pi' \in \mathcal{H}(l, S, \mu)\ \text{ and}\\ \Phi_S(\Pi') = b\end{array}\ \right\}$.

If $\mathcal{H}(l, S, \mu)$ is empty, then $O_S(l, \mu) = (\infty, \infty)$, where ∞ indicates the "infinity" value.

We set $O_S(l, \mu)$ as (∞, ∞) if $1 < \min\{\ \Phi_S(\Pi)\ \mid\ \Pi \in \mathcal{H}(l, S, \mu)\ \}$. Then, given a subtree S, $O_S(\cdot, \cdot)$ is a function whose domain is $\mathbb{N}_{[1,k]} \times (w(E) \cup \{1\})$ and image $\{(\infty, \infty)\} \cup (w(E) \cup \{0\}) \times \mathbb{R}_{[0,1]}$ where $w(E) = \{\ w(e)\ \mid\ e \in E\ \}$. If $O_S(l, \mu) = (M, b) \neq (\infty, \infty)$, then, by using equations (1), (2) and (3), we obtain that $O_S(l, \mu)$ encodes a clustering Π (not necessarily unique) where:

$$\Phi_S(\Pi) = b = \max\left\{\frac{M}{\mu}, \Phi(\Pi \setminus \{h(\Pi)\})\right\}. \tag{4}$$

For the sake of simplicity, we use the following notation for $O_S(l, \mu) = (M, b)$: $O_S(l, \mu)[1] = M$ and $O_S(l, \mu)[2] = b$. Note that if we have the function O_T then the evaluation of the optimal clusterings for Problems 2.1 and 2.2 are $\min\{\ O_T(k, \mu)[2]\ \mid\ \mu \in w(E) \cup \{1\}\ \}$, and $\min\{\ O_T(k, \mu)[2]\ \mid\ k \in \mathbb{N}_{[2,n]}\ \text{ and }\ \mu \in w(E) \cup \{1\}\ \}$, respectively.

▶ **Lemma 4.2.** *Let S be a subtree rooted at v. Let Π and Π' be two different clusterings of S such that $\min(E(h(\Pi))) = \min(E(h(\Pi'))) = \mu$. If $\Phi_S(\Pi) < \Phi_S(\Pi') \leq 1$ then $\max(Out_S(h(\Pi))) \leq \max(Out_S(h(\Pi')))$.*

▶ **Corollary 4.3.** *Let S be a subtree. For a given value $O_S(l, \mu) = (M, b) \neq (\infty, \infty)$, every l-clustering $\Pi \in \mathcal{H}(l, S, \mu)$ fulfills that: $\Phi_S(\Pi) \geq b$, and $\max(Out_S(h(\Pi))) \geq M$.*

The following lemma is the key of the proposed dynamic programming:

▶ **Lemma 4.4.** *Let S be a subtree rooted at v. Let $O_S(l, \mu) = (M, b) \neq (\infty, \infty)$ and let Π be an l-clustering of S encoded by $O_S(l, \mu)$. Let Q be a subtree of S rooted at $v' \in c(v)$. By removing the edge $e = \{v', v\}$ from S, an l'-clustering $\mathcal{A}$ of Q is induced. Let $\mu' = \min(E(h(\mathcal{A})))$. By replacing $\mathcal{A}$ with a clustering $\mathcal{B}$ encoded by $O_Q(l', \mu')$ and restoring the edge e, a new clustering Π' of S is obtained, which is also encoded as $O_S(l, \mu) = (M, b)$ (Figure 3 depicts the case when e connects nodes in different clusters).*

Using above properties, the recurrence formulas for dynamic programming can be established. Let us mention here one of them, which corresponds to one case in the proof of the Theorem 4.5 (`UpToParent` operation). Let $\omega = w(\{v, p(v)\})$. For $\mu = 1$ we can prove that:

$$O_{\overline{S}}(l, 1) = \left(\omega, \min_{\mu'}\left\{\max\left\{\omega, O_S(l-1, \mu')[2], \frac{\max\{\omega, O_S(l-1, \mu')[1]\}}{\mu'}\right\}\right\}\right).$$

▶ **Theorem 4.5** (UpToParent operation). *Let $S = T_v$ such that $T_v \neq T$, and let $\overline{S}$ be the subtree formed by adding $p(v)$ to S. If we know the function O_S, then the function $O_{\overline{S}}$ can be computed in $O(kn)$.*

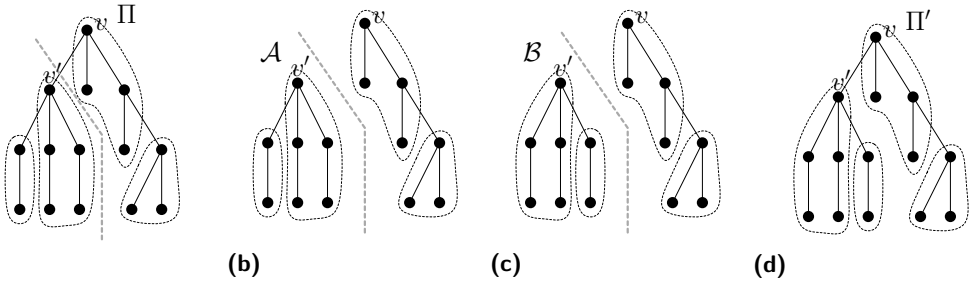


Figure 3 Removing the edge $e = \{v, v'\}$ when e connects nodes in different clusters. (a) Initial situation. (b) Induced clustering $\mathcal{A}$ when e is removed. (c) Replacing $\mathcal{A}$ with another clustering $\mathcal{B}$. (d) Restoring the edge e and obtaining a new clustering Π'.

Finally, for the second operation we have:

▶ **Theorem 4.6** (AddChildTree operation). *Let $Q = T_v$ such that $T_v \neq T$, and let S be a subtree rooted at $p(v)$ such that v is not in S. Let P denote the subtree formed by joining S and Q. If we know the functions O_S and O_Q, then the function O_P can be computed in $O(k^2n^2)$.*

4.1 Complexity of the algorithm

Given a tree T and a value k we can calculate O_T by computing O_{T_v} for every node v in T in a bottom-up (from the leaves to the root) procedure using the mentioned operations. Note that, if v is leaf, then $O_{T_v}(1,1) = (0,0)$ and $O_{T_v}(l,\mu) = (\infty,\infty)$ if $l > 1$ or $\mu < 1$. To compute the function O_{T_v} of an inner node v, we proceed as follows: Let $\{v_1, \ldots, v_m\}$ be the set of children of v. First, considering $S = T_{v_1}$, compute $O_{\overline{S}}$ from O_S using the `UpToParent` operation. Subsequently, we proceed with joining the subtrees T_{v_i} one by one using the `AddChildTree` operation. When all the children have been added, the resultant subtree corresponds to T_v. Note that we apply a single operation per edge. Consequently, this algorithm takes $O(k^2n^3)$ time. Note, that with this algorithm, we obtain the evaluation of the optimal clustering; the clusters of an optimal solution can be computed "*navigating backwards*" through the computed functions.

Problem 2.2 can be solved using the same idea with a slightly more complex approach. We can use a similar algorithm based on functions $O_S(\mu)$, saving the parameter l, (which corresponds to the number of clusters) and then the spent time is $O\left(n^3\right)$ time.

References

1 I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

2 T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the fourth annual symposium on Computational geometry*, 1988.

3 L. E. Caraballo, J. M. Díaz-Báñez, and N. Kroher. *A Polynomial Algorithm for Balanced Clustering via Graph Partitioning.* arXiv:1801.03347, 2018.

4 N. Kroher, J.-M. Díaz-Báñez, and A. Pikrakis. Discovery of repeated melodic phrases in folk singing recordings. *IEEE Transactions on Multimedia*, 2017.

5 A. Ollero and I. Maza. *Multiple heterogeneous unmanned aerial vehicles.* Springer Publishing Company, Incorporated, 2007.

Approximate stabbing queries with sub-logarithmic local replacement

Ivor Hoog v.d.[1] and Maarten Löffler[1]

1 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands**
[i.d.vanderhoog|m.loffler]@uu.nl

Abstract

In this work we present the key ingredients to construct a linear-size data structure that stores a set of spheres or axis-parallel hypercubes in $\mathbb{R}^d$ and supports what we define as 2^m-approximate stabbing queries in logarithmic time and local replacement in sub-logarithmic time, if the regions not overlap "too much". This work uses known techniques such as quadtrees and marked-ancestor trees and introduces a new concept: *key facets* in a d-dimensional quadtree. We show the intuition behind how this new concept can help us perform fast (approximate) stabbing queries with sub-logarithmic local replacement if the dimension d and the approximation variable m are constant. For a detailed description of the query algorithm and for proofs of correctness we refer to our full version.

1 Introduction

An important and well-studied problem in Computational Geometry is the problem where one is given a set $\mathcal{B}$ of n regions in $\mathbb{R}^d$ and needs to find the regions in that set that contain a given query point q. Queries of this form are called *stabbing queries* and in this work we focus on the reporting variant where we have to return the regions that contain q. In a static environment, it is common to make a subdivision of the space based on the regions. Given q, we then quickly find the cell in the subdivision that contains q. Well-known subdivision methods are R-trees, quadtrees and (after applying a duality transformation) k-d trees [4]. Most current research on this topic focuses on the dynamic version of the problem where one wants to maintain a set of regions subject to stabbing queries and adding, removing or translating regions. These dynamic stabbing queries appear as a sub-problem in many day-to-day applications such as GPS tracking, handling data imprecision and data analysis. In certain applications a special kind of update called *local replacement* (Definition **??**) [5] is frequently performed. Intuitively, a local replacement replaces a region by another region similar to it. For example: the ever-increasing uncertainty radius between GPS updates or the moving action radius of an ambulance could both be modeled using local replacement. We assume that the unit itself has a *finger* to its location in the data structure and we want to use the finger to update the data structure and any auxiliary data structures.

In this paper regions in $\mathbb{R}^1$ will be defined as compact intervals. Regions in $\mathbb{R}^d$ will be spheres and axis-parallel hypercubes. A local replacement does not "change too much" in the set of regions and because of this, it is conjectured [5] that it should be possible to perform such a replacement strictly faster than the traditional logarithmic time required for deleting and inserting a region. Löffler *et al.* in [4] present a data structure that supports stabbing queries in logarithmic time and local replacement in sub-logarithmic time for disjoint regions in $\mathbb{R}^1$ and $\mathbb{R}^2$. In [3] Khramtcova and Löffler extend this approach by allowing regions in $\mathbb{R}^1$ to overlap. In this work, we improve the data structure in [3, 4] to work for overlapping regions in $\mathbb{R}^d$ if regions are spheres or axis-aligned hypercubes and if queries are not exact, but what we call 2^m-approximate for constant m. The question of whether exact logarithmic

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

stabbing queries can be performed in logarithmic time with sub-logarithmic replacement remains open.

1.1 Problem definition

In this paper we are given a set $\mathcal{B}$ of n spheres or axis-aligned hypercubes in $\mathbb{R}^d$. We measure the size of any region $B \in \mathcal{B}$ as its diameter. We assume that all regions are contained within an axis-aligned bounding hypercube $\mathcal{K}$. In [4] Löffler *et al.* demanded that the regions in $\mathcal{B}$ are disjoint. In [3] Khramtcova and Löffler relaxed this constraint by introducing limited ply: the **ply** of a set $\mathcal{B}$ of regions in $\mathbb{R}^d$ is the maximum over all points $q \in \mathbb{R}^d$ of the number of $B \in \mathcal{B}$ that contain q. With this restriction from [3] on the set of regions $\mathcal{B}$ we want to perform stabbing queries subject to sub-logarithmic local replacement. Local replacement is defined as replacing a region B_1 with a region B_2 such that the two regions are ρ-similar for constant ρ:

▶ **Definition 1.** Given two regions $B_1, B_2 \in \mathcal{B}$ and a $\rho \geq 1$, we call B_1 and B_2 **ρ-similar** if there exists a region $A \subset \mathbb{R}^d$ with $|A| \leq \rho \min\{|B_1|, |B_2|\}$ such that $B_1, B_2 \subset A$.

▶ **Definition 2.** We call replacing a $B_1 \in \mathcal{B}$ with B_2 a **local replacement** if B_1 and B_2 are ρ-similar for a constant ρ.

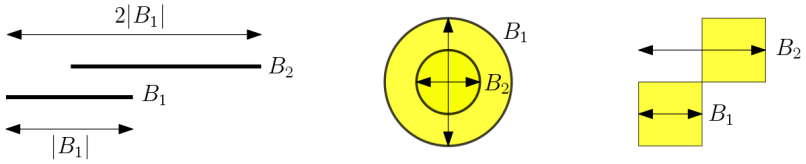


Figure 1 Three examples of a region B_1 and a 2-similar region B_2.

To perform this local replacement we use what we later define as a **level query**. In the full version of this paper we show that level queries cannot be solved in sub-logarithmic time in $\mathbb{R}^d$, so we decided to relax the requirements for stabbing queries and replace exact stabbing queries with approximate queries. Intuitively, we approximate each region B with a smaller inner region. Approximate stabbing queries return all regions B whose inner region $I(B)$ contains q and possibly regions B whose outer region $B \backslash I(B)$ contains q, whilst ply is still defined on the outer region B. The area between the outer and inner region could be seen as a "buffer" that safeguards the inner region. We define our 2^m-approximate stabbing queries using this concept of inner and outer region, the time bounds of our operations depend on the approximation constant m.

▶ **Definition 3.** For any convex region B, we define the **inner region** with respect to m as a map I_m which takes a region and produces its 2^m-approximate inner region. Given a region B, $I_m(B)$ is the scaled down version of B with $|B| = (1 + 2^{-m})|I_m(B)|$ with the center of $I_m(B)$ on the center of B.

▶ **Definition 4.** A 2^m-approximate query on a set of regions $\mathcal{B}$ is a query that given a point $q \in \mathbb{R}^d$ returns all $B \in \mathcal{B}$ for which q is contained in $I_m(B)$ and might return other regions which contain q but does not return regions which not contain q.

Our main result is the following theorem:

▶ **Theorem 5.** *Given a set $\mathcal{B}$ of either spheres or axis-aligned hypercubes in $\mathbb{R}^d$ with ply k and fixed m we can construct a data structure that*

- *takes* $\mathcal{O}(dn)$ *space,*
- *supports regular insertion and deletion of elements and a* 2^m*-approximation of the stabbing queries in* $\mathcal{O}(3^d k \frac{\log(n)}{\log(\log(n))} + \log(n) + m)$ *time,*
- *supports local replacement in* $\mathcal{O}(2^{(d-1)m} k \frac{\log(n)}{\log(\log(n))})$ *time.*

2 Preliminaries

Quadtrees. We always work within an axis-aligned bounding hypercube $\mathcal{K}$ with finite size. A **quadtree** T on $\mathcal{K}$ is a hierarchical partition of $\mathcal{K}$ into smaller axis-aligned **cells**. Given $\mathcal{B}$ we build a dynamic compressed quadtree T that stores $\mathcal{B}$ with the following storing condition: A cell C stores a region B if C is the largest (possible) cell in T such that B contains C *and* C contains the center point of B. In this extended abstract we assume for any set $\mathcal{B}$ we can compute quadtree T storing $\mathcal{B}$ with the following three properties (see [2] for details):

- The tree takes $\mathcal{O}(dn)$ space.
- If for any two cells $C_1, C_2 \in T$ their corresponding hypercubes are ρ-similar for constant ρ, we can walk from C_1 to C_2 in constant time using pointers.
- For each point q we can locate the smallest cell in T that contains q in logarithmic time.

Marked-ancestor trees. Suppose we are given a tree T with nodes (cells in our case) and a fixed simple directed path π over T (this path does not have to follow pre-existing edges in the tree, it is just an arbitrary simple path through the cells in T) where some cell in the path can be marked. In this model we want to support the following query for any cell C: Which is the first marked cell which comes after cell C in the path?. We also want to support updates where cells can be marked or unmarked and inserted into or deleted from the path. This is known as the *marked successor problem*. This problem is solved in [1] with the use of marked-ancestor trees. These trees support the marking and unmarking of cells in the path in $\mathcal{O}(\frac{\log(n)}{\log(\log(n))})$ time. Each marked-ancestor tree with a path π in T allows for what they call the *firstmarked* query:

▶ **Definition 6.** Given a connected path π in T, we can construct a marked-ancestor tree over T such that for each cell $C \in \pi$, **firstmarked(**C**,** π**,** T**)** gives the first marked cell in π starting from C.

The firstmarked query can be solved in $\mathcal{O}(\frac{\log(n)}{\log(\log(n))})$ time [1]. This paper will make extensive use of the firstmarked query. The marked successor problem is a more generic version of the marked-ancestor problem: "Given a cell C in a tree T, which is the first marked node that is an ancestor of C in T?". Observe that the marked-ancestor problem can be solved with a firstmarked query, we call this a **marked-ancestor query**.

3 Intuition and key facets

Let $\mathcal{B}$ be a set of closed and bounded intervals in $\mathbb{R}^1$. The goal of this section is twofold: we introduce a new concept called **key facets** and we use this new concept to introduce a data structure that supports exact stabbing queries in $\mathcal{B}$ in logarithmic time and local updates in sub-logarithmic time. The results in the remainder of this paper are already known: this abstract is an adaption of the data structure and methods used in [3] so that it works with key facets. However, this work contains the basic data structure that we use for 2^m-approximate stabbing queries in $\mathbb{R}^d$ in the full version [2].

Assume that we have a quadtree T that stores $\mathcal{B}$ and that for any query point q we can find the quadtree cell $C \in T$ which contains q in logarithmic time. Then any region B that

contains q either covers C, or intersects C from the left or the right. Let R be the set of all regions that have their center point to the right of C. Let $B_1 \in R$ be the region with the left-most left endpoint of the regions in R. If any region in R contains q then B_1 must also contain q. We can symmetrically construct the set L. Because the ply is limited by k, we only have to find the k left-most/right-most regions in R and L respectively to find all regions that can contain q. As in [3] we find these regions in $\mathcal{O}(k\frac{\log(n)}{\log(\log(n))})$ time using two marked-ancestor trees T_L and T_R and $2k$ marked-ancestor queries.

In [3] the authors obtained the marked-ancestor trees T_L and T_R by looking at which cells contained the left-most and right-most endpoints of the regions B and by marking those cells in T_R and T_L respectively. If we want to extend these results to $\mathbb{R}^d$ we must ask what an equivalent structure would look like in higher dimensions: given q we need to identify a constant number of *directions* and somehow find the k closest regions to q per direction. To do this we introduce **key facets**.

If T is a quadtree in $\mathbb{R}^d$ then its cells are hypercubes in $\mathbb{R}^d$. Each of these cells C have $3^d - 1$ d'-dimensional facets with $d' < d$. Given T we intuitively define the **key facet set** Ξ as the abstract notion of all these facets. For example: in $\mathbb{R}^2$ the set Ξ contains the abstract notion of four vertices and four edges.

▶ **Definition 7.** Let $\mathcal{K}$ be a d-dimensional bounding box and $\mathcal{P}(\mathcal{K})$ be the infinite set of all potential quadtree cells in $\mathcal{K}$. We then define the **key facet set** Ξ as a set of maps (**key facets**) $\chi :: \mathcal{P}(\mathcal{K}) \to \mathbb{R}^d$ where χ projects each cell C to the same d'-dimensional facet of that cell. By **the key facets of a cell** C we mean the set $\Xi(C) := \{\chi(C) \mid \chi \in \Xi\}$.

Let $\chi \in \Xi$ be any key facet and let $C \in T$ be an arbitrary quadtree cell. Let p be the center of C. If $\chi(C)$ is a point then the half-line l through p and $\chi(C)$ is unique. Otherwise let l be a half-line through p and $\chi(C)$ perpendicular to $\chi(C)$. Observe that for any cell $C \in T$, l has the same direction and observe that we can therefore rotate $\mathbb{R}^d$ such that l aligns with the x-axis. This rotation allows us to create a partial order on the cells in T for each $\chi \in \Xi$ by ordering the cells on their lowest x-coordinate after the rotation. Figure 2 shows this rotation where the cells in the partial order are $A <_\chi B =_\chi C <_\chi D <_\chi E$.

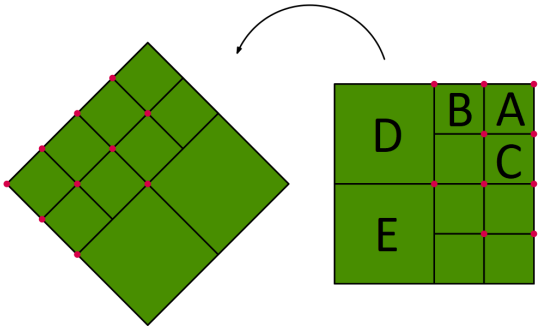


Figure 2 A quadtree in $\mathbb{R}^2$, We can choose χ as the general concept of the "top right vertex". This Figure then contains a rotated quadtree T, and the projection $\chi(T)$ as red dots.

▶ **Definition 8.** Let T be an arbitrary quadtree in $\mathbb{R}^d$ and $\chi \in \Xi$ be a key facet. We then define a **facet path** π_χ as any simple path through T with two properties: the linear order of this path is an extension of the partial ordering given by χ and if $C_1 =_\chi C_2$ but $C_2 \subset C_1$ then C_1 is in the path before C_2.

▶ Observation 1. In $\mathbb{R}^d$ there exists a facet path for each key facet. In $\mathbb{R}^1$ the facet path is unique and it is the pre-order traversal of the quadtree in both directions.

4 The solution in $\mathbb{R}^1$

Marked-ancestor trees together with key facets provide the tools we need to perform approximate stabbing queries in $\mathbb{R}^d$ with sub-logarithmic local replacement. However the algorithm and construction in $\mathbb{R}^d$ is beyond the scope of this paper. We instead introduce the data structure for $\mathbb{R}^d$ and show that with this data structure we can perform exact stabbing queries in $\mathbb{R}^1$ with the same time bounds as in [3] with sub-logarithmic local replacement.

Let the ply of $\mathcal{B}$ be bounded by k. In our data structure we maintain a quadtree T that stores $\mathcal{B}$ as specified in the Preliminaries. Together with T we maintain $k|\Xi| + 1$ marked-ancestor trees over T as follows: for each $\chi \in \Xi$ we maintain k marked-ancestor trees or *levels* denoted as a family of trees $\{T^i_\chi\}_{i\leq k}$. In this family of trees each tree T^i_χ will get the same facet path π_χ. Apart from these $k|\Xi|$ trees we maintain another marked-ancestor tree denoted by T^*. For each region $B \in \mathcal{B}$, we mark all the cells C that B intersects in one of the marked-ancestor trees. If B is stored in C, it marks C in T^*. Else it marks C based on the following condition:

▶ Condition 1. For any family of marked-ancestor trees $\{T^i_\chi\}$ apart from T^*, a cell C is marked by a region B in T^i_χ if:

1. $\chi(C)$ is the highest-dimensional key facet of C that B intersects *and*
2. i is the largest i such that there is a descendant C_d of C which is marked in T^{i-1}_χ and B intersects $\chi(C_d)$.

Stabbing queries in $\mathbb{R}^1$: With this marking condition we can use our marked-ancestor trees to solve any exact stabbing query in logarithmic time. Given a point q, we find the smallest cell C in T that contains q in logarithmic time. We then query each marked-ancestor tree T^i_χ with the *firstmarked* query from C with the path π_χ, this takes $\mathcal{O}(\log(n)+k\frac{\log(n)}{\log(\log(n))})$ time. In the remainder of this section we prove that the returned regions are the only regions that can contain $q \in \mathbb{R}^1$.

▶ **Lemma 9.** *Let C and C_a both be marked in T^i_χ in the same level i by a region B and B_a respectively. Let χ be the map to rightmost point of each cell. If C_a is an ancestor of C then the leftmost point of the region B_a must lie to the right of the leftmost point of the region B. A symmetric property holds if χ is the rightmost point.*

Proof. We prove this by contradiction: assume that the leftmost point of B_a lies to the left of B. Then clearly all C' where B intersects $\chi(C')$ are also intersected by B_a and thus also $\chi(C)$. If $i < k$ then because $\chi(C)$ is intersected by B_a, B_a should have been stored in T^{i+1}_χ and not T^i_χ. If $i = k$ then per definition, B intersects the key facet $\chi(C_d)$ of a descendant C_d of C. C_d is therefore per definition marked in T^i_χ by a region B_d. By our earlier observation, B_a must also intersect B_d in $\chi(C_d)$. We continue this argument all the way down to T^1_χ and see that we violate a ply of k. ◀

▶ **Lemma 10.** *Given a point $q \in \mathbb{R}^1$ contained in a cell C, if C has a lowest-marked ancestor C_1 marked in T^i_χ for any i, χ by an interval B_1 then B_1 is the only region marking an ancestor of C in T^i_χ that can contain q.*

Proof. Let T^i_χ be an arbitrary marked-ancestor tree in $\mathbb{R}^1$. Then χ is either the left-most or right-most vertex of a cell. Assume that we have found the lowest marked ancestor of the cell C in the marked-ancestor tree T^i_χ, the cell C_1 marked by a region B_1. Then B_1 either contains the query point q or does not. If B_1 does not contain q then Lemma 9 demands that any ancestor of C_1 marked in T^i_χ is marked by a region that reaches less far than B_1

does and so any other region marking an ancestor of C_1 cannot reach q. If B_1 does contain q then any region marking a higher ancestor of C in T_χ^i that reaches q must also intersect $\chi(C_1)$ and should therefore have marked the ancestor in T_χ^{i+1} or violate the ply of k. ◀

The result of this lemma is that in $\mathbb{R}^1$ with constant ply we can solve stabbing queries in logarithmic time with a marked-ancestor query in each marked-ancestor tree (the lowest-marked ancestor in T^* always contains q).

The level query in $\mathbb{R}^1$: Assume we want to replace a region B_1 with a region B_2 such that B_1 and B_2 are ρ-similar and that we have a *finger* to the cell C_1 that stores B_1. Because B_1 and B_2 are ρ-similar we can use at most $\mathcal{O}(\rho)$ pointers to reach the cell C_2 that should store B_2 (See preliminaries). What remains is to update the marked-ancestor trees in sub-logarithmic time. For that we define the **level query**.

▶ **Definition 11.** A **level query** checks for a given region B, cell C that B intersects and a family of marked-ancestor trees $\{T_\chi^i\}$ in which level i the region B marks C (if any).

In $\mathbb{R}^1$ the level query can be solved for each $\{T_\chi^i\}$ by just incrementally performing at most k firstmarked queries with the unique facet path π_χ. One can show that the result of that query gives the unique region that could "force" B to mark C in a higher level.

The solution in $\mathbb{R}^d$. We show in the full version [2] that in $\mathbb{R}^d$ the abstract level query has a lower bound of logarithmic time. This version also contains a more elaborate description of the data structure required to store and approximately query regions in $\mathbb{R}^d$. Specifically we introduce an extension of the marking Condition 1 and show how to perform 2^m-approximate stabbing queries and local replacements with the query times as specified in Theorem 5.

In this version and the full version we have demonstrated how the concept of key facets can give information about the closest regions that lie in a certain *direction* from a query point q. Given the cell that contains the query point, we can even provide this information in sub-logarithmic time. A future research direction could be to see if we can use key facets and marked-ancestor queries for sub-logarithmic visibility queries.

Acknowledgements. The authors would like to thank Elena Khramtcova for inspiring discussion of the problem. I.H. and M.L. were partially supported by the Netherlands Organisation for Scientific Research (NWO) through project no 614.001.504.

References

1 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 534–543. IEEE, 1998.

2 Ivor Hoog v.d. and Maarten Löffler. Approximate stabbing queries with sub-logarithmic local replacement (full version). *arXiv to appear.*

3 Elena Khramtcova and Maarten Löffler. Dynamic stabbing queries with sub-logarithmic local updates for overlapping intervals: Proc. 12th international computer science symposium in russia. *Computer Science–Theory and Applications*, 10304, 2017.

4 Maarten Löffler, Joseph A. Simons, and Darren Strash. Dynamic planar point location with sub-logarithmic local updates. In *13th Int. Symp. Algorithms and Data Structures (WADS)*, pages 499–511. Springer Berlin Heidelberg, 2013. full version: arXiv preprint arXiv:1204.4714.

5 Yakov Nekrich. Data structures with local update operations. *Algorithm Theory–SWAT 2008*, pages 138–147, 2008.

Subquadratic Encodings for Point Configurations*

Jean Cardinal[†1], Timothy M. Chan[‡2], John Iacono[§1], Stefan Langerman[¶1], and Aurélien Ooms[‖1]

1 **Département d'Informatique, Université libre de Bruxelles (ULB), Belgium**
2 **Department of Computer Science, University of Illinois at Urbana-Champaign, USA**

Abstract

For many algorithms dealing with sets of points in the plane, the only relevant information carried by the input is the combinatorial configuration of the points: the orientation of each triple of points in the set (clockwise, counterclockwise, or collinear). This information is called the *order type* of the point set. In the dual, realizable order types and abstract order types are combinatorial analogues of line arrangements and pseudoline arrangements. Too often in the literature we analyze algorithms in the real-RAM model for simplicity, putting aside the fact that computers as we know them cannot handle arbitrary real numbers without some sort of encoding. Encoding an order type by the integer coordinates of some realizing point set is known to yield doubly exponential coordinates in some cases. Other known encodings can achieve quadratic space or fast orientation queries, but not both. In this contribution, we give a compact encoding for abstract order types that allows efficient query of the orientation of any triple: the encoding uses $O(n^2)$ bits and an orientation query takes $O(\log n)$ time in the word-RAM model. This encoding is space-optimal for abstract order types. We show how to shorten the encoding to $O(n^2(\log\log n)^2/\log n)$ bits for realizable order types, giving the first subquadratic encoding for those order types with fast orientation queries. We further refine our encoding to attain $O(\log n/\log\log n)$ query time at the expense of a negligibly larger space requirement. In the realizable case, we show that all those encodings can be computed efficiently. Finally, we generalize our results to the encoding of point configurations in higher dimension.

1 Introduction

At SoCG'86, Chazelle asked [16]:

> "How many bits does it take to know an order type?"

This question is of importance in Computational Geometry for the following two reasons: First, in many algorithms dealing with sets of points in the plane, the only relevant information carried by the input is the combinatorial configuration of the points given by the orientation of each triple of points in the set (clockwise, counterclockwise, or collinear) [7]. Second, computers as we know them can only handle numbers with finite description and we cannot

* Details, proofs, and improvements in the arXiv version [5].

† `jcardin@ulb.ac.be`. Supported by the "Action de Recherche Concertée" (ARC) COPHYMA, convention number 4.110.H.000023.

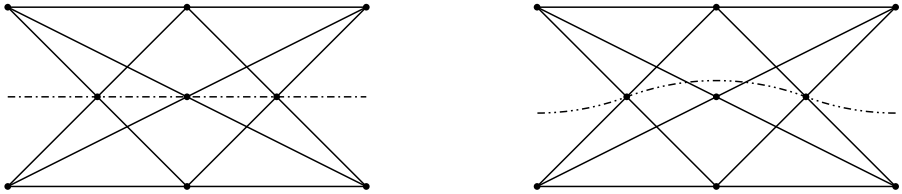
‡ `tmc@illinois.edu`.

§ `eurocg18@johniacono.com`. Supported by NSF grants CCF-1319648, CCF-1533564, CCF-0430849 and MRI-1229185, a Fulbright Fellowship and by the Fonds de la Recherche Scientifique-FNRS under Grant n° MISU F 6001 1 and two Missions Scientifiques.

¶ `slanger@ulb.ac.be`. Directeur de recherches du Fonds de la Recherche Scientifique-FNRS.

‖ `aureooms@ulb.ac.be`. Supported by the Fund for Research Training in Industry and Agriculture (FRIA).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.



(a) Realizable order type. **(b)** Abstract order type which is not realizable.

Figure 1 Pappus's configuration.

assume that they can handle arbitrary real numbers without some sort of encoding. The study of *robust* algorithms is focused on ensuring the correct solution of problems on finite precision machines. Chapter 41 of The Handbook of Discrete and Computational Geometry is dedicated to this issue [23].

The (counterclockwise) orientation $\nabla(p, q, r)$ of a triple of points p, q, and r with coordinates (x_p, y_p), (x_q, y_q), and (x_r, y_r) is the sign of the determinant

$$\begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix}.$$

Given a set of n labeled points $P = \{ p_1, p_2, \ldots, p_n \}$, we define the *order type* of P to be the function $\chi\colon [n]^3 \to \{-, 0, +\}\colon (a, b, c) \mapsto \nabla(p_a, p_b, p_c)$ that maps each triple of point labels to the orientation of the corresponding points, up to isomorphism. The order type of a point set has been further abstracted into combinatorial objects known as (rank-three) *oriented matroids* [10]. The *chirotope axioms* define consistent systems of signs of triples [3]. From the topological representation theorem [4], all such *abstract* order types correspond to pseudoline arrangements, while, from the standard projective duality, order types of point sets correspond to straight line arrangements. See Chapter 6 of The Handbook for more details [21].

When the order type of a pseudoline arrangement can be realized by an arrangement of straight lines, we call the pseudoline arrangement *stretchable*. As an example of a nonstretchable arrangement, Levi gives Pappus's configuration where eight triples of concurrent straight lines force a ninth, whereas the ninth triple cannot be enforced by pseudolines [19] (see Figure 1). Ringel shows how to convert the so-called "non-Pappus" arrangement of Figure 1 (b) to a simple arrangement while preserving nonstretchability [22]. All arrangements of eight or fewer pseudolines are stretchable [13], and the only nonstretchable simple arrangement of nine pseudolines is the one given by Ringel [20]. More information on pseudoline arrangements is available in Chapter 5 of The Handbook [11].

Figure 1 shows that not all pseudoline arrangements are stretchable. Indeed, most are not: there are $2^{\Theta(n^2)}$ abstract order types [8] and only $2^{\Theta(n \log n)}$ realizable order types [1, 15].

Information theory implies that we need quadratic space for abstract order types whereas we only need linearithmic space for realizable order types. Hence, storing all $\binom{n}{3}$ orientations in a lookup table seems wasteful. Another obvious idea for storing the order type of a point set is to store the coordinates of the points, and answer orientation queries by computing the corresponding determinant. While this should work in many practical settings, it cannot work for all point sets. Perles's configuration shows that some configuration of points, containing collinear triples, forces at least one coordinate to be irrational [18]. Order types of points in general position can always be represented by rational coordinates. It is well known, however,

that some configurations require doubly exponential coordinates, hence coordinates with exponential bitsizes if represented in the normal way [17].

Goodman and Pollack defined λ-matrices which can encode abstract order types using $O(n^2 \log n)$ bits [14]. They asked if the space requirements could be moved closer to the information-theoretic lower bounds. Felsner and Valtr showed how to encode abstract order types optimally in $O(n^2)$ bits via the wiring diagram of their corresponding allowable sequence [8, 9] (as defined in [12]). Aloupis et al. gave an encoding of size $O(n^2)$ that can be computed in $O(n^2)$ time and that can be used to test for the isomorphism of two distinct point sets in the same amount of time [2]. However, it is not known how to decode the orientation of one triple from any of those encodings in, say, sublinear time. Moreover, since the information-theoretic lower bound for realizable order types is only $\Omega(n \log n)$, we must ask if this space bound is approachable for those order types while keeping orientation queries reasonably efficient.

Our Results

In this contribution, we are interested in *compact* encodings for order types: we wish to design data structures using as few bits as possible that can be used to quickly answer orientation queries of a given abstract or realizable order type.

▶ **Definition 1.1.** For fixed k and given a function $f : [n]^k \to [O(1)]$, we define a $(S(n), Q(n))$-encoding of f to be a string of $S(n)$ bits such that, given this string and any $t \in [n]^k$, we can compute $f(t)$ in $Q(n)$ query time in the word-RAM model.

We give the first optimal encoding for abstract order types that allows efficient query of the orientation of any triple: the encoding is a data structure that uses $O(n^2)$ bits of space with queries taking $O(\log n)$ time in the word-RAM model.

▶ **Theorem 1.2.** *All abstract order types have an $(O(n^2), O(\log n))$-encoding.*

Our encoding is far from being space-optimal for realizable order types. We show that its construction can be easily tuned to only require $O(n^2(\log\log n)^2/\log n)$ bits in this case.

▶ **Theorem 1.3.** *All realizable order types have an $(O(\frac{n^2(\log\log n)^2}{\log n}), O(\log n))$-encoding.*

We further refine our encoding to reduce the query time to $O(\log n/\log\log n)$.

▶ **Theorem 1.4.** *All abstract order types have an $(O(n^2), O(\frac{\log n}{\log\log n}))$-encoding.*

▶ **Theorem 1.5.** *All realizable order types have a $(O(\frac{n^2 \log^\varepsilon n}{\log n}), O(\frac{\log n}{\log\log n}))$-encoding.*

In the realizable case, we give quadratic upper bounds on the preprocessing time required to compute an encoding in the real-RAM model.

▶ **Theorem 1.6.** *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^2$ we can compute the encoding of their order type as in Theorems 1.4 and 1.5 in $O(n^2)$ time.*

We generalize our encodings for chirotopes of point sets in higher dimension.

▶ **Theorem 1.7.** *All realizable chirotopes of rank $k \geq 4$ have a $(O(\frac{n^{k-1}(\log\log n)^2}{\log n}), O(\frac{\log n}{\log\log n}))$-encoding.*

▶ **Theorem 1.8.** *In the real-RAM model and the constant-degree algebraic decision tree model, given n real-coordinate input points in $\mathbb{R}^d$ we can compute the encoding of their chirotope as in Theorem 1.7 in $O(n^d)$ time.*

Our data structure is the first subquadratic encoding for realizable order types that allows efficient query of the orientation of any triple. It is not known whether a subquadratic constant-degree algebraic decision tree exists for the related problem of deciding whether a point set contains a collinear triple. Any such decision tree would yield another subquadratic encoding for realizable order types. We see the design of compact encodings for realizable order types as a subgoal towards subquadratic nonuniform algorithms for this related problem, a major open problem in Computational Geometry. Note that pushing the preprocessing time below quadratic would yield such an algorithm.

2 Encoding Order Types via Hierarchical Cuttings

We assume that we can access some arrangement of lines or pseudolines that realizes the order type we want to encode. We thus exclusively focus on the problem of encoding the order type of a given arrangement. This does not pose a threat against the existence of an encoding. In this extended abstract, we sketch the general idea for a simple subquadratic encoding. For full details, proofs, and improvements, we refer to the arXiv version [5].

Hierarchical Cuttings

We encode the order type of an arrangement via hierarchical cuttings as defined in [6]. A cutting in $\mathbb{R}^d$ is a set of (possibly unbounded and/or non-full dimensional) constant-complexity cells that together partition $\mathbb{R}^d$. A $\frac{1}{r}$-cutting of a set of n hyperplanes is a cutting with the constraint that each of its cells is intersected by at most $\frac{n}{r}$ hyperplanes. There exist various ways of constructing $\frac{1}{r}$-cuttings of size $O(r^d)$. In the plane, hierarchical cuttings can be constructed for arrangement of pseudolines with the same properties.

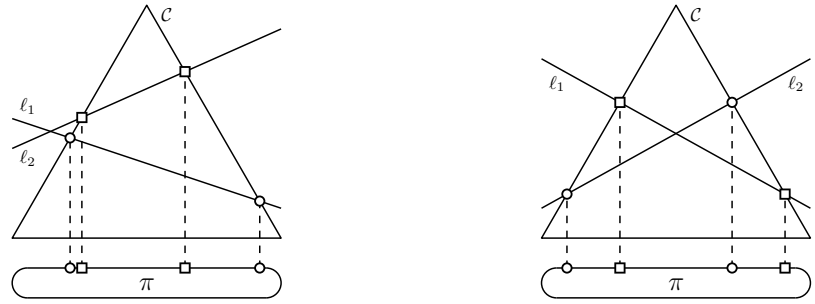
Idea

We want to preprocess n pseudolines $\{\ell_1, \ell_2, \ldots, \ell_n\}$ in the plane so that, given three indices a, b, and c, we can compute their orientation, that is, whether the intersection $\ell_a \cap \ell_b$ lies above, below or on ℓ_c. Our data structure builds on cuttings as follows: Given a cutting Ξ and the three indices, we can locate the intersection of ℓ_a and ℓ_b inside Ξ. The location of this intersection is a cell of Ξ. The next step is to decide whether ℓ_c lies above, lies below, contains or intersects that cell. In the first three cases, we are done. Otherwise, we can answer the query by recursing on the subset of pseudolines intersecting the cell containing the intersection. We build on hierarchical cuttings to solve all subproblems efficiently.

Intersection Location

When the ℓ_a are straight lines, locating the intersection $\ell_a \cap \ell_b$ in Ξ is trivial if we know the real parameters of ℓ_a and ℓ_b and of the descriptions of the subcells of Ξ. However, in our model we are not allowed to store real numbers. To circumvent this annoyance, and to handle arrangements of pseudolines, we make a simple observation illustrated by Figure 2.

▶ Observation 1. Two pseudolines ℓ_a and ℓ_b intersect in the interior of a full-dimensional cell $\mathcal{C}$ if and only if each pseudoline properly intersects the boundary of $\mathcal{C}$ exactly twice and their intersections with its boundary alternate.



(a) $\ell_a \cap \ell_b \cap \mathcal{C} = \emptyset$. **(b)** $\ell_a \cap \ell_b \cap \mathcal{C} \neq \emptyset$.

Figure 2 Cyclic permutations (π).

This gives us a way to encode the location of the intersection of ℓ_a and ℓ_b in Ξ using only bits. We define the *cyclic permutation* of a full-dimensional cell $\mathcal{C}$ and a finite set of pseudolines $\mathcal{L}$ to be the finite sequence of properly intersecting pseudolines from $\mathcal{L}$ encountered when walking along the boundary of $\mathcal{C}$ in clockwise or counterclockwise order, up to rotation and reversal. Location in a non-full-dimensional cell can be encoded similarily.

Encoding

Given n pseudolines in the plane and some fixed parameter r, compute a hierarchical $\frac{1}{r}$-cutting of those pseudolines. This hierarchical cutting consists of ℓ levels labeled $0, 1, \ldots, \ell-1$. Level i has $O(r^{2i})$ cells. Each of those cells is further partitioned into $O(r^2)$ subcells. The $O(r^{2(i+1)})$ subcells of level i are the cells of level $i+1$. Each cell of level i is intersected by at most $\frac{n}{r^i}$ pseudolines, and hence each subcell is intersected by at most $\frac{n}{r^{i+1}}$ pseudolines.

We compute and store a combinatorial representation of the hierarchical cutting as follows: For each level of the hierarchy, for each cell in that level, for each pseudoline intersecting that cell, for each subcell of that cell, we store two bits to indicate the location of the pseudoline with respect to that subcell. When a pseudoline intersects the interior of a 2-dimensional subcell, we also store the two indices of the intersections of that pseudoline with the subcell in the cyclic permutation associated with that subcell, beginning at an arbitrary location in, say, clockwise order. Location in a non-full-dimensional subcell can be encoded similarily.

The hierarchy is such that each subcell of the last level is intersected by no more than $t = \frac{n}{r^\ell}$ pseudolines. For those subcells, we answer the queries by table lookup. The use of hierarchical cuttings essentially guarantees we get quadratic preprocessing time, quadratic space, and logarithmic query time in the abstract case. In the realizable case, we know there can only be $2^{O(t \log t)}$ distinct lookup tables. Choosing the right superconstant t leads to subquadratic space in that case.

References

1 Noga Alon. The number of polytopes configurations and real matroids. *Mathematika*, 33(1):62–71, 1986.
2 Greg Aloupis, John Iacono, Stefan Langerman, Özgür Özkan, and Stefanie Wuhrer. The complexity of order type isomorphism. In *SODA*, pages 405–415. SIAM, 2014.
3 Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Günter M Ziegler. Oriented matroids. In *Encyclopedia of Mathematics*, volume 46. Cambridge University Press, 1993.

4 Jürgen Bokowski, Susanne Mock, and Ileana Streinu. On the Folkman-Lawrence topological representation theorem for oriented matroids of rank 3. *European Journal of Combinatorics*, 22(5):601–615, 2001.

5 Jean Cardinal, Timothy M. Chan, John Iacono, Stefan Langerman, and Aurélien Ooms. Subquadratic encodings for point configurations. *ArXiv e-prints*, 2018. arXiv:1801.01767 [cs.CG].

6 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158.

7 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10. Springer Science & Business Media, 2012.

8 Stefan Felsner. On the number of arrangements of pseudolines. In *SOCG*, pages 30–37. ACM, 1996.

9 Stefan Felsner and Pavel Valtr. Coding and counting arrangements of pseudolines. *Discrete & Computational Geometry*, 46(3):405–416, 2011.

10 Jon Folkman and Jim Lawrence. Oriented matroids. *Journal of Combinatorial Theory, Series B*, 25(2):199–236, 1978.

11 Jacob E. Goodman. Pseudoline arrangements. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 97–128. Chapman and Hall/CRC.

12 Jacob E. Goodman. Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics*, 32(1):27–35, 1980.

13 Jacob E. Goodman and Richard Pollack. Proof of Grünbaum's conjecture on the stretchability of certain arrangements of pseudolines. *Journal of Combinatorial Theory, Series A*, 29(3):385–390, 1980.

14 Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM Journal on Computing*, 12(3):484–507, 1983.

15 Jacob E. Goodman and Richard Pollack. Upper bounds for configurations and polytopes in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 1:219–227, 1986.

16 Jacob E. Goodman and Richard Pollack. Allowable sequences and order types in discrete and computational geometry. In *New Trends in Discrete and Computational Geometry*, pages 103–134. Springer, 1993.

17 Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. Coordinate representation of order types requires exponential storage. In *STOC*, pages 405–410. ACM, 1989.

18 Branko Grünbaum. *Convex Polytopes*. Springer, 2005.

19 Friedrich Levi. Die teilung der projektiven ebene durch gerade oder pseudogerade. *Ber. Math.-Phys. Kl. Sächs. Akad. Wiss*, 78:256–267, 1926.

20 Jürgen Richter. Kombinatorische realisierbarkeitskriterien für orientierte matroide. *Mitt. Math. Sem. Univ. Giessen*, 194:1–112.

21 Jürgen Richter-Gebert and Günter M. Ziegler. Oriented matroids. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 129–151. Chapman and Hall/CRC, 2004.

22 Gerhard Ringel. Teilungen der ebene durch geraden oder topologische geraden. *Mathematische Zeitschrift*, 64(1):79–102, 1956.

23 Chee K. Yap. Robust geometric computation. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 927–952. Chapman and Hall/CRC.

QPTAS and Subexponential Algorithm for Maximum Clique on Disk Graphs *

Édouard Bonnet[1], Panos Giannopoulos[1], Eun Jung Kim[2], Paweł Rzążewski[3], and Florian Sikora[2]

1 **Department of Computer Science, Middlesex University, London**
edouard.bonnet@dauphine.fr, p.giannopoulos@mdx.ac.uk
2 **Université Paris-Dauphine, PSL Research University, CNRS UMR, LAMSADE, Paris, France**
{eun-jung.kim,florian.sikora}@dauphine.fr
3 **Faculty of Mathematics and Information Science, Warsaw University of Technology** p.rzazewski@mini.pw.edu.pl

Abstract

A disk graph is the intersection graph of closed disks in the plane. We show the structural result that a disjoint union of cycles is the complement of a disk graph if and only if at most one of those cycles is of odd length. From that, we derive the first QPTAS and subexponential algorithm running in time $2^{\tilde{O}(n^{2/3})}$ for Maximum Clique on disk graphs. In contrast, the problem is unlikely to have such algorithms on intersection graphs of filled ellipses or filled triangles.

1 Introduction

Intersection graphs for many different families of geometric objects have been widely studied due to their practical applications and rich structural properties [13]. Among the most studied ones are (unit) *disk graphs*, which are intersection graphs of closed (unit) disks in the plane with applications ranging from sensor networks to map labeling.

Clark et al. [8] gave a polynomial-time algorithm for Maximum Clique on unit disk graphs. The complexity of the problem on general disk graphs is unfortunately still unknown. Ambühl and Wagner [1] gave a simple 2-approximation algorithm for Maximum Clique on general disk graphs, showed the problem to be APX-hard on intersection graphs of ellipses and gave a $9\rho^2$-approximation algorithm for filled ellipses of aspect ratio at most ρ.

Results. We show that the disjoint union of two odd cycles is not the complement of a disk graph, providing an infinite family of forbidden induced subgraphs, an analogue to the work of Atminas and Zamaraev on unit disk graphs [2]. Using this property we give a QPTAS and a subexponential-time algorithm for Maximum Clique on disk graphs. Finally, we show that for filled ellipses or filled triangles, there is a constant $\alpha > 1$ for which an α-approximation algorithm running in subexponential time is highly unlikely.

Definitions and notations. For two integers $i \leqslant j$, let $[i, j]$ be the set $\{i, i+1, \ldots, j-1, j\}$ and $[i]$ the set $[1, i]$. For a subset S of vertices of a graph, let $N(S)$ be the open neighborhood of S and $N[S]$ the set $N(S) \cup S$. The *2-subdivision* of a graph G is the graph H obtained by subdividing each edge of G twice. The *co-2-subdivision* of G is the complement of H. The *co-degree* of G is the maximum degree of its complement. A *co-disk* is the complement of a disk graph. For two distinct points x and y in the plane, we denote by $\ell(x, y)$ the unique line going through x and y, and by $\text{seg}(x, y)$ the closed straight-line segment whose endpoints are x and y. For a segment s with positive length, let $\ell(s)$ be the unique line containing s.

* Partially supported by EPSRC (EP/N029143/1) and ANR (ANR-17-CE40-0028).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2 Disk graphs with co-degree 2

We fully characterize the degree-2 complements of disk graphs by showing the following.

▶ **Theorem 1.** *A disjoint union of cycles is the complement of a disk graph if and only if the number of odd cycles is at most one.*

We only show the first part of this theorem, i.e., the union of two disjoint odd cycles is not the complement of a disk graph. As disk graphs are closed under taking induced subgraphs, two vertex-disjoint odd cycles in the complement of a disk graph have to be linked by at least one edge. The second part, i.e., how to represent the complement of the disjoint union of even cycles and one odd cycle, is deferred to the full version of this abstract [4].

2.1 The disjoint union of two odd cycles is not co-disk

A *proper representation* is a disk representation where every edge is witnessed by a proper intersection of the two corresponding disks, i.e., the interiors of the two disks intersect. It is easy to transform a disk representation into a proper one (of the same graph) where, additionally, no three disk centers are aligned. With this assumption, we can show that in a representation of a $K_{2,2}$ with the four centers in convex position, both non-edges have to be *diagonal*.

▶ **Lemma 2.** *In a disk representation of $K_{2,2}$ with the four centers in convex position, the non-edges are between vertices corresponding to opposite centers in the quadrangle.*

A useful consequence of the previous lemma is the following.

▶ **Corollary 3.** *In any disk representation of $K_{2,2}$ with centers c_1, c_2, c_3, c_4 with the two non-edges between the vertices corresponding to c_1 and c_2, and between c_3 and c_4, it should be that $\ell(c_1, c_2)$ intersects $seg(c_3, c_4)$ or $\ell(c_3, c_4)$ intersects $seg(c_1, c_2)$.*

We can now prove the main result of this section thanks to the previous corollary, parity arguments, and some elementary properties of closed plane curves.

▶ **Theorem 4.** *The complement of the disjoint union of two odd cycles is not a disk graph.*

Proof. Let s and t be two positive integers and $G = \overline{C_{2s+1} + C_{2t+1}}$ the complement of the disjoint union of two cycles of lengths $2s+1$ and $2t+1$. Assume that G is a disk graph. Let $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) be the cycle embedded in the plane formed by $2s+1$ (resp. $2t+1$) straight-line segments joining the consecutive centers of disks along the first (resp. second) cycle. We number the segments of $\mathcal{C}_1$ from S_1 to S_{2s+1}, and the segments of $\mathcal{C}_2$, from S'_1 to S'_{2t+1}.

For the i-th segment S_i of $\mathcal{C}_1$, let a_i be the number of segments of $\mathcal{C}_2$ intersected by the line $\ell(S_i)$ prolonging S_i, let b_i be the number of segments S'_j of $\mathcal{C}_2$ such that $\ell(S'_j)$ intersects S_i, and let c_i be the number of segments of $\mathcal{C}_2$ intersecting S_i. For the second cycle, we define similarly a'_j, b'_j, c'_j. The quantity $a_i + b_i - c_i$ counts the number of segments of $\mathcal{C}_2$ which can possibly represent a $K_{2,2}$ with S_i according to Corollary 3. Since G is a disk graph, $a_i + b_i - c_i = 2t + 1$ for every $i \in [2s + 1]$. Otherwise there would be at least one segment S'_j of $\mathcal{C}_2$ such that $\ell(S_i)$ does not intersect S'_j *and* $\ell(S'_j)$ does not intersect S_i.

Observe that a_i is an even integer since $\mathcal{C}_2$ is a closed curve. Also, $\Sigma_{i=1}^{2s+1} a_i + b_i - c_i = (2t + 1)(2s + 1)$ is an odd number, as the product of two odd numbers. This implies that $\Sigma_{i=1}^{2s+1} b_i - c_i$ shall be odd. $\Sigma_{i=1}^{2s+1} c_i$ counts the number of intersections of the two closed curves $\mathcal{C}_1$ and $\mathcal{C}_2$, and is therefore even. Hence, $\Sigma_{i=1}^{2s+1} b_i$ shall be odd. Observe that $\Sigma_{i=1}^{2s+1} b_i = \Sigma_{j=1}^{2t+1} a'_j$ by reordering and reinterpreting the sum from the point of view of the segments of $\mathcal{C}_2$. Since the a'_j are all even, $\Sigma_{i=1}^{2s+1} b_i$ is also even; a contradiction. ◀

3 Algorithmic consequences

As a clique in a graph G is an independent set in $\overline{G}$, we focus on Maximum Independent Set on graphs without two vertex-disjoint odd cycles with no edges connecting them.

3.1 QPTAS

The odd cycle packing number $\mathrm{ocp}(H)$ of a graph H is the maximum number of vertex-disjoint odd cycles in H. The condition that $\overline{G}$ does not contain two vertex-disjoint odd cycles with no edges between them is not the same as saying that $\mathrm{ocp}(\overline{G}) = 1$. Otherwise, we could directly use the PTAS on graphs H with n vertices and $\mathrm{ocp}(H) = o(n/\log n)$ by Bock et al. [3], which does not need the odd cycle packing as an input. This is important, since finding a maximum odd cycle packing is NP-hard [11]. Using Theorem 4, we can show the following lemma, which spares us having to determine the odd cycle packing number.

▶ **Lemma 5.** *Let H be a graph with n vertices, whose complement is a disk graph. If $\mathrm{ocp}(H) > n/\log^2 n$, then H has a vertex of degree at least $n/\log^4 n$.*

If $\overline{G}$ has no vertex of degree at least $n/\log^4 n$, by Lemma 5, we know that $\mathrm{ocp}(\overline{G}) \leqslant n/\log^2 n = o(n/\log n)$ and run the PTAS of Bock et al. If $\overline{G}$ has a vertex v of degree at least $n/\log^4 n$ (it may still hold that $\mathrm{ocp}(\overline{G}) = o(n/\log n)$), we branch on v: either we include v in our solution and remove $N[v]$, or we discard v. The complexity of this is given by $F(n) \leqslant F(n-1) + F(n - n/\log^4 n)$, which solves to the running time below.

▶ **Theorem 6.** *For any $\varepsilon > 0$, Maximum Clique can be $(1+\varepsilon)$-approximated in time $2^{O(\log^5 n)}$, when the input is a disk graph with n vertices.*

3.2 Subexponential algorithm

An *odd cycle cover* is a subset of vertices whose deletion makes the graph bipartite. Györi et al. [9] showed that graphs with small odd girth have small odd cycle cover. This can be seen as relativizing the fact that odd cycles do not have the Erdős-Pósa property. Bock et al. [3] turned the non-constructive proof into a polynomial-time algorithm.

▶ **Theorem 7** ([9] and [3])**.** *Let H be a n-vertex graph with no odd cycle shorter than δn. H has a polynomial-time computable odd cycle cover of size at most $(48/\delta)\ln(5/\delta)$.*

We start by showing three variants of an algorithm.

▶ **Theorem 8.** *Let G be a disk graph with n vertices and Δ, c the maximum degree and odd girth of $\overline{G}$. Maximum Clique has a branching or can be solved, up to a polynomial factor, in time: (i) $2^{\tilde{O}(n/\Delta)}$ (branching), (ii) $2^{\tilde{O}(n/c)}$ (solved), (iii) $2^{O(c\Delta)}$ (solved).*

Proof. We look for a maximum independent set in $\overline{G}$. For (i), let v be a vertex of degree Δ in $\overline{G}$. We branch on v: either we include v in our solution and remove $N[v]$, or discard v. The complexity is given by $F(n) \leqslant F(n-1) + F(n-(\Delta+1))$, which solves to (i). This does not give a $2^{\tilde{O}(n/\Delta)}$-time algorithm as the maximum degree might drop. We do the branching as long as it is *good enough* and finish with the algorithms corresponding to (ii) and (iii).

For (ii) and (iii), let C be the cycle of length c, it can be found in polynomial time. By Theorem 7, with $\delta = c/n$, we find an odd cycle cover X in $\overline{G}$ of size $\tilde{O}(n/c)$ in polynomial time. We exhaustively guess in time $2^{\tilde{O}(n/c)}$ the intersection I of an optimum solution with X and finish by finding, in polynomial time, a maximum independent set in the bipartite graph $\overline{G} - (X \cup N(I))$. The total complexity of this case is $2^{\tilde{O}(n/c)}$, which shows (ii).

For (iii), $\overline{G} - N[C]$ is bipartite as $\overline{G}$ contains no two vertex-disjoint odd cycles with no edges between them. As every vertex in $\overline{G}$ has degree at most Δ, it holds that $|N[C]| \leqslant c(\Delta-1) \leqslant c\Delta$. Indeed, a vertex of C can only have $c(\Delta-2)$ neighbors outside C. We guess the intersection of the optimal solution with $N[C]$ and find the maximum independent set in a bipartite graph (a subgraph of $\overline{G} - N[C]$), which can be done in total time $2^{O(c\Delta)}$. ◀

The structure of G affects the bounds in Theorem 8 as follows.

▶ **Corollary 9.** *Let G be a disk graph with n vertices.* Maximum Clique *can be solved in time: (a) $2^{\tilde{O}(n^{2/3})}$, (b) $2^{\tilde{O}(\sqrt{n})}$ if the maximum degree of $\overline{G}$ is constant, (c) polynomial, if both the maximum degree and the odd girth of $\overline{G}$ are constant.*

On general graphs, (b) is the hardest case for Maximum Clique. Moreover, the win-win strategy of Corollary 9 can be directly applied to solve Maximum Weighted Clique.

4 Intersection graphs of filled ellipses and filled triangles

For Maximum Clique on intersection graphs of (*non-filled*) ellipses and triangles, APX-hardness was shown by Ambühl and Wagner [1]. Their reduction also implies that there is no subexponential algorithm for this problem, unless the ETH fails [10]. They claim that their hardness result extends to filled ellipses since *"intersection graphs of ellipses without interior are also intersection graphs of filled ellipses"*. Unfortunately, this claim is incorrect.

▶ **Theorem 10.** *There is a graph which has an intersection representation with ellipses without their interior, but has no intersection representation with convex sets.*

Figure 1 shows a counterexample and the argument is similar to the one used by Brimkov et al. [5], which was in turn inspired by the construction by Kratochvíl and Matoušek [12].

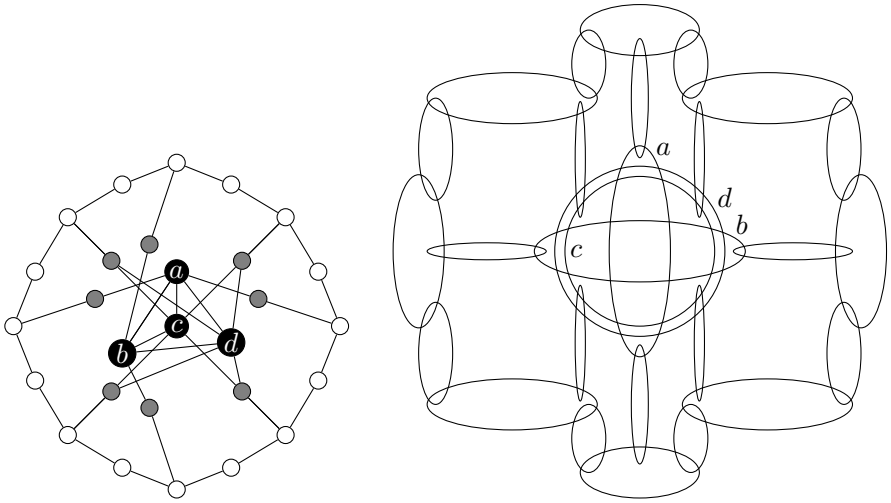


Figure 1 A graph (left), which has a representation with empty ellipses (right) but no representation with convex sets.

Fortunately, we can show that the hardness result *does* hold for filled ellipses (and filled triangles) with a different reduction. Our construction can be seen as streamlining the ideas of Ambühl and Wagner [1] and, we believe, is simpler.

▶ **Theorem 11.** *There is a constant $\alpha > 1$, such that for every $\varepsilon > 0$,* Maximum Clique *on intersection graphs of filled ellipses or filled triangles has no α-approximation algorithm running in subexponential time $2^{n^{1-\varepsilon}}$, unless the ETH fails. For ellipses, this holds even when they have arbitrarily small eccentricity and arbitrarily close value of major axis.*

This contrasts with the subexponential algorithm and QPTAS for eccentricity 0 (disks). It also subsumes [6] (where NP-hardness is shown for connected shapes contained in a disk of radius 1 and containing a concentric disk of radius $1 - \varepsilon$ for arbitrarily small $\varepsilon > 0$).

We first show the lower bound for Maximum Weighted Independent Set on the class of all 2-subdivisions, and, hence, the same for Maximum Weighted Clique on all co-2-subdivisions. Then we show that intersection graphs of filled ellipses or of filled triangles contain all co-2-subdivisions.

The following inapproximability result for Maximum Independent Set on bounded-degree graphs was shown by Chlebík and Chlebíková [7]. As their reduction is almost linear, the PCP of Moshkovitz and Raz [14] boosts this hardness result from ruling out polynomial-time up to ruling out subexponential time $2^{n^{1-\varepsilon}}$ for any $\varepsilon > 0$.

▶ **Theorem 12** ([7, 14])**.** *There is a constant $\beta > 0$ such that* Maximum Independent Set *on graphs with n vertices and maximum degree Δ cannot be $1+\beta$-approximated in time $2^{n^{1-\varepsilon}}$ for any $\varepsilon > 0$, unless the ETH fails.*

▶ **Theorem 13.** *There is a constant $\alpha > 1$ such that for any $\varepsilon > 0$,* Maximum Independent Set *on the class of all the 2-subdivisions has no α-approximation algorithm running in subexponential time $2^{n^{1-\varepsilon}}$, unless the ETH fails.*

Proof. Let G be a graph with maximum degree Δ, n vertices $v_1, \ldots, v_n$ and m edges $e_1, \ldots, e_m$. Let H be its 2-subdivision. The $2m$ vertices in $V(H) \setminus V(G)$, representing edges, are called *edge vertices* and denoted by $v^+(e_1), v^-(e_1), \ldots, v^+(e_m), v^-(e_m)$, as opposed to the other *original vertices* of H. If $e_k = v_i v_j$ is an edge of G, then $v^+(e_k)$ (resp. $v^-(e_k)$) has two neighbors: $v^-(e_k)$ and v_i (resp. $v^+(e_k)$ and v_j).

There is a maximum independent set S which contains exactly one of $v^+(e_k), v^-(e_k)$ for every $k \in [m]$. S cannot contain both $v^+(e_k)$ and $v^-(e_k)$ as they are adjacent. If S contains neither $v^+(e_k)$ nor $v^-(e_k)$, then adding $v^+(e_k)$ to S and potentially removing the other neighbor of $v^+(e_k)$, can only increase the size of the independent set. Hence S contains m edge vertices and $s \leqslant n$ original vertices, and there is no larger independent set in H.

Assume an approximation with ratio $\alpha := 1 + \frac{2\beta}{(\Delta+1)^2}$ for Maximum Independent Set on 2-subdivisions running in subexponential time, where ratio $1+\beta > 1$ is not attainable for Maximum Independent Set on graphs of maximum degree Δ (Theorem 12). On instance H, this algorithm would output a solution with m' edge vertices and s' original vertices. This solution can be easily (in polynomial time) transformed into an at-least-as-good solution with m edge vertices and s'' original vertices forming an independent set in G. We assume that $s'' \geqslant n/(\Delta+1)$ since for any independent set of G, we can obtain an independent set of H consisting of the same set of original vertices and m edge vertices. Since $m \leqslant n\Delta/2$ and $s'' \geqslant n/(\Delta+1)$, we obtain $m \leqslant s''\Delta(\Delta+1)/2$ and $2m/(\Delta+1)^2 \leqslant s''\Delta/(\Delta+1)$. From $\frac{m+s}{m+s''} \leqslant \alpha$ and $\Delta \geqslant 3$, we easily get that $s \leqslant s''(1+\beta)$, contradicting the Theorem 12. ◀

Finally, we can prove the following lemma; see Figure 2 for an example with filled triangles. This, together with Theorem 13, proves Theorem 11.

▶ **Lemma 14.** *The class of intersection graphs of filled triangles or filled ellipses contains all co-2-subdivisions.*

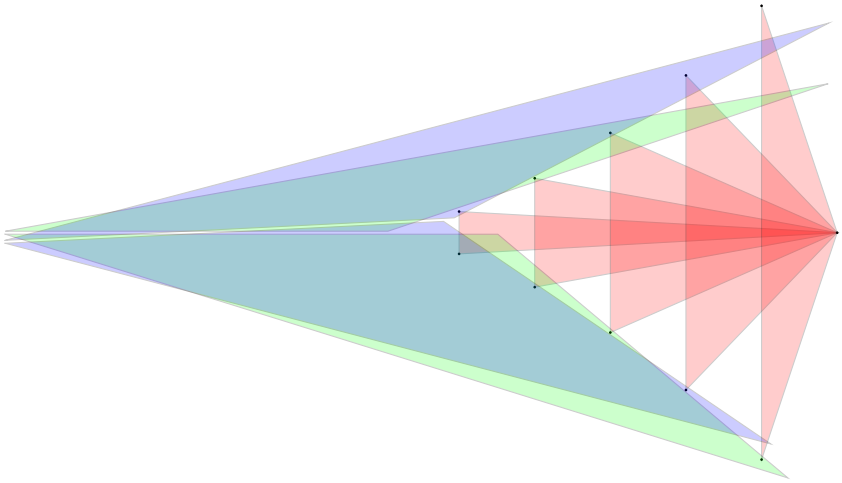


Figure 2 A co-2-subdivision of a graph with 5 vertices (in red) represented with triangles. Two edges are represented: between vertices 1 and 4 (in green) and between vertices 2 and 3 (in blue).

References

1 C. Ambühl and U. Wagner. The Clique Problem in Intersection Graphs of Ellipses and Triangles. *Theory Comput. Syst.*, 38(3):279–292, 2005.

2 A. Atminas and V. Zamaraev. On forbidden induced subgraphs for unit disk graphs. *arXiv preprint arXiv:1602.08148*, 2016.

3 A. Bock, Y. Faenza, C. Moldenhauer, and A. J. Ruiz-Vargas. Solving the Stable Set Problem in Terms of the Odd Cycle Packing Number. In *34th FSTTCS*, pages 187–198, 2014.

4 É. Bonnet, P. Giannopoulos, E. J. Kim, P. Rzążewski, and F. Sikora. QPTAS and Subexponential Algorithm for Maximum Clique on Disk Graphs. *arXiv preprint arXiv:1712.05010*, 2017.

5 V. E. Brimkov, K. Junosza-Szaniawski, S. Kafer, J. Kratochvíl, M. Pergel, P. Rzążewski, M. Szczepankiewicz, and J. Terhaar. Homothetic polygons and beyond: Intersection graphs, recognition, and maximum clique. *CoRR*, abs/1411.2928, 2014.

6 S. Ceroi. The clique number of unit quasi-disk graphs. TR RR-4419, INRIA, 2002.

7 M. Chlebík and J. Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theor. Comput. Sci.*, 354(3):320–338, 2006.

8 B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

9 E. Györi, A. V. Kostochka, and T. Łuczak. Graphs without short odd cycles are nearly bipartite. *Discrete Mathematics.*, 163(1):279 – 284, 1997.

10 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, Dec. 2001.

11 K. Kawarabayashi and B. A. Reed. Odd cycle packing. In *42nd STOC*, pages 695–704, 2010.

12 J. Kratochvíl and J. Matoušek. Intersection graphs of segments. *J. Comb. Theory, Ser. B*, 62(2):289–315, 1994.

13 T. A. McKee and F. R. McMorris. *Topics in intersection graph theory.* SIAM, 1999.

14 D. Moshkovitz and R. Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5):29:1–29:29, 2010.

Automatic Drawing for Tokyo Metro Map

Masahiro Onda[1], Masaki Moriguchi[2], and Keiko Imai[3]

1 **Graduate School of Science and Engineering, Chuo University**
monda@imai-lab.ise.chuo-u.ac.jp

2 **Meiji Institute for Advanced Study of Mathematical Sciences, Meiji University**
moriguchi@meiji.ac.jp

3 **Faculty of Science and Engineering, Chuo University**
imai@ise.chuo-u.ac.jp

Abstract

The Tokyo subway is one of the most complex networks in the world, and it is difficult to obtain its easy-to-read metro map using existing layout methods. In this paper, we present a new method that can generate complex metro maps like the Tokyo metro map. Our method consists of two phases. The first phase generates rough metro maps. It decomposes the metro networks into smaller subgraphs, and generates rough metro maps partially. In the second phase, we use a local search technique to improve the aesthetic quality of the rough metro maps. The experimental results including the Tokyo metro map are shown.

1 Introduction

Metro maps are useful tools for passengers in complex metro networks. They help us in planning the route from one station to another promptly. Metro maps give us information, such as where to change stations, which direction to go for the destination, and how many stations that are from the departure point to the goal. Passengers access the topology of the network before the exact geographical location.

Such a metro map was first created by Henry Beck [1], who was an English technical draftsman. He proposed the criteria for drawing maps that are easy to read. It is still difficult to draw them manually, even if a draftsman takes sufficient time. Therefore, the problem of drawing metro map layouts automatically (in general, we call it the *metro map layout problem*) has been investigated for a long time.

Hong *et al.* [3] presented five methods for the metro map layout problem. These methods are based on the spring-embedder paradigm, where several attracting and repelling forces act between the vertices. The forces iteratively optimize the metro map until a locally optimal equilibrium. They experimented with real-world data, and were able to produce metro maps quickly. The Tokyo metro map can be obtained by using their method. However, there are crossings of edges and labels.

Stott *et al.* [4] described a method of drawing metro maps using multi-criteria optimization based on hill climbing. They defined various metrics, which are used in a weighted sum to find a fitness value for a layout of metro maps. Their approach uses some clustering mechanisms to avoid local minima.

Nöllenburg and Wolff [2] considered the layout problem of railroad lines connecting stations and the labeling problem for the stations simultaneously using mixed-integer programming (MIP). The solvability of the problem depends on the size of the MIP. If their method can solve the problem in terms of the size of the MIP, the results obtained by their method almost satisfy Beck's criteria and they are easy to read.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2 The Metro Map Layout Problem

This section describes the metro map layout problem based on the formulation defined by Nöllenburg and Wolff [2]. A metro network consists of metro lines and stations, and it can be seen as a planar graph. If two metro lines intersect at the stations, by adding such intersections as vertices, the network can be considered as a planar graph $G = (V, E)$. Let L be a set of metro lines. Each edge of G belongs to at least one metro line. An embedded graph of G in the plane is called a *metro map* Γ. Let $l_e > 0$ be the minimum length of $\Gamma(e)$ for each edge $e \in E$, and $d_{\min} > 0$ be the minimum distance between each pair of non-incident edges in Γ. Nöllenburg and Wolff gave seven design rules for drawing metro maps, and split these rules into hard and soft constraints. The following formulation of the metro map layout problem has been defined by Nöllenburg and Wolff [2].

- (H1) For each edge e, $\Gamma(e)$ must be octilinear.
- (H2) For each vertex v, the circular order of its neighbors must agree in Γ and the input embedding.
- (H3) For each edge e, $\Gamma(e)$ must have a length at least l_e.
- (H4) For each edge e, $\Gamma(e)$ must have a distance of at least $d_{\min} > 0$ from each non-incident edge in Γ.
- (S1) The lines in L should have few bends in Γ, and the bend angle($< 180°$) should be as large as possible.
- (S2) For each pair of adjacent vertices (u, v), their relative position should be preserved in Γ.
- (S3) The total edge length of Γ should be small.

The objective function is the sum obtained by multiplying soft constraints by each weight. Thus the problem can be defined in [2] as follows:

Metro-Map Layout Problem

Input: a plane graph $G = (V, E)$ with maximum degree eight, metro lines L of G, minimum edge lengths $l_e > 0$, for each $e \in E$, and a minimum distance $d_{\min} > 0$.

Output: a metro map Γ that satisfies the hard constraints and optimizes the soft constraints.

The method can be applied to metro networks having station name labels as follows. They considered the label region of some stations as a quadrilateral. They added edges expressing the boundaries of the label regions into the metro network. They formulated a MIP model for this modified graph. Further, they presented some heuristics that reduce the size of the graph and the MIP model because the size of the MIP is generally large. The approach that reduces the size of the graph replaces each path of continued vertices with degree two temporarily by a path of length 3. The approach does not consider overlapping-free constraints in the initial MIP formulation. Then, during the optimization process, they add the constraints on demand, and repeat until the constraints are satisfied.

3 The Metro Map Drawing Method

If the solution can be obtained by the MIP formulation defined by Nöllenburg and Wolff [2], the results are good metro maps. However, the metro map in Tokyo is too complex and, in our experiment, their method did not succeed to generate the Tokyo metro map; our

implementation of their method with CPLEX ran out of memory and terminated after running for five days without finding any feasible solution. The implementation used the size reduction techniques (simplification of degree-2 vertices and callback-based resolution of edge intersections) and ran on the system described in Section 4. Therefore, in this paper, we try to modify their method and optimize partially rather than overall. Our method has two phases: a brief incremental construction phase and an iterative improvement phase. The first incremental construction phase draws rough metro maps by MIP based on the methods in [2]. The second iterative improvement phase improves the maps generated by the incremental construction phase to make them more aesthetic. We forgo the optimum solution for the soft constraints. Instead, our method improves the metro maps that satisfy the hard constraints by a local search technique, and gives a local optimum solution.

3.1 Preprocessing

We preprocess the Tokyo metro network by adding the label regions and replace every long path with a path with three edges using an approach similar to Nöllenburg and Wolff's method. The Tokyo metro network has stations whose degree is greater than eight. However, such stations cannot use the MIP formulation in [2]. Therefore, we modify the graph such that for every vertex, its degree is less than nine. A vertex v_{in}, whose degree is greater than two, is called an *interchange vertex* (See Figure 1 (a)). We split each interchange vertex into three vertices and connect these three vertices by horizontal line segments to reduce its degree (See Figure 1 (b)). Subsequently, the vertices that are initially adjacent to the interchange vertex v_{in} reconnect to one of the three new vertices by the direction of the original edge. We set the label of the interchange vertex using line segments connecting the three vertices as the interchange label region. For the rest of this paper, the graph obtained after splitting interchange vertices is described as $\widetilde{G}$

Figure 1 (a) An interchange vertex v_{in}. (b) An interchange vertex after adding two vertices and reconnecting.

3.2 Reduced Optimization

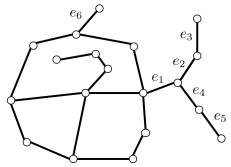
We generate metro maps by reduced optimization using MIP formulation [2]. In reduced optimization, we formulate as some edges can move and the others cannot move but these length can be adjustable. By doing so, it is easy to move edges while satisfying the constraint of overlapping-free (H4), and to generate good metro maps.

3.3 Incremental Construction

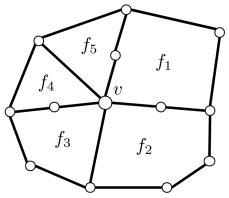
The incremental construction method presented in this section generates metro maps that satisfy the hard constraints.

Division method

It is difficult to generate a metro map from a large input graph even if we use Nöllenburg and Wolff's method. Therefore, our method decomposes the input graph into smaller subgraphs, external trees, and faces. We define a *external edge* as an edge incident with only non-boundary face, and an *external tree* T as a tree consisting of connected external edges. The trees consisting of the edges $\{e_1, e_2, e_3, e_4, e_5\}$ and $\{e_6\}$ in Figure 2(a) are external trees. For each face f of G, there is a subgraph that forms a boundary of f. We call it the *facial subgraph* of f. Next, for a vertex v incident with an inner face, we define its *facial neighborhood* $N(v)$ as the union of the facial subgraphs of all the *inner faces* incident with v. Note that the union of graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2), ..., G_k = (V_k, E_k)$ is defined as the graph whose vertex set is $\cup_{i=1}^{k} V_i$ and whose edge set is $\cup_{i=1}^{k} E_i$. See Figure 2(b). We sort $\{N(v)\}$ in terms of the degree of the vertex v in G in descending order, and this sorted list is described as $\mathcal{N} = \{N(v_1), N(v_2), ..., N(v_m)\}$.



(a) External trees are $\{e_1, e_2, e_3, e_4, e_5\}$ and $\{e_6\}$.



(b) A facial neighborhood of v, and facial subgraphs of f_1, f_2, f_3, f_4 and f_5.

Figure 2 Division method in the incremental construction method.

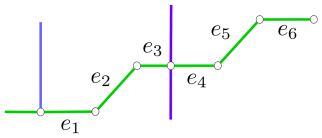


Figure 3 An interchange path that is defined as a set of edges that is from a non-degree 2 vertex to another non-degree 2 vertex.

The Incremental Construction Algorithm

Suppose that all external trees and the sorted list $\mathcal{N} = \{N(v_1), N(v_2), ..., N(v_m)\}$ are known. For this input data, the incremental construction algorithm treats subgraphs in order of the facial neighborhoods and external trees. When we generate a rough metro map by reduced optimization, we use the MIP formulation and heuristics from [2]. To reduce the size of the input graph, all the paths between two interchange vertices and external trees $\{T_1, T_2, ..., T_l\}$ in the graph $\widetilde{G}$ are simplified, and they consist of at most three edges. Note that the paths in $N(v_i)$ are also simplified.

First, we treat the facial neighborhoods in order of the sorted list. Then, we treat the external trees one at a time.

Step 1

Set G_0 to an empty set.

For each $i = 1, ..., m$:

Add $N(v_i) = (V_{N(v_i)}, E_{N(v_i)})$ to G_{i-1} and the obtained graph is called $G_i = (V_i, E_i)$.

That means $V_i = V_{i-1} \cup V_{N(v_i)}$ and $E_i = E_{i-1} \cup E_{N(v_i)}$.

However, if all vertices and edges in $N(v_i)$ have already been added to G_{i-1}, $N(v_i)$ is not added to G_{i-1}, and G_{i-1} becomes G_i.

Then, we solve the MIP for G_i by fixing the direction of the edge in Γ_{i-1}, and obtain the new map Γ.

Step 2

For each $j = 1, ..., l$:

Add T_j to G_{m+j-1} and obtain a graph G_{m+j}.

Solve the MIP for G_{m+j} by fixing the direction of the edge in Γ_{m+j-1}, and obtain the new map Γ_{m+j}.

After step 2, the map Γ satisfies all the hard constraints and this can be used as the input of the next iterative improvement method. By generating metro maps from facial neighborhood of the vertex with large degree, it is possible to generate metro maps satisfying the constraint of cyclic ordering (H2).

3.4 Iterative Improvement

An iterative improvement method improves the metro map generated by the incremental construction algorithm to obtain a local optimal solution. We decompose a metro map into paths. In Γ, an *interchange path* is defined as a set of edges that are from a non-degree 2 vertex to another non-degree 2 vertex. For example, the edges e_1, e_2, and e_3, or the edges e_4, e_5, and e_6 in Figure 3.

In the iterative improvement method, for each interchange path we generate metro maps by reduced optimization. Thus, an iterative improvement method generates optimal metro maps so that only each interchange path can move direction.

Because the original objective function is linear with respect to the variables representing edge lengths, it does not penalize non-uniform edge lengths. Therefore, the results might be poorly balanced. To prevent this, we add the following soft constraint:

(S4) For each degree 2 vertex v whose adjacent vertices u, w do not have degree 2, the difference between the lengths of $\Gamma(uv)$ and $\Gamma(vw)$ should be as small as possible.

Accordingly, we add the following cost function to the objective function

$$\sum_{v\in V,\deg(v)=2,\deg(u)\neq 2,\deg(w)\neq 2} |\lambda(uv) - \lambda(vw)|,$$

where $\deg(v)$ is the degree of vertex v and $\lambda(uv)$ is the length of edge uv. This cost function, proposed in [4], is linear and explicitly penalizes non-uniform edge lengths.

4 Experimental Results

We applied our methods to the Tokyo metro network that has 434 vertices and 531 edges. We solved the MIP with the optimizer CPLEX 12.7.1 running on a computer with a 6-core Intel Xeon 3.60 GHz processor, and 128 GB RAM.

The Tokyo metro map drawn using our methods is shown in Figure 4. The weights used in the objective function were as following: in the incremental construction method, (S1) is 1, (S2) is 30, and (S3) is 1, and in the iterative improvement method, (S1) is 1, (S2) is 20, (S3) is 5, and (S4) is 1. To speed up the computation, we experimentally increased the weight of relative position (S2). Conversely, increasing the weight of the edge length (S3) tends to increase the calculation time. The layout was mostly generated within 5 hours: the incremental construction method took 13 minutes, and the iterative improvement method took 5 hours.

5 Conclusion

We have proposed an incremental construction method and an iterative improvement method that improves metro maps generated by the incremental construction method. The former

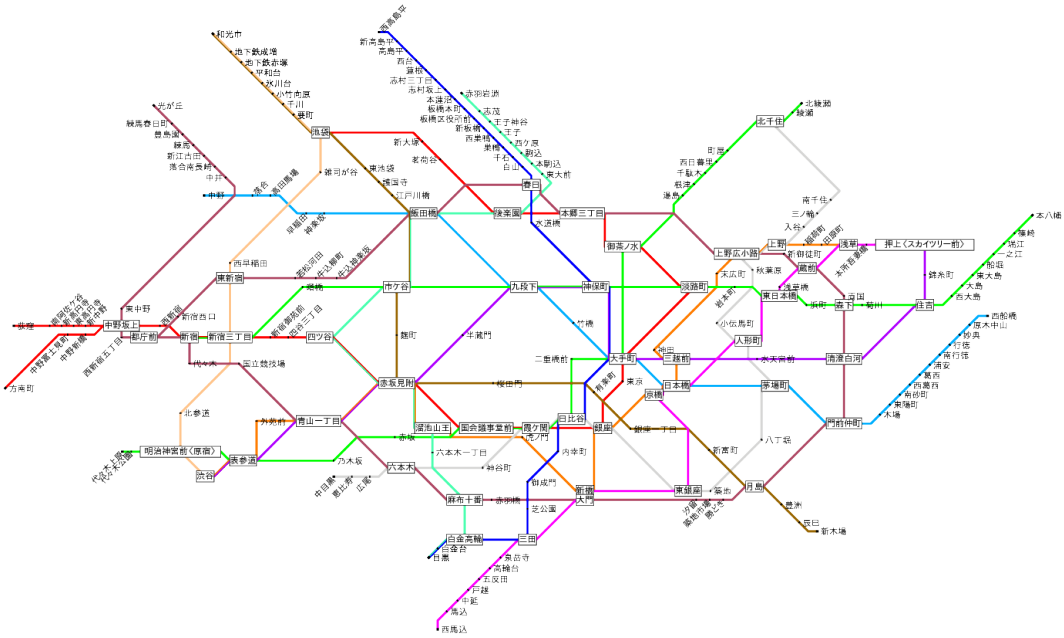


Figure 4 Layout of Tokyo railway network produced by our approach.

can generate complex metro maps that satisfy hard constraints quickly, because we decompose the input graph into subgraphs and generate a rough metro map partially. The latter improves metro maps and arranges vertices in a balanced position. Our methods can generate complex metro maps such as the Tokyo metro map that cannot be directly produced by the approach of Nöllenburg and Wolff [2].

One limitation of the iterative improvement method is that it is slow. We plan to reveal the cause for why phase 2 takes so much longer than phase 1. Another limitation is that there are some artifacts around terminal stations and dummy crossings in the results. We also plan to improve it using a post-processing.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable and constructive comments. This work was supported by JSPS KAKENHI Grant Numbers 16K00024, 16H01728, and 17K14580.

References

1 K. Garland. *Mr Beck's Underground Map.* Capital Transport, 1994.

2 M. Nöllenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2011.

3 S.-H. Hong, D. Merrick, and H. A. D. do Nascimento. Automatic visualization of metro maps. *J. Visual Languages and Computing*, 17(3):203–224, 2006.

4 J. Stott, P. Rodgers, J. C. Martinez-Ovando, and S. G. Walker. Automatic metro map layout using multicriteria optimization. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):101–114, 2011.

On the Weak Line Cover Numbers

Oksana Firman[1], Alexander Ravsky[2], and Alexander Wolff[1]

1 **Universität Würzburg, Germany**
`firstname.lastname@uni-wuerzburg.de`
2 **Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, National Academy of Science of Ukraine, Lviv, Ukraine**
`alexander.ravsky@uni-wuerzburg.de`

Abstract

For a given graph, we want to find crossing-free straight-line drawings of low visual complexity. A measure for the visual complexity of a drawing that has been considered before is the minimum number of lines needed to cover all vertices. In 3D, this number, the *3D weak line cover number*, is denoted by $\pi_3^1(G)$ for a given graph G. In 2D, for any planar graph G, the *2D weak line cover number* is denoted by $\pi_2^1(G)$.

We inductively construct an infinite family of polyhedral graphs with maximum degree 6, treewidth 3, and unbounded π_2^1-value. We also determine the π_2^1- and π_3^1-values of the Platonic graphs. We pose a number of open questions about the 2D and 3D weak line cover numbers.

1 Introduction

Recently, there has been considerable interest in representing graphs with as few objects as possible. The idea behind this objective is to keep the visual complexity of a drawing low for the observer. The types of objects that have been used are straight-line segments [4–7] and circular arcs [6, 10].

Chaplick et al. [1] considered *covering* straight-line drawings of graphs by unbounded objects (lines, planes) and defined the following new graph parameters. Let $1 \leq l < d$, and let G be a graph. The *l-dimensional affine cover* number of G in $\mathbb{R}^d$, denoted by $\rho_d^l(G)$, is defined as the minimum number of l-dimensional planes in $\mathbb{R}^d$ such that G has a crossing-free straight-line drawing that is contained in the union of these planes. The *weak* l-dimensional affine cover number of G in $\mathbb{R}^d$, denoted by $\pi_d^l(G)$, is defined similarly to $\rho_d^l(G)$, but under the weaker restriction that the vertices (and not necessarily the edges) of G are contained in the union of the planes.

Clearly, for any suitable combination of l and d, it holds that $\pi_d^l(G) \leq \rho_d^l(G)$. For any graph G, if $l' \leq l$ and $d' \leq d$ then $\pi_d^l(G) \leq \pi_{d'}^{l'}(G)$ and $\rho_d^l(G) \leq \rho_{d'}^{l'}(G)$. Chaplick et al. showed that it suffices to study the parameters ρ_2^1, ρ_3^1, ρ_3^2, and π_2^1, π_3^1, π_3^2:

▶ **Theorem 1** (Collapse of the Affine Hierarchy [1]). *For any integers $1 \leq l < 3 \leq d$ and for any graph G, it holds that $\pi_d^l(G) = \pi_3^l(G)$ and $\rho_d^l(G) = \rho_3^l(G)$.*

We call $\pi_2^1(G)$ and $\pi_2^1(G)$ also the 2D and 3D *weak line cover number* of G, respectively. For a given graph G, $\pi_2^1(G)$ is at most as large as $\rho_2^1(G)$ but it can be much smaller. For instance, Chaplick et al. showed that for the nested-triangles graph $T_k = C_3 \times P_k$ (shown in Fig. 2 for $k = 4$) with $n = 3k$ vertices it holds that $\rho_2^1(T_k) \geq n/2$, whereas clearly $\pi_2^1(T_k) \leq 3$.

Chaplick et al. [2] also investigated the complexity of computing the (weak) affine cover numbers. Among others, they showed that in 3D, for $l \in \{1, 2\}$, it is NP-complete to decide whether $\pi_3^l(G) \leq 2$ for a given graph G. In 2D, the question is still open.

▶ **Open Problem 1.** Is it NP-hard to compute, for a given planar graph G, its weak line cover number $\pi_2^1(G)$?

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Independently of this complexity issue, Chaplick et al. also asked the following question:

▶ **Open Problem 2.** Does the class of planar graphs admit a sublinear upper bound for π_2^1?

Even for restricted graph classes, the problem remains open. So far only two graph families with unbounded π_2^1-value are known [1, 9]. The first graph family, by Ravsky and Verbitsky [9], has treewidth 8 (which we define below). As Da Lozzo et al. [3] noted, the construction can be modified so that the treewidth of the graphs in the family becomes 5. In the second graph family, by Chaplick et al. [1], the maximum degree is bounded by 12 and $\pi_2^1(G) \geq n^{0.01}$. This yields a new type of open question:

▶ **Open Problem 3.** How small can we make the maximum degree in a family of planar graphs such that their π_2^1-value is still unbounded?

In this paper, we'll improve upon the result of Chaplick et al. in this respect.

We can also ask the opposite question – if we restrict the maximum degree of a graph family, how large can we make its π_2^1-value?

▶ **Open Problem 4.** Does the class of planar graphs with constant maximum degree admit a sublinear upper bound on π_2^1? In particular, is there a constant upper bound on π_2^1 for the class of planar graphs of maximum degree 3?

Another thread of research concerns the (un)boundedness of π_2^1 in the class of graphs of bounded treewidth. A graph has treewidth at most k if it is a subgraph of a k-tree. The class of *k-trees* (consisting of not necessarily planar graphs) is defined recursively as follows. The complete graph K_{k+1} is a k-tree; if G is a k-tree and H is obtained from G by adding a new vertex and connecting it to a k-clique of G then H is also a k-tree. Observe that the 1-trees are exactly the usual trees.

It is well known that the treewidth of any outerplanar graph is at most 2, and all graphs of treewidth 2 are planar. Chaplick et al. [1] proved that $\pi_2^1(G) \leq 2$ for any outerplanar graph G and asked the following question, which is still open.

▶ **Open Problem 5.** Does the class of treewidth-2 graphs have constant π_2^1-value?

Our contribution. As a warm-up, we compute the weak line cover numbers π_2^1 and π_3^1 of the Platonic graphs (1-skeletons of the Platonic solids) in 2D and 3D, respectively; see Section 2. Our main result is the construction of an infinite family of polyhedral graphs with maximum degree 6, treewidth 3, and unbounded π_2^1-value; see Section 3.

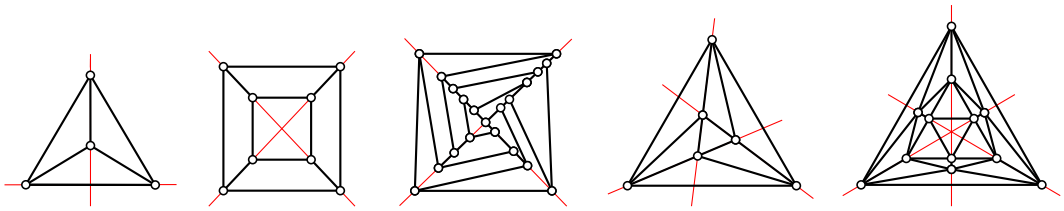
2 Optimal Weak Line Covers of the Platonic Graphs

The Platonic solids are convex polyhedra; hence, the Platonic graphs are planar. Kryven et al. [8] computed various parameters of visual complexity for these graphs. We compute the weak line cover numbers of the Platonic graphs in 2D and 3D. Interestingly, two graphs in this family behave differently in 2D and 3D.

Recall that a *linear forest* is a forest whose connected components are paths.

▶ **Proposition 1.** *Let G be a Platonic graph. Then*

(a) $\pi_2^1(G) = 2$ if G is the graph of the tetrahedron, the cube, or the dodecahedron;
(b) $\pi_2^1(G) = 3$ if G is the graph of the octahedron or the icosahedron.



(a) tetrahedron **(b)** cube **(c)** dodecahedron **(d)** octahedron **(e)** icosahedron

Figure 1 π_2^1-optimal drawings of the Platonic graphs.

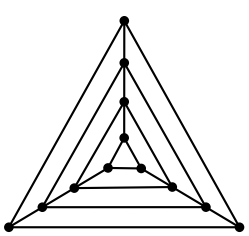


Figure 2 The nested-triangles graph $T_4 = C_3 \times P_4$.

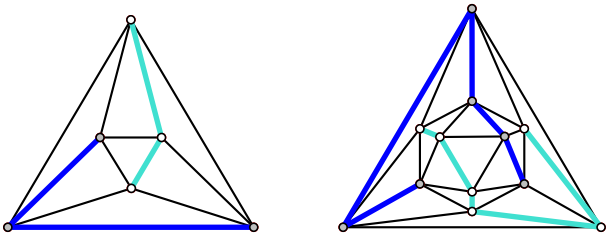


Figure 3 For any Platonic graph, its vertex set can be partitioned into two subsets that each induces a linear forest.

Proof. (a) See Figs. 1a, 1b, and 1c and note that only linear forests have π_2^1-value 1.

(b) Let G be the graph of the octahedron or the icosahedron. Then $\pi_2^1(G) \leq 3$; see Figs. 1d and 1e. On the other hand, $\pi_2^1(G) \geq 3$. Indeed, assume that there exists a plane drawing of G such that all vertices of G are covered by two straight lines ℓ_1 and ℓ_2. Since the outer face of G is a triangle, one of these straight lines, say ℓ_1, contains two vertices of the outer face. Thus ℓ_1 contains no other vertices of G; all of them are placed on ℓ_2. But this is impossible since the subgraph induced by these vertices is not a linear forest (in fact, it even contains a triangle), a contradiction. Hence, $\pi_2^1(G) = 3$. ◀

Chaplick et al. [1] related the affine cover numbers to standard combinatorial characteristics of graphs. The *linear vertex arboricity* $\mathrm{lva}(G)$ of a graph G is the smallest size r of a partition $V(G) = V_1 \cup \cdots \cup V_r$ such that every V_i induces a linear forest. Chaplick et al. showed that the combinatorial parameter $\mathrm{lva}(G)$ actually coincides with the geometric parameter $\pi_3^1(G)$.

▶ **Theorem 2** ([1]). *For any graph G, it holds that $\pi_3^1(G) = \mathrm{lva}(G)$.*

We exploit this to compute the π_3^1-values of the Platonic graphs.

▶ **Proposition 2.** *For any Platonic graph G, it holds that $\pi_3^1(G) = 2$.*

Proof. If G is the graph of the tetrahedron, the cube or the dodecahedron, the claim follows from the fact that $\pi_3^1(G) \leq \pi_2^1(G)$. If G is the graph of the octahedron or the icosahedron, we use Theorem 2 and note that $\mathrm{lva}(G) = 2$; see Fig. 3. ◀

3 A Family of Graphs with Unbounded Weak Line Cover Number

We say that two plane graphs are *strongly equivalent* (have the same combinatorial embedding) if they are obtainable from one another by a plane homeomorphism, and are *equivalent* if they are obtainable from one another by a plane homeomorphism, up to the choice of the outer face.

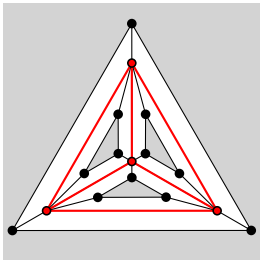


Figure 4 The graph G_1 from the proof of Proposition 3. The graph H_1 is obtained from G_1 by removing the vertices of the outer face.

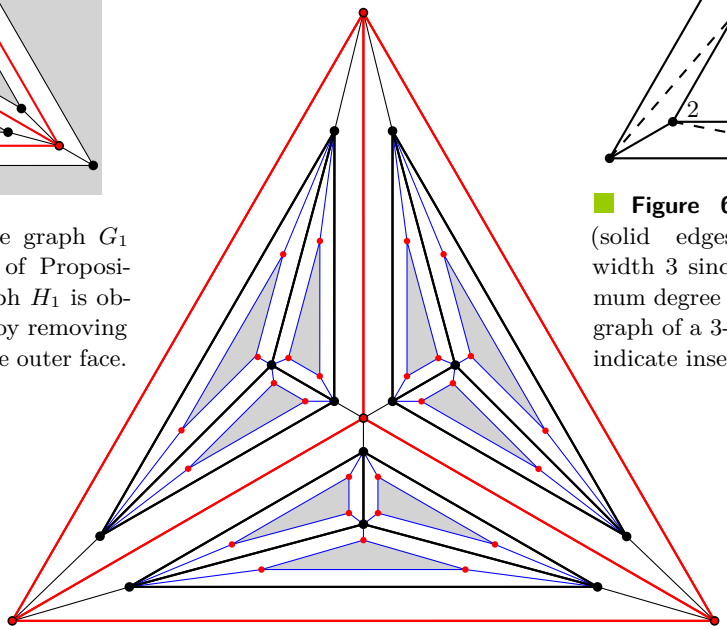


Figure 5 The plane graph H_2 is constructed by replacing each of the special faces of H_1 by another copy of H_1.

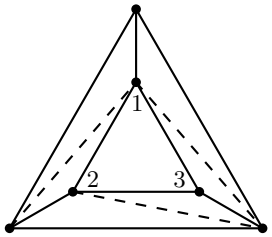


Figure 6 The prism (solid edges) has treewidth 3 since it has minimum degree 3 and is a subgraph of a 3-tree (numbers indicate insertion order).

▶ **Proposition 3.** *There is an infinite family of polyhedral graphs with maximum degree* 6*, treewidth* 3*, and unbounded (logarithmic)* π_2^1*-value.*

Proof. The base of our inductive construction is the graph G_1 depicted in Fig. 4. It has four *special* gray *faces* (the three triangles and the outer face); they are disjoint from the unique K_4-subgraph (red in Fig. 4), which we imagine to be a tetrahedron with four equal faces. Let H_1 be the *plane* graph obtained by removing from G_1 the vertices of its outer face. Assume that, at the induction step $i \geq 1$, we are given a graph G_i and a plane graph H_i with $4 \cdot 3^{i-1}$ and 3^i special faces, respectively. We construct a graph G_{i+1} from G_i and a plane graph H_{i+1} from H_i by replacing each of their special faces by a copy of the graph H_1; see Fig. 5. Then G_{i+1} and H_{i+1} have $4 \cdot 3^i$ and 3^{i+1} special faces, respectively. Note that G_i can be naturally interpreted as a 1-skeleton of a convex polyhedron, and the construction of G_{i+1} preserves this property. Thus, the graph G_{i+1} is polyhedral.

Using the fact that the special faces of G_{i+1} are disjoint from G_i, it is easy to see that $\Delta(G_{i+1}) = 6$. Since G_i is polyhedral, by a well-known result of Whitney [11], all its plane embeddings are equivalent. But, independently of the choice of the outer face in this equivalence, each plane embedding of G_i contains a subgraph strongly equivalent to H_i.

It is easy to check that we can build a 1-skeleton of a triangular prism from a triangle keeping treewidth 3; see Fig. 6. So during both actions, (a) attaching to a tetrahedron gray triangles in order to construct G_1 and (b) replacing each of the special faces of the graph G_i by a copy of H_1, the treewidth of the resulting graph remains at most 3. On the other hand, the treewidth of G_i is lowerbounded by the minimum degree of G_i, which is 3.

We need at least two straight lines to cover all vertices of a graph strongly equivalent to H_1. For $i \geq 1$, let the *central vertex* of H_i be the unique vertex that is incident to all vertices on the outer face. If $i \geq 1$, it is not difficult to check that, for each straight line ℓ

containing the central vertex of a graph strongly equivalent to H_i, there exists a subgraph strongly equivalent to H_{i-1} drawn inside a special face of H_i disjoint from ℓ. Taking into account this subgraph, we need at least one more line to cover the central vertex of the graph strongly equivalent to H_i. By induction we see that we need at least $i+1$ straight lines to cover all vertices of a graph strongly equivalent to H_i. Since any drawing of G_i contains such a graph, we have $\pi_2^1(G_i) \geq i+1$. The graph G_i has $n_i = 20 \cdot 3^{i-1} - 4$ vertices, thus $\pi_2^1(G_i) \in \Omega(\log n_i)$. ◀

References

1 Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. Drawing graphs on few lines and few planes. In Yifan Hu and Martin Nöllenburg, editors, *Proc. 24th Int. Symp. Graph Drawing & Network Vis. (GD'16)*, volume 9801 of *LNCS*, pages 166–180. Springer, 2016. URL: `http://arxiv.org/abs/1607.01196`.

2 Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. The complexity of drawing graphs on few lines and few planes. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Proc. Algorithms Data Struct. Symp. (WADS'17)*, volume 10389 of *LNCS*, pages 265–276. Springer, 2017. URL: `http://arxiv.org/abs/1607.06444`, `doi:10.1007/978-3-319-62127-2_23`.

3 Giordano Da Lozzo, Vida Dujmovic, Fabrizio Frati, Tamara Mchedlidze, and Vincenzo Roselli. Drawing planar graphs with many collinear vertices. In Yifan Hu and Martin Nöllenburg, editors, *Proc. 24th Int. Symp. Graph Drawing & Network Vis. (GD'16)*, volume 9801 of *LNCS*, pages 152–165. Springer, 2016. `doi:10.1007/978-3-319-50106-2_13`.

4 Vida Dujmović, David Eppstein, Matthew Suderman, and David R. Wood. Drawings of planar graphs with few slopes and segments. *Comput. Geom. Theory Appl.*, 38(3):194–212, 2007. `doi:10.1016/j.comgeo.2006.09.002`.

5 Stephane Durocher and Debajyoti Mondal. Drawing plane triangulations with few segments. In *Proc. 26th Canad. Conf. Comput. Geom. (CCCG'14)*, pages 40–45, 2014. URL: `http://cccg.ca/proceedings/2014/papers/paper06.pdf`.

6 Gregor Hültenschmidt, Philipp Kindermann, Wouter Meulemans, and André Schulz. Drawing planar graphs with few geometric primitives. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Proc. 43th Int. Workshop Graph-Theoretic Concepts Comput. Sci. (WG'17)*, volume 10520 of *LNCS*, pages 316–329. Springer, 2017. `doi:10.1007/978-3-319-68705-6_24`.

7 Philipp Kindermann, Wouter Meulemans, and André Schulz. Experimental analysis of the accessibility of drawings with few segments. In Fabrizio Frati and Kwan-Liu Ma, editors, *Proc. 25th Int. Symp. Graph Drawing & Network Vis. (GD'17)*, volume 10692 of *LNCS*, pages 52–64. Springer, 2017. `doi:10.1007/978-3-319-73915-1_5`.

8 Myroslav Kryven, Alexander Ravsky, and Alexander Wolff. Drawing graphs on few circles and few spheres. In B.S. Panda and Partha P. Goswami, editors, *Proc. 4th Annu. Int. Conf. Algorithms Discrete Appl. Math. (CALDAM'18)*, volume 10743 of *LNCS*, pages 164–178. Springer, 2018. URL: `https://arxiv.org/abs/1709.06965`.

9 Alexander Ravsky and Oleg Verbitsky. On collinear sets in straight-line drawings. In Petr Kolman and Jan Kratochvíl, editors, *Proc. 37th Int. Workshop Graph-Theoretic Concepts Comput. Sci. (WG'11)*, volume 6986 of *LNCS*, pages 295–306. Springer, 2011. URL: `http://arxiv.org/abs/0806.0253`, `doi:10.1007/978-3-642-25870-1_27`.

10 André Schulz. Drawing graphs with few arcs. *J. Graph Algorithms Appl.*, 19(1):393–412, 2015. `doi:10.7155/jgaa.00366`.

11 Hassler Whitney. Congruent graphs and the connectivity of graphs. *Amer. J. Math.*, 54:150–168, 1932. `doi:10.2307/2371086`.

Convexity-Increasing Morphs of Planar Graphs

Linda Kleist[1], Boris Klemz[2], Anna Lubiw[3], Lena Schlipf[4], Frank Staals[5], and Darren Strash[6]

1 **Technische Universität Berlin, Germany,** kleist@math.tu-berlin.de
2 **Freie Universität Berlin, Germany,** klemz@inf.fu-berlin.de
3 **University of Waterloo, Canada,** alubiw@uwaterloo.ca
4 **FernUniversität in Hagen, Germany,** lena.schlipf@fernuni-hagen.de
5 **Utrecht University, The Netherlands,** f.staals@uu.nl
6 **Colgate University, USA,** dstrash@cs.colgate.edu

Abstract

We study the problem of *convexifying* drawings of planar graphs. Given any planar straight-line drawing of a 3-connected graph G, we show how to morph the drawing to one with convex faces while maintaining planarity at all times. Furthermore, the morph is *convexity increasing*, meaning that angles of inner faces never change from convex to reflex. We give a polynomial time algorithm that constructs such a morph as a composition of a linear number of steps where each step either moves vertices along horizontal lines or moves vertices along vertical lines.

1 Introduction

A *morph* between two planar straight-line drawings Γ_0 and Γ_1 of a graph G is a continuous movement of the vertices from one to the other, with the edges following along as straight-line segments between their endpoints. A morph is *planar* if it preserves planarity of the drawing at all times. Motivated by applications in animation and in reconstruction of 3D shapes from 2D slices, the study of morphing has focused on finding a morph between two given planar drawings. The existence of planar morphs was established long ago [4, 15], followed by algorithms that produce good visual results [7, 8], and algorithms that find "piece-wise linear" morphs with a linear number of steps [1, 2].

Our focus is somewhat different, and more aligned with graph drawing goals—our input is a planar graph drawing and our aim is to morph it to a better drawing, in particular to a convex drawing. A morph *convexifies* a given straight-line graph drawing if the end result is a *convex graph drawing*, i.e. a planar straight-line graph drawing in which every face is a convex polygon.

We first observe that it is easy, using known results, to find a planar morph that convexifies a given graph drawing—we can just create a convex drawing with the same faces (assuming such a drawing exists), and morph to that specific drawing using the known planar morphing algorithms. However, ideally, convex angles should remain convex throughout a morph. We therefore impose the stronger condition that the morph be *convexity-increasing*, meaning that an angle of an inner face never switches from convex to reflex. Besides the theoretical goal of studying continuous motion that is monotonic in some measure (e.g. edge lengths [10]), another motivation comes from visualization—a morph of a graph drawing should maintain the user's "mental model" [13] which means changing as little as possible, and making observable progress towards a goal. Previous morphing algorithms fail to provide convexity-increasing morphs even if the target is a convex drawing because they all start by triangulating the drawing. This means that an original convex angle may be subdivided by new triangulation edges, so there is no constraint that keeps it convex. (An exception is Angelini et al. [2] which morphs a convex drawing to a convex drawing, preserving convexity.)

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

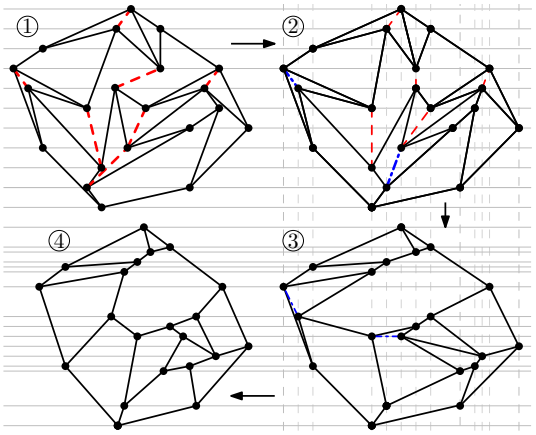


Figure 1 A sequence of convexity-increasing morphs (horizontal, vertical, horizontal) that morph a straight-line drawing of a graph G (drawn in black) into a strictly convex drawing of G.

Our Results. We give the first algorithm to convexify any straight-line planar drawing of a 3-connected graph via a planar convexity-increasing morph. We show a surprising stronger property—that the morph can be composed of a linear number of *horizontal* and *vertical* morphs. A *horizontal* morph moves all vertices at constant speeds along horizontal lines, and a *vertical* morph is defined similarly. See Figure 1. Orthogonality is a very desirable and well-studied criterion for graph drawing [6], in part because there is evidence that the human visual cortex comprehends orthogonal lines more easily [3, 14]. Similarly, it seems natural that orthogonal motion should be easier to comprehend, though morphing algorithms have so far not explored this criterion. To be precise, we prove the following theorem.

▶ **Theorem 1.1.** *Let Γ be a planar straight-line drawing of a 3-connected graph G on n vertices. Then Γ can be morphed to a strictly convex drawing via a sequence of convexity-increasing planar morphs each of which is either a horizontal morph or a vertical morph. If Γ has a convex outer face then the number of morphs in the sequence is at most $r + 1$, where r is the number of internal reflex angles in Γ. In general, the number of morphs in the sequence is at most $1.5n$. Furthermore there is an $O(n^{1+\omega/2})$ time algorithm to find the sequence of morphs, where ω is the matrix multiplication exponent.*

Due to space constraints, some proofs in this paper are only sketched or omitted entirely. Full proofs of all claims can be found in the full preprint [11].

2 Preliminaries

Two planar drawings of a graph G have the *same combinatorial embedding* if they have the same clockwise cyclic ordering of edges around the outer face and around each inner face. We use the terms *convex*, *strictly convex*, *reflex* with their standard meanings. We say that a face of a planar graph drawing is *y-monotone* if the boundary of the face consists of two y-monotone chains. A chain is *y-monotone* if the y-coordinates of points along the chain are strictly increasing.

Horizontal and vertical morphs are special cases of *unidirectional morphs* [1] which move vertices along parallel lines, with each vertex moving at constant speed (different vertices are allowed to move at different speeds, and some may remain stationary). A unidirectional morph is completely specified by the initial and final drawings. We use the notation $\langle \Gamma_1, \Gamma_2 \rangle$

to denote the linear morph from drawing Γ_1 to Γ_2. We use the following nice properties of unidirectional morphs; details are in the long version.

▶ **Lemma 2.1.** *[1, Lemma 13] If Γ and Γ' are two planar straight-line drawings of the same graph such that every line parallel to the x-axis crosses the same ordered sequence of edges and vertices in both drawings, then the linear morph from Γ to Γ' is planar.*

▶ **Lemma 2.2.** *Let $\Gamma_1, \Gamma_2, \Gamma_3$ be three planar straight-line drawings where the linear morphs $\langle \Gamma_i, \Gamma_{i+1} \rangle$, $i = 1, 2$ are horizontal and planar. Then the linear morph $\langle \Gamma_1, \Gamma_3 \rangle$ is a horizontal planar morph.*

▶ **Lemma 2.3.** *During a horizontal morph, an angle cannot change more than once between reflex and convex or vice versa. If $\langle \Gamma_1, \Gamma_2 \rangle$ is a horizontal morph and any convex internal angle of Γ_1 is also convex in Γ_2 then the morph is convexity-increasing.*

▶ **Observation 2.4.** *[1, Lemma 13] If Γ is a (not necessarily straight-line) planar graph drawing of a graph G with all faces (including the outer face) y-monotone and Γ' is another planar drawing of G that has the same combinatorial embedding, the same y-coordinates of vertices, and has y-monotone edges, then every line parallel to the x-axis crosses the same ordered sequence of edges and vertices in both drawings.*

2.1 Redrawing with Convex Faces while Preserving y-Coordinates

We build upon an algorithm due to Hong and Nagamochi [9] that redraws a planar graph to have convex faces while preserving the y-coordinates of the vertices. Angelini et al. [2] strengthened the result to strictly convex faces. We limit ourselves to 3-connected graphs and improve the running time.

▶ **Lemma 2.5** (based on [9, 2])**.** *Let Γ be a planar drawing of a 3-connected graph G such that every face is y-monotone (including the outer face). Let C be a strictly convex straight-line drawing of the outer face of G such that every vertex of C has the same y-coordinate as in Γ. Then there is a straight-line strictly convex drawing Γ' of G that has C as the outer face and such that every vertex of Γ' has the same y-coordinate as in Γ. Furthermore, Γ' can be found in time $O(n^{\omega/2})$, where ω is the matrix multiplication exponent.*

We prove Lemma 2.5 in the long version using Tutte's graph drawing algorithm. This is quite different from the previous approaches, and gives the improved run-time.

Hong and Nagamochi [9] proved a version of Lemma 2.5, but did not guarantee a strictly convex drawing and gave a run-time of $O(n^2)$. Angelini et al. [2] strengthened the result to strictly convex faces by perturbing vertices to avoid angles of 180°. They did not analyze run-time. Our run time is $O(n^{1.5})$ without fast matrix multiplication, $O(n^{1.1865})$ with. Both [9] and [2] expressed their results in terms of *level planar drawings* of *hierarchical plane st-graphs*, and handled more generally the class of graphs that have [strictly] convex drawings.

3 Computing Convexity-Increasing Morphs

3.1 Morphing Drawings with a Convex Outer Face

To give some intuition about the proof, we first consider an easy case where the outer face of Γ is strictly convex and all faces are y-monotone. Then we can immediately apply Lemma 2.5 with the outer face fixed to obtain a new straight-line strictly convex drawing Γ' with all

vertices at the same y-coordinates. The properties of unidirectional morphs can then be used to show that the morph from Γ to Γ' is planar, horizontal, and convexity-increasing.

A face f is y-monotone if and only if it has only one *local maximum* and only one *local minimum*, where a vertex v is a *local minimum* (*local maximum*) of face f if the neighbors of v in f lie above v (below v, respectively). A *local extremum* refers to a local minimum or a local maximum. We can augment a graph drawing to have y-monotone faces by adding y-monotone edges (not necessarily straight-line). This is a standard operation in upward planar (or "monotone") drawing [5, Lemma 4.1] [12, Lemma 3.1], but we need the stronger property that new edges are only incident to local extrema:

▶ **Proposition 3.1.** *Any straight-line planar graph drawing can be augmented to have y-monotone inner faces by adding edges such that each edge can be drawn as a y-monotone curve joining two local extrema in some face. Furthermore, these edges can be found in time $O(n \log n)$.*

This proposition allows us to prove the following:

▶ **Lemma 3.2.** *Let Γ be a straight-line planar drawing with a convex outer face and no horizontal edge. There exists a horizontal planar morph to a straight-line drawing Γ' such that Γ' has a strictly convex outer face and every internal angle that is not a local extremum is strictly convex in Γ'. Furthermore, the morph is convexity-increasing, and can be found in time $O(n^{\omega/2})$, where ω is the matrix multiplication exponent.*

Proof sketch. We use Proposition 3.1 to augment Γ with a set of edges A such that $\Gamma \cup A$ is a planar drawing in which all faces are y-monotone, and any edge of A goes between two local extrema in some inner face. This takes time $O(n \log n)$. Next, we apply Lemma 2.5 to obtain a new straight-line strictly convex drawing $\Gamma' \cup A'$ with all vertices at the same y-coordinates as in Γ. Finally, we apply the unidirectional properties to prove that the morph from Γ to Γ' is planar, horizontal, and convexity-increasing. Any internal angle of Γ that is not a local extremum has no edge of A incident to it, and thus is strictly convex in Γ'. The time required is $O(n^{\omega/2})$. ◀

Lemma 3.2 convexifies any *h-reflex* angle, where a reflex angle of inner face f is *h-reflex* if it occurs at a vertex that has one neighbor in f above and the other below (i.e., it is not a local extremum of f). To convexify the remaining reflex angles, the plan is to conceptually "turn the paper" by 90° and perform a vertical morph to make any *v-reflex* angle convex, where a reflex angle of inner face f is called *v-reflex* angle if it occurs at a vertex that has one neighbor in f to the left and the other to the right. The final aspect of the proof is to ensure that at each step there is at least one h-reflex or v-reflex angle, so that the algorithm makes progress in each step. To do this we strengthen Lemma 3.2:

▶ **Lemma 3.3.** *Let Γ be a straight-line planar drawing with a convex outer face and no horizontal edge. There exists a horizontal planar morph to a straight-line drawing Γ'' such that*
(i) the outer face of Γ'' is strictly convex,
(ii) every internal angle that is not a local extremum is convex in Γ'',
(iii) Γ'' has no vertical edge, and
(iv) if Γ'' is not convex, then it has at least one v-reflex angle.
Furthermore, the morph is convexity-increasing, and can be found in time $O(n^{\omega/2})$.

Proof sketch. See Figure 2. We first apply Lemma 3.2 to obtain a morph from Γ to a drawing Γ' that satisfies (i) and (ii). If Γ' satisfies all the requirements, we are done. Otherwise

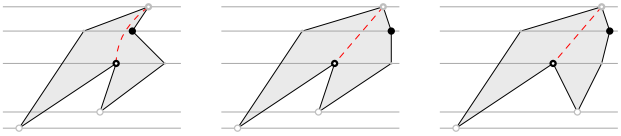


Figure 2 (left) A face that is not y-monotone. (middle) The face after application of Lemma 3.2. There is a vertical edge and the single reflex vertex is not v-reflex. (right) After applying a horizontal shear transformation, the reflex vertex is v-reflex and there are no vertical edges.

we will achieve the remaining properties by choosing one reflex angle (necessarily a local extremum), say at vertex u in inner face f, and applying a horizontal shear transformation to create a drawing Γ'' in which the angle at u becomes v-reflex. Since shearing is an affine transformation, the convexity status of all angles is preserved. We then use the properties of unidirectional morphs to show that the morph from Γ to Γ'' is planar, horizontal, and convexity-increasing. The morph can be found in time $O(n^{\omega/2})$. ◀

Proof sketch of Theorem 1.1 for a convex outer face. Apply Lemma 3.3 alternately in the horizontal and vertical directions until the drawing is convex. In each step there is at least one h-reflex or v-reflex angle that becomes convex. Thus the number of horizontal morphs is at most $r + 1$ and the run-time is $O(n^{1+\omega/2})$. ◀

3.2 Morphing Drawings with a Non-convex Outer Face

In this section we outline the proof of Theorem 1.1 when the outer face is not convex. We augment the outer face of Γ with new edges A from its convex hull to obtain a drawing with a convex outer face. We apply the results from Section 3.1 to morph to a strictly convex drawing and then remove the edges of A one-by-one. After each edge is removed we morph to a strictly convex drawing of the reduced graph using at most three horizontal or vertical morphs. Each edge $e \in A$ is part of the boundary of an inner face f_e of $\Gamma \cup A$. We call f_e the *pocket* of e. In order to remove edge e we "pop" its pocket outward using the following lemma:

▶ **Lemma 3.4.** *Let Γ be a strictly convex drawing of graph G, with an edge e on the outer face. Suppose that $G - e$ is 3-connected. Then $\Gamma - e$ can be morphed to a strictly convex drawing of $G - e$ via at most three convexity-increasing morphs, each of which is horizontal or vertical. Furthermore, the morphs can be found in time $O(n^{\omega/2})$.*

See Figure 3 for the proof idea. Observe that each application of Lemma 3.4 increases the number of vertices of G on the convex hull. Thus, by induction we obtain the proof of Theorem 1.1. We have used at most 3 horizontal and vertical morphs per pocket. To obtain the initial strictly convex drawing of $\Gamma \cup A$ we require at most n morphs. Thus, the total number of morphs is bounded by $4n$. In the full version we decrease the total number of morphs to $1.5n$. The run time of the algorithm is $O(n^{1+\omega/2})$.

Acknowledgments. We wish to thank André Schulz for helpful discussions on generalizations of Tutte's algorithm. This work was begun at Dagstuhl workshop 17072, "Applications of Topology to the Analysis of 1-Dimensional Objects." We thank Dagstuhl, the organizers, and the other participants for a stimulating workshop. In particular, we want to thank Carola Wenk and Regina Rotmann for joining some of our discussions, and Irina Kostitsyna for contributing many valuable ideas.

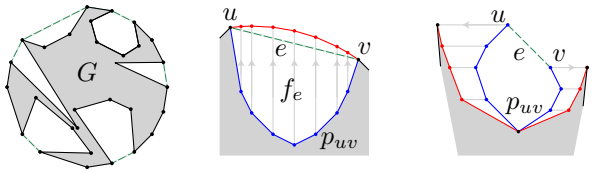


Figure 3 (left) Schematic of the convex drawing of $G \cup A$. Graph G is depicted in gray, edges of A are dashed, and the pockets are white. (middle),(right) Illustration of Lemma 3.4. If a pocket p_{uv} is x-monotone we can pop it out (middle), otherwise we first make it x-monotone (right).

References

1 S. Alamdari, P. Angelini, F. Barrera-Cruz, T. M. Chan, G. Da Lozzo, G. Di Battista, F. Frati, P. Haxell, A. Lubiw, M. Patrignani, V. Roselli, S. Singla, and B. T. Wilkinson. How to morph planar graph drawings. *SIAM J. Computing*, 46(2):29 pages, 2017.

2 P. Angelini, G. Da Lozzo, F. Frati, A. Lubiw, M. Patrignani, and V. Roselli. Optimal Morphs of Convex Drawings. In L. Arge and J. Pach, editors, *Proceedings of the 31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 126–140, Dagstuhl, Germany, 2015.

3 S. Appelle. Perception and discrimination as a function of stimulus orientation: the "oblique effect" in man and animals. *Psychological Bulletin*, 78(4):266, 1972.

4 S. Cairns. Deformations of plane rectilinear complexes. *The American Mathematical Monthly*, 51(5):247–252, 1944.

5 G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2-3):175–198, 1988.

6 M. Eiglsperger, S. P. Fekete, and G. W. Klau. Orthogonal graph drawing. In *Drawing Graphs*, pages 121–171. Springer, 2001.

7 M. S. Floater and C. Gotsman. How to morph tilings injectively. *Journal of Computational and Applied Mathematics*, 101(1-2):117–129, 1999.

8 C. Gotsman and V. Surazhsky. Guaranteed intersection-free polygon morphing. *Computers & Graphics*, 25(1):67–75, 2001.

9 S.-H. Hong and H. Nagamochi. Convex drawings of hierarchical planar graphs and clustered planar graphs. *Journal of Discrete Algorithms*, 8(3):282–295, 2010.

10 H. N. Iben, J. F. O'Brien, and E. D. Demaine. Refolding planar polygons. *Discrete & Computational Geometry*, 41(3):444–460, 2009.

11 L. Kleist, B. Klemz, A. Lubiw, L. Schlipf, F. Staals, and D. Strash. Convexity-increasing morphs of planar graphs. *CoRR*, abs/1802.06579, 2018.

12 J. Pach and G. Tóth. Monotone drawings of planar graphs. *Journal of Graph Theory*, 46(1):39–47, 2004.

13 H. C. Purchase, E. Hoggan, and C. Görg. How important is the "mental map"?–an empirical investigation of a dynamic graph layout algorithm. In *International Symposium on Graph Drawing*, pages 184–195. Springer, 2006.

14 H. C. Purchase, C. Pilcher, and B. Plimmer. Graph drawing aesthetics—created by users, not algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):81–92, 2012.

15 C. Thomassen. Deformations of plane graphs. *Journal of Combinatorial Theory, Series B*, 34(3):244–257, 1983.

On the Topology of Walkable Environments

Benjamin Burton[1], Arne Hillebrand[2], Maarten Löffler[2], Schleimer, Saul[3], Dylan Thurston[4], Stephan Tillmann[5], and Wouter van Toll[2]

1 **University of Queensland, Australia**
2 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands**
3 **University of Warwick - Coventry, Great Britain**
4 **Indiana University - Bloomington, United States of America**
5 **University of Sydney, Australia**

Abstract

Motivated by motion planning applications, we study 2-dimensional surfaces embedded in 3-dimensional space with the property that their vertical projection is an immersion. We provide bounds on the complexity of a triangulation of such a surface, given that the projection of the boundary is a polygon with m segments. We then show how these bounds lead to efficient algorithm to compute such a triangulation. Finally, we relate our result to concrete motion planning setting and review related open questions.

1 Introduction

Simulations and computer games are often occupied by virtual characters that need to move autonomously through a virtual environment. This asks for efficient data structures and algorithms for path planning and crowd simulation. An environment such as a multi-storey building is three-dimensional, but the characters are restricted to surfaces on which they can walk. Therefore, while these surfaces are embedded in $\mathbb{R}^3$, they are (in some ways) locally similar to $\mathbb{R}^2$. They form an interesting class of environments that we call *walkable environments* (WEs).

For the purpose of path planning, it is important to automatically obtain an efficient representation of a walkable environment [3, 4, 9, 10]. In particular, it is desirable to subdivide a walkable environment into surfaces that can each be projected onto $\mathbb{R}^2$ without overlap. We refer to such a decomposition as a *multi-layered environment* (MLE). An individual surface within an MLE is called a *layer*, and the 'cuts' made for the decomposition are called *connections*. The main advantage of an MLE is that each separate layer can be treated as a 2D component, which allows the extension of 2D data structures [9].

Technically, a connection is a curve along the WE between two boundary vertices of the WE. For the application's data structures and algorithms, it is also important that each connection is a straight line segment when projected onto $\mathbb{R}^2$. Furthermore, it is often desirable to obtain an MLE with a small number of connections because this number influences the complexity of various algorithms. To understand these demands better, we first need to study the topological properties of a walkable environment.

Contributions In this paper, we study the topology of walkable environments, and we present an algorithm that triangulates a WE. Suppose a walkable environment $\mathcal{W}$ is given as a triangulated surface with n vertices and m *boundary* vertices. (We will give a more precise definition in Section 2.) We prove that any triangulation of $\mathcal{W}$ has $O(m)$ diagonals, and we present an algorithm that computes a triangulation of $\mathcal{W}$ in $O(n + m \log m)$ time, in such a way that each diagonal is a line segment when projected onto $\mathbb{R}^2$.

At this point, an alert reader may wonder why we should wish to triangulate a surface that is already triangulated. The difference is that the input triangulation has linear

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

complexity in n, while the output triangulation only has linear complexity in m, which can be much smaller. For example, an environment containing hilly terrains has many interior vertices, which will not be present in our output triangulation. The output triangulation does not have straight edges in $\mathbb{R}^3$, but the projections of its edges are straight segments in $\mathbb{R}^2$. This type of triangulation is interesting for path planning applications because these applications often use projected distances [10], and the diagonals of this triangulation may be good candidates for connections in an MLE.

Related work There are several applications and algorithms that are already using some form of a multi-layered environment. Van Toll et al. [9] use an MLE to create a multi-layered navigation mesh based on the medial axis, which allows for fast path planning queries. Rodriguez and Amato [8] convert a multi-layered environment to a roadmap representation, which they then use to find a strategy for efficiently clearing a building. However, both of these works do not describe how to *obtain* such an MLE from an arbitrary 3D environment.

A popular way to convert a 3D environment to a walkable environment is to approximate the environment by 3D grid cells (*voxels*), in which each voxel is marked as walkable or non-walkable. Voxelization typically uses the graphics card. Several methods use this concept as the first step in a pipeline for computing a navigation mesh [1, 6, 7].

Converting a walkable environment to a *multi-layered environment*, preferably with a small number of cuts, is a separate problem. Hillebrand et al. [3, 4] model this as a graph problem. They prove that obtaining an MLE with a minimum number of cuts is NP-hard, but that good MLEs can often be obtained using heuristics. However, these cuts are not always suitable as connections since they are restricted to edges of the input triangulation.

Outline To obtain our results, we first introduce some subtly different concepts in Section 2 that capture the special properties of the surfaces we encounter. In Section 3, we analyse the possible values of the genus of these surfaces. Then, in Section 4, we show how an adaptation of the polygon decomposition algorithm by Lee and Preparata [5] can be used to triangulate our surfaces. Finally, in Section 5, we discuss how these results relate to the problem of finding a good MLE, and we pose the main open question in this area.

2 Definitions

Sloths and walkable environments. We define a *sloth* to be a compact surface Σ continuously embedded in $\mathbb{R}^3$ so that the vertical projection of Σ to $\mathbb{R}^2$ is an immersion with a polygonal boundary consisting of m line segments (or, equivalently, m vertices). That is, the vertical direction is transverse to Σ. We say that a sloth is *realistic* if the turning angle (projected to $\mathbb{R}^2$) around any boundary vertex is at most 360 degrees (aka 2π).

We define a *walkable environment* (WE) as a geometric representation of a realistic sloth by a set of connected triangles in $\mathbb{R}^3$. By definition, a WE has the following properties:

- The angle between the normal of each triangle and the normal of the ground plane is less than 90 degrees.
- The minimal vertical distance between any two triangles in the WE is non-zero, with the exception of shared edges and vertices.

Let n be the total number of vertices in the WE. Note that n can be much larger than the number of *boundary* vertices m. Thus, a WE is a particular type of sloth represented by triangles. Some parts of this paper apply to sloths in general; other parts apply specifically to realistic sloths or to WEs because they rely on extra properties.

Triangulations of sloths. We define a *topological triangulation* of a sloth Σ to be a subdivision of Σ into topological triangles whose vertices coincide with the vertices of Σ; that is, we add arbitrary arcs on the surface that connect pairs of vertices that were not connected before, and the arcs do not intersect each other except possibly at endpoints.

We define a *geometric triangulation* of a sloth Σ to be a topological triangulation with the additional restriction that each edge, when projected to $\mathbb{R}^2$, is a straight line segment.

3 Bounds on the genus of sloths

In this section, we provide bounds on the potential genus of sloths, expressed in the complexity of their boundaries. Specifically, for a sloth Σ with m boundary vertices, we are interested in possible values of $\text{genus}(\Sigma)$ as a linear function of m.

Gauss–Bonnet Recall that the *Euler characteristic* of a compact, orientable surface Σ is:

$$\chi(\Sigma) = 2 - 2\text{genus}(\Sigma) - \#\partial\Sigma,$$

where $\#\partial\Sigma$ is the number of connected components in the boundary of Σ. The following result relates the topology of the surface to its geometry. We use the special case when the curvature vanishes on the interior and the curvature of the boundary is zero except at the polygonal corners.

▶ **Theorem 3.1** (Gauss–Bonnet, flat version)**.** *Let Σ be a surface with a locally Euclidean metric and polygonal boundary, with corners at c_i with interior angle θ_i. Then*

$$\chi(\Sigma) = \frac{1}{2\pi}\sum_i (\pi - \theta_i).$$

Here, $\pi - \theta_i$ should be thought of as the bending angle at c_i: zero if there is no actual corner, positive if the corner is convex as on the boundary of a convex polygon in the plane, and negative if the corner is concave.

Application to sloths We use Theorem 3.1 to prove an upper bound of the genus of a sloth.

▶ **Theorem 3.2.** *Let Σ be a surface with boundary and let $f\colon \Sigma \to \mathbb{R}^2$ be an immersion on the interior of Σ so that $f(\partial\Sigma)$ is a polygonal path with m line segments. Then* $\text{genus}(\Sigma) \le m(m+1)/4$. *Furthermore, there are examples coming from embeddings in $\mathbb{R}^3$ with* $\text{genus}(\Sigma) = (m/8 - 1)^2$.

Proof. The examples achieving quadratic genus growth are "parking garages" $P_{k,l}$, as shown in Figure 1:

- take k parallel rectangular sheets;
- cut out l slits from each sheet (stacked on top of each other); and
- rejoin across the slits, shifting down one level as you go.

We can apply Theorem 3.1 (the Gauss–Bonnet theorem) to the metric on Σ coming from the map to $\mathbb{R}^2$. Here, $\pi - \theta_i$ should be thought of as the bending angle at c_i: zero if there is no actual corner, positive if the corner is convex as on the boundary of a convex polygon in the plane, and negative if the corner is concave. Some of the corners in $P_{k,l}$ are very concave, with a total internal angle of approximately $2k\pi$. The result of this computation is that $\text{genus}(P_{k,l}) = (k-1)(l-1)$. Furthermore, $P_{k,l}$ can be realized with a polygonal boundary with $4k + 4l$ corners.

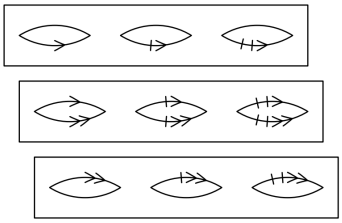


Figure 1 A parking garage, shown for $k = 3$ and $l = 3$. Edges with corresponding marking are glued; this can be achieved by stacking the sheets in 3 dimensions and attaching connecting ramps.

For the upper bounds on genus, we again apply Theorem 3.1 and give an upper bound on the interior angles θ_i. To do this, we first bound the total multiplicity in any region, the degree by which it is covered by Σ. The multiplicity at a point $x \in \mathbb{R}^2$ can be computed by sending a ray out to infinity in either direction from x, and so is at most $m/2$. The angle θ_i at a corner c_i is bounded by 2π times the multiplicity in any adjoining region. This yields the stated upper bound on genus. ◀

The last observation leads to the following theorem:

▶ **Theorem 3.3.** *For a realistic sloth Σ with m boundary vertices,* genus(Σ) *is $O(m)$.*

4 Geometric triangulations of sloths

We are interested in geometric triangulations of walkable environments because these can be useful representations for the purpose of path planning and crowd simulation. In this section, we first discuss triangulations of sloths in general, and then we present an algorithm for triangulating a WE.

▶ **Observation 4.1.** *If a sloth has m vertices, b boundary components, and genus g, then every (topological or geometric) triangulation has $t = 4g + b + m - 4$ triangles.*

▶ **Lemma 4.2.** *Every sloth has at least one geometric triangulation.*

Proof. We show that whenever Σ is not yet triangulated, we can always find a diagonal that splits Σ into two valid sloths, which we can then recursively triangulate again.

Consider a convex vertex v. A *surface ray* is a curve in $\mathbb{R}^3$ that lies completely inside Σ, but whose vertical projection to $\mathbb{R}^2$ is a straight line segment. We shoot a surface ray r from v over the sloth in an arbitrary direction; note that, once we fix the direction, the ray is unique except potentially when it passes through a vertex of Σ. If r does pass through a vertex of Σ, we are happy. If r does not pass through a vertex of Σ, we can continuously rotate r about v until it does; note we can rotate r in two directions until it coincides with either of the incident boundary edges of Σ at v. We distinguish two cases.

1. The ray r passes through a vertex x of Σ, and x is not a neighbour of v on the boundary of Σ. By construction, there is a unobstructed path on Σ from v to x which projects to a line segment. We add edge vx to our triangulation, and recursively triangulate the one or two smaller sloths.
2. When sweeping in both directions, the ray r hit no vertices other than u and w, the neighbours of v on the boundary of Σ. Consider the edge uw, which projects to a straight segment. If uw crosses any existing boundary edge e of Σ, there must be either

be a vertex x contained in the triangle uvw (on Σ), or the edge e crosses (passes over or under) uv or uw. In the first case, we should have found x when sweeping r. In the second case, there must be another edge e' on Σ that e crosses, in order to move over or under uv or uw. We now argue similarly about the endpoinds of e': either, there is an endpoint inside uvw, or e' also crosses uv or uw. Eventually, we run out of edges. Contradiction. We add edge uw to our triangulation, and recursively triangulate the smaller sloth. ◀

Combined with Theorem 3.3, we can conclude the following:

- Every sloth with m vertices has a geometric triangulation with $O(m^2)$ triangles.
- Every sloth with m vertices, whose vertices all have a constant maximum turning angle, has a geometric triangulation with $O(m)$ triangles. By definition, this also holds for every *realistic* sloth, and for every walkable environment with m boundary vertices.
- All triangulations of a given sloth have the same number of triangles and diagonals.

We now describe an algorithm that computes a geometric triangulation of a WE with n vertices and m boundary vertices in $O(n + m \log m)$ time. It applies only to WEs because it relies on the maximum turning angle of realistic sloths around the boundary vertices.

▶ **Lemma 4.3.** *A geometric triangulation of a walkable environment $\mathcal{W}$ with n vertices and m boundary vertices can be computed in $O(n + m \log m)$ time.*

Proof sketch. The idea is to split $\mathcal{W}$ into y-monotone pieces via a 2D plane sweep over all boundary points sorted by y-coordinate, similar to the algorithm by Lee and Preparata [5] for splitting a 2D polygon (with or without holes) into y-monotone pieces. However, several complications arise due to the nature of WEs.

First, we need to obtain a boundary representation of $\mathcal{W}$ such that we can move from any boundary vertex to any adjacent boundary vertex in constant time. This can be done in $O(n)$ time if $\mathcal{W}$ is given as a DCEL.

Using this boundary representation as input, we sweep a plane H parallel to the x- and z-axes, starting at $y = -\infty$, and we maintain a set of curves on H where it intersects $\mathcal{W}$. We may encounter four types of events, reminiscent of the Reeb graph: start events (where a new curve appears), split events (where a curve splits into two curves), merge events (where two curves merge into a single curve), and end events (where a curve disappears).

Since the projection in the z-direction of the boundary of $\mathcal{W}$ is polygonal, and the sweep plane is parallel to the z-axis, all events occur at vertices of $\mathcal{W}$. Furthermore, because $\mathcal{W}$ is a realistic sloth, all events are indeed related to at most two curves. (This would not be the case in the non-realistic parking garage from Figure 1, where a single vertex can induce many curves). Hence, we can detect and sort all events in $O(m \log m)$ time.

Now, we process the events maintaining the latest merge vertex, as in the classic algorithm in [5]. We provide a complete description of the algorithm in the full version of this paper.

After applying the algorithm, we have subdivided $\mathcal{W}$ into a set of y-monotone pieces. Observe that each y-monotone piece P_i is a simple polygon when projected to $\mathbb{R}^2$. Therefore, each P_i (with m_i boundary vertices) can be triangulated in $O(m_i)$ time, and the combined time for triangulating all parts is $O(m)$. The result is a triangulation of the WE.

Combined with the pre-processing time for obtaining a boundary representation, this proves the lemma. ◀

5 Layers and Connections

The results from the previous sections form a first step towards the practical decomposition of a walkable environment into layers for path planning purposes. Recall that we are interested in obtaining a *multi-layered environment* (MLE) with a minimum number of connections, where each connection is a straight line segment when projected onto $\mathbb{R}^2$.

Expressed in terms of sloths, we define a sloth to be *flat* if its projection to $\mathbb{R}^2$ is a polygon without self-intersections; that is, no two points on the sloth project to the same point in $\mathbb{R}^2$. The generalized version of our problem is to decompose a sloth Σ into flat sloths, by cutting Σ in such a way that the endpoints of each cut are boundary vertices of Σ, and each cut projects to a line segment in $\mathbb{R}^2$.

The triangulation algorithm from Section 4 first decomposes Σ into y-monotone pieces (and then into triangles). This subdivision into y-monotone pieces already induces a valid MLE. However, better subdivisions (with fewer connections) might exist.

Let $C^*(\Sigma)$ be the minimum number of connections required for subdividing a sloth Σ into layers. We conclude by stating the following open question:

▶ Question 1. Given a sloth Σ represented as a WE with n internal vertices and m boundary vertices, (how) can we subdivide it into a multi-layered environment with $C^*(\Sigma)$ connections, or with a number of connections that approximates $C^*(\Sigma)$?

Acknowledgements. This research was initiated at Dagstuhl Seminar 17072. We would like to thank all participants of the seminar for their fruitful discussions. M.L was partially supported by the Netherlands Organisation for Scientific Research (NWO) through project no. 614.001.504.

References

1 L. Deusdado, A.R. Fernandes, and O. Belo, *Path planning for complex 3D multilevel environments*. Proc. 24th Spring Conf. on Computer Graphics, pp. 187–194 (2008).

2 J. Guo, F. Hüffner, E. Kenar, R. Niedermeijer, and J. Uhlmann, *Complexity and exact algorithms for multicut*, Proc. Current Trends in Theory and Practice of Computer Science, SOFSEM 3831, pp. 137–147 (2006).

3 A. Hillebrand, M. van den Akker, R. Geraerts, and H. Hoogeveen, *Performing multicut on walkable environments*, Proc. 10th Int. Conf. on Combinatorial Optimization and Applications, pp. 311–325 (2016).

4 A. Hillebrand, M. van den Akker, R. Geraerts, and H. Hoogeveen, *Separating a walkable environment into layers*, Proc. 9th Int. Conf. on Motion in Games, pp. 101–106 (2016).

5 D.T. Lee and F.P. Preparata, *Location of a point in a planar subdivision and its applications*, SIAM Journal of Computing, 6:594–606 (1977).

6 R. Oliva and N. Pelechano, *NEOGEN: Near optimal generator of navigation meshes for 3D multi-layered environments*. Computers & Graphics 37(5), pp. 403–412 (2013).

7 J. Pettré, J.P. Laumond, and D. Thalmann, *A navigation graph for real-time crowd animation on multilayered and uneven terrain*, Proc. First Int. W. Cr. Sim., pp. 81–89 (2005).

8 S. Rodriguez and N.M. Amato, *Roadmap-based level clearing of buildings*, Lecture Notes in Computer Science, vol. 7060 LNCS, pp. 340–352 (2011).

9 W. van Toll, A. F. Cook IV, and R. Geraerts, *Navigation meshes for realistic multi-layered environments*, Proc. Int. Conf. on Intelligent Robots and Systems, pp. 3526–3532 (2011).

10 W. van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts, *A Comparative Study of Navigation Meshes*, Proc. 9th Int. Conf. on Motion in Games, pp. 91–100 (2016).

The hardness of Witness puzzles*

Irina Kostitsyna[1], Maarten Löffler[2], Max Sondag[1], Willem Sonke[1], and Jules Wulms[1]

1 **Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands**
`[i.kostitsyna | m.f.m.sondag | w.m.sonke | j.j.h.m.wulms]@tue.nl`
2 **Dept. of Information and Computing Sciences, Utrecht University, The Netherlands**
`m.loffler@uu.nl`

1 Introduction

The Witness is a puzzle video game that was produced by Thekla [1]. Since its release in 2016 the game has been critically acclaimed, and has been praised for its intelligence and astonishing visuals. The Witness contains 9 principal types of puzzles, and a number of hidden "environmental" puzzles, totaling to an amount of 664 puzzles spread over the open world of the game. A player is invited to explore the world and to deduce the rules of various puzzles they encounter.

Most of the puzzles in the game are based on a square grid, requiring the player to connect a starting point to an end point with a simple grid path while satisfying certain constraints, such as splitting the grid cells of different colors, passing through a set of given points, and drawing polyomino shapes comprising of a set of tetris blocks, among others. For example, the puzzle depicted in Figure 1 asks to connect the bottom-left corner of the grid to the top-right corner with a grid path separating the white square from the black squares.

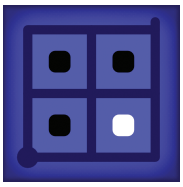


Figure 1 An example of a *Black and White Squares* puzzle from the Witness. (Image from [1].)

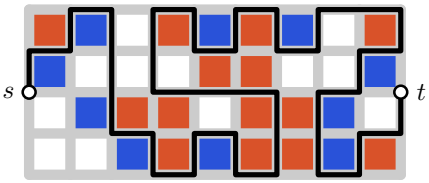


Figure 2 An example of a solved colored squares puzzle. All red squares are separated from the blue squares by the path.

In this short paper we resolve the complexity of two out of the nine types of puzzles in the game, namely the *Black and White Squares* and the *Multicolor Squares* puzzles. We show that in a restricted setting these types of puzzles can be solved in polynomial running time, but that in general they are NP-complete.

To formalize the setting, consider a rectangular grid of size $w \times h$, which we interpret as a graph $G = (V, E)$, where $V = [0, 1, \ldots, w] \times [0, 1, \ldots, h]$ and two vertices (a, b) and (c, d) are connected by an edge in E if and only if $a = c$ and $|b - d| = 1$, or $|a - c| = 1$ and $b = d$. G is a planar graph whose natural embedding decomposes the plane into $(w - 1)(h - 1)$ square faces that we call *cells*, and one unbounded outer face. Cells are colored by a color c from a

* Research on the topic of this paper was initiated at the 2nd Workshop on Applied Geometric Algorithms (AGA 2016) in Vierhouten, The Netherlands, supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208. M.L., M.S. and W.S. are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 614.001.504 (M.L.) and 639.023.208 (M.S. and W.S.). J.W. is supported by the Netherlands eScience Center (NLeSC) under grant number 027.015.G02.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

set $C \cup \{\square\}$. Cells colored by $\square$ are called *empty*. Furthermore, two vertices s and t from V lying on the outer face are selected as the start and end point respectively. The decision version of the puzzle asks whether there exists a simple path P from s to t in the graph G, whose embedding on the grid splits the grid into several connected components of cells, that each contain cells of only one color from C and possibly some empty cells. We call such a path P a color-separating path. An example of a puzzle and its solution is given in Figure 2.

Related work. The Witness puzzle is closely related to Sheep and Wolves, a puzzle where one has to build a fence that separates sheep from wolves on a grid introduced by Dave Tuller.[1] In his version, however, some cells contain additional clues describing the number of fences directly incident to the cell. This puzzle type is known, among other names, as Slither Link, which was proven to be NP-hard by Yato [9]. The study of the computational complexity of puzzle games is as old as the notion of NP-completeness itself [8], and has a rich history, which is beyond the scope of this short note to discuss; instead, we refer the interested reader to the excellent surveys by Demaine and Hearn [3] and by Kendall *et al.* [4].

The object of the puzzles in The Witness, to find a curve separating white from black cells on a grid, is also closely related to the problem of finding a grid-aligned approximation of a shape: given a region overlaid by a grid, we wish to find a curve that has all cells that lie completely inside the shape inside, and all cells that lie completely outside the shape outside, with applications ranging from early computer graphics (casting characters to low-resolution screens) [7] to geographical information systems. Often, one has additional objectives, such as minimizing the symmetric difference or another distance measure of the two shapes. This area remains an active domain of research [2, 6].

2 Algorithms

First we consider the problem every cell is colored either red or blue. We will give an algorithm to solve this problem in time linear in the size of the grid $O(wh)$.

Observations. We can make a couple of basic observations about the path that will help us in understanding the problem. We let a *border* be defined as a maximal set of connected edges between an area of red cells and blue cells. Furthermore we let p_1 and p_2 be the paths from s to t along the edges of the outer face (see Figure 3). We observe that if a color-separating path P exists, then it must go through all the edges of all borders to separate the colors. Moreover as soon as the path P encounters a vertex on some border, it cannot deviate from the border. If it could deviate then only one endpoint of an edge e would be on P as P must be simple, and thus the colors adjacent to e would not be separated. It follows that the path through a border must start at either p_1 or p_2, as that is where the endpoints of the borders are. Large parts of any potential path P are thus fixed.

We categorize the borders into two types: *crossings* (that start on p_1 and end on p_2, or vice versa) and *bumps* (that both start and end on either p_1 or p_2). These are depicted in Figure 4. As a special case, borders that start or end in s or t, respectively, are considered bumps. A third possible type would be an *island* (that does not have any vertices on p_1 or p_2). These however cannot exist if a simple color-separating path P exists, as P would need to contain all edges around the island, which results in a cycle. Furthermore, bumps cannot be nested if there is to be a solution, as any color-separating path needs to traverse all of those bumps, which requires the path to be non-simple.

[1] https://www.amazon.com/Challenge-Brain-Logic-Puzzles-Mensa/dp/1402714491

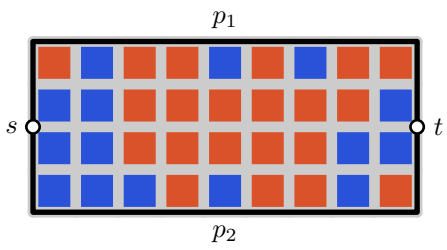


Figure 3 Paths p_1 and p_2 go from s to t along the outer face.

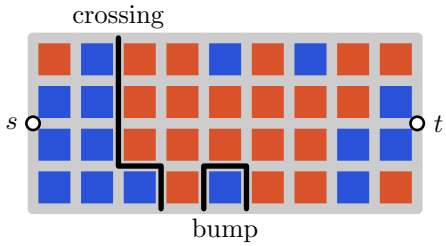


Figure 4 An example of a crossing and a bump are shown in black.

Witty algorithm. The idea of the algorithm is to construct a directed graph G' that admits an st-path if and only if there is a simple color-separating path in our input grid. It works as follows. We cut the grid along all crossings, producing a sequence of pieces $\mathcal{G} = [g_1, g_2, \ldots, g_k]$, ordered such that s is in g_1, t is in g_k, and for all $1 < i < k$, g_i shares a border with g_{i-1} and g_{i+1}. For each piece we remove all vertices (and adjacent edges) that are part of its adjacent crossings, except for the endpoints on p_1 and p_2. As we have subdivided G along all crossings, the only borders left within the pieces are bumps.

For each grid $g \in \mathcal{G}$, we now create a small directed graph g'. Let s_1, s_2 be the vertices on p_1 and p_2, respectively, that are closest to s, and similarly let t_1, t_2 be the vertices on p_1 and p_2 that are closest to t. For g_1 it holds that $s = s_1 = s_2$, and in g_k, $t = t_1 = t_2$. For each grid g we put vertices representing s_1, s_2, t_1, t_2 into g'.

Grids g_i and g_{i+1} used to be connected to each other by a crossing in the original grid. We know that any separating path P must use this crossing and thus we connect t_1 of g'_i to s_2 of g'_{i+1} and s_1 of g'_i to t_2 of g'_{i+1} to form G'. It then remains to determine how P can route within the grids $g \in \mathcal{G}$.

As each border must be followed from start to end by P, the path in a grid g_i must always start at $s_{1,2}$ and end at $t_{1,2}$. We add an edge between s_x and t_y in g' if there exists a color-separating path in g_i starting at s_x and ending at t_y. The existence of such paths can be determined by checking two conditions on g_i. Firstly, if there are bumps on both p_1 and p_2, then there must be at least two grid cells between the bumps on p_1 and p_2 such that the path can pass between them. Secondly, there must be enough vertices between bumps on p_1 and p_2 such that we can exit/enter p_1 and p_2 to go to the other side (and back again if needed). We need to enter/exit each path at most once. Figure 5a shows an example of such a path: Graph g_2 has an extra vertex after the bump on p_2, and enough space between the bumps on p_1 and p_2. A path starting at the bottom left can separate all bumps at the bottom, route back to the top left and separate all bumps at the top to end in the top right.

The graph G' now exactly represents all possible paths P can take. The answer to our original problem is thus reduced to whether there exists a path from s to t in G'.

▶ **Lemma 1.** *There exists a simple color-separating path P from s to t in a fully 2-colored grid G iff there exists a simple path from s to t in G'.*

More colors. In Lemma 1 we assumed that the grid contained only two colors. The same algorithm works using any number of colors, if we make some small adjustments. We construct graph g' to characterize P, and we use the fact that P should always follow the borders. However, if we have more than two colors, borders no longer have to be paths. Three/four faces with distinct colors can share a vertex v. Whenever this happens we know that there is no color-separating path, because the path would need to visit v more than once. We thus know that all borders are paths, and these borders can only be crossings or

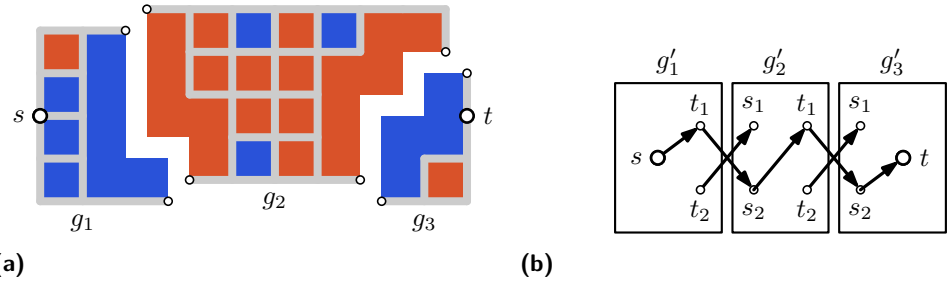


Figure 5 (a) Sequence of graphs created by removing crossings. (b) The graph that is created to decide whether there is a simple path that separates squares of different colors.

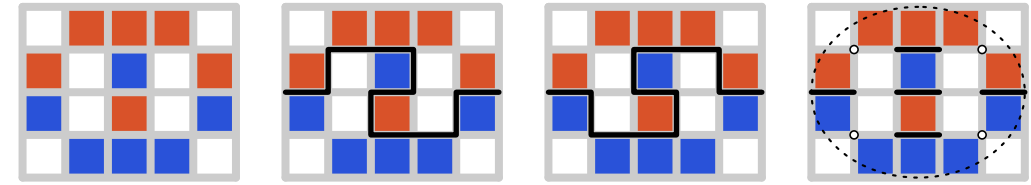


Figure 6 An egg. (a) The input. (b) One possible internal state of a path traversing the egg. (c) The other possible state. (d) The fixed edges and sockets of the egg. The egg outline is drawn in black, for easy recognition when used in later constructions.

bumps. We can therefore recolor the grid using only colors from $\{r, b\}$, such that on both sides of each crossing we get a different color. Assume that r is used to color cells between two crossings (or between s or t and a crossing) then b can be used to color the bumps (or vice versa). We can now apply Lemma 1 to find out whether there is a color separating path.

Running time. We can find the at most $O(wh)$ crossing borders in $O(wh)$ time by walking over p_1 and following the outline of every color to check if it hits p_2. As the total size of all pieces combined is bounded by $O(wh)$, we can thus create the graph G' in $O(wh)$. Finally we can determine if a path from s to t exists in the graph G' in $O(wh)$ with a simple DFS.

▶ **Theorem 2.** *Deciding whether there exists a simple color-separating path P from s to t in a fully-colored grid G can be done in $O(wh)$ time.*

3 Hardness

In this section we show that deciding whether there exists a simple color-separating path P from s to t in a grid G with 2 colors is NP-hard if the grid contains empty squares. To prove this we provide a reduction from planar rectilinear 3-SAT [5].

Eggs. Our reduction is built from several gadgets. Our most basic gadget consists of a 5×4 grid with 12 colored squares, which we call an *egg*. The gadget is illustrated in Figure 6a. Note that five edges of G inside the egg must be on P, because the adjacent cells are of different colors. There are then only two possible ways how P can traverse the egg (note that it is not possible to connect the edges to the boundary of the gadget using multiple paths), depicted in Figures 6b and 6c. Two pair vertices of G are used by only one of the two paths, we refer to those vertices as the *sockets* of the egg (see Figure 6d).

Variables. Variables are built by connecting many eggs together at their sockets: if one egg uses a socket, the adjacent egg cannot use the same socket. We define an *egg snake* of length

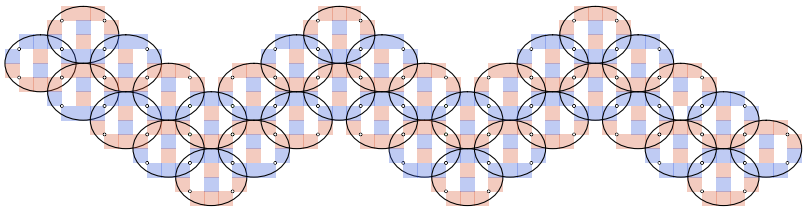


Figure 7 An egg snake of length 3.

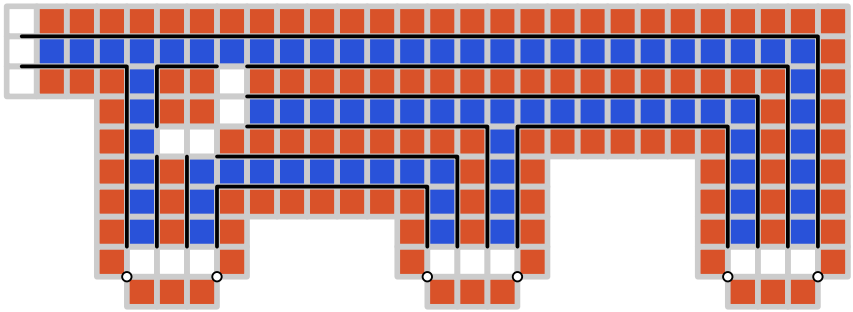


Figure 8 A clause comb. There are 6 sets of adjacent empty cells; one at the handle, two ornaments and three teeth.

k to be an arrangement of $12k - 2$ eggs, as depicted in Figure 7.

If x_i is `true`, then all top left and bottom right sockets (in the orientation shown in Figure 6d) of every egg are used, otherwise all bottom left and top right sockets of every egg are used. Note that, by the nature of their arrangement, all eggs in a snake must be in the same state. We globally place all variable egg snakes in a horizontal line; see Figure 9.

Clauses. We represent each clause of the SAT instance by a *comb* gadget, consisting of long strips of adjacent blue and red squares; see Figure 8. The exact shape of the comb is flexible: horizontal and vertical stretches can be made longer as required. The outside of the comb is covered by red squares, except for a single blue square in the top left which we refer to as the *handle* of the comb. Any color-separating path P must enter through the handle, collect (surround) all blue squares, and then leave again through the same gap.

There are five places in a comb where a choice can be made: three *teeth* and two *ornaments*, each can be found in Figure 8 as a set of adjacent empty cells. We can think of the choice to make as filling each tooth and ornament with either red or blue squares; this then fully determines the course of P. However, not all choices lead to valid paths: the two ornaments cannot both be set to blue because they would cause P to touch itself, and the left ornament and left tooth or the right ornament and right tooth cannot both be set to blue because this would create a red island. Similarly, we argue that exactly two out of the five teeth/ornaments must be blue and three must be red, otherwise there will be either a red or a blue island. It follows that at least one of the teeth must be blue.

Now, we observe that when a tooth is blue, it causes the path to run lower than when a tooth is red. We have a left and a right *socket* on each tooth of the comb, which we will let overlap with sockets of eggs. We connect a variable to the left socket for positive variables as the top right socket is not used in the case, and the right socket for negative variables for the same reason. Figure 9 shows a small example of an instance, leaving out most of the puzzle details, but showing how a color-separating path can be routed. The remaining space between variable and clause gadgets is filled by a grid of empty cells.

Satisfiability. We now show that we can efficiently find a solution to a planar 3-SAT formula

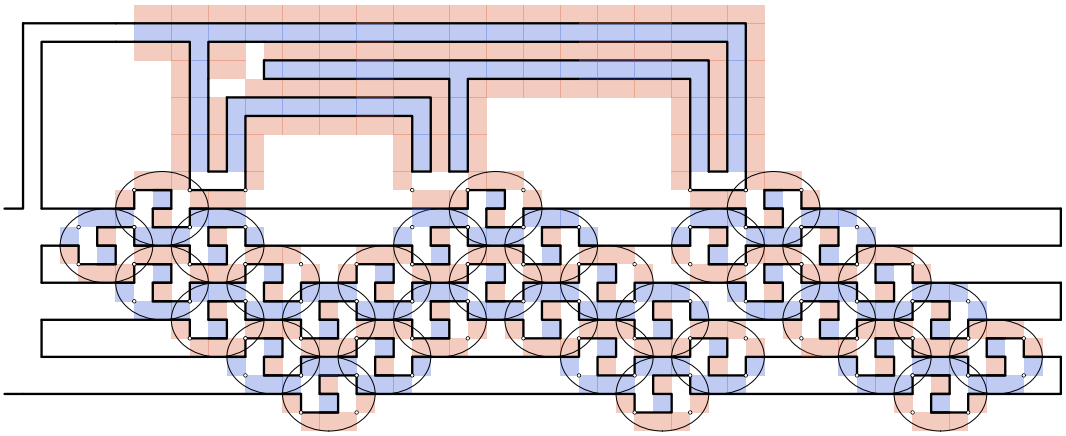


Figure 9 An example of a small satisfiable instance. The left egg snake represent the variable x and the right egg snake represent the variable y. The clause represent the statement $x \vee \neg x \vee \neg y$. The color-separating path shows us that setting x to true and y to false satisfies the clause.

if we can efficiently find a color-separating path in the constructed instance for such a formula.

To find a color-separating path, we thread six horizontal paths through all variables. The ends of the topmost and bottommost of these paths will be routed through the clause gadgets above and below the variable line respectively. Finally, we connect the six paths through the variables into a single path to create a color-separating path.

▶ **Theorem 3.** *Deciding whether there exists a simple color-separating path P from s to t in a grid G with two colors is NP-hard if the grid contains empty squares.*

References

1 The Witness. `http://the-witness.net/`, 2010. Accessed on 2018-01-07.

2 Quirijn W. Bouts, Irina Kostitsyna, Marc van Kreveld, Wouter Meulemans, Willem Sonke, and Kevin Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *Proc. 24th Annual European Symposium on Algorithms (ESA)*, pages 22:1–22:16, 2016.

3 Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In Michael H. Albert and Richard J. Nowakowski, editors, *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009.

4 Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

5 Donald E Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992.

6 Maarten Löffler and Wouter Meulemans. Discretized approaches to schematization. In *Proc. 29th Canadian Conference on Computational Geometry (CCCG)*, pages 220–225, 2017.

7 Nadia Magnenat-Thalmann and Daniel Thalmann. *New Trends in Computer Graphics*. Springer, 1st edition, 1988.

8 Edward Robertson and Ian Munro. NP-completeness, puzzles and games. *Utilitas Mathematica*, 13:99–116, 1978.

9 Takayuki Yato. On the NP-completeness of the Slither Link puzzle. *IPSJ SIGNotes ALgorithms*, 84(2000-AL-074):25–31, 2000. In Japanese.

Finding the Girth in Disk Graphs and a Directed Triangle in Transmission Graphs*

Haim Kaplan[1], Katharina Klost[2], Wolfgang Mulzer[2], and Liam Roditty[3]

1 **Tel Aviv University, Israel**
haimk@post.tau.ac.il
2 **Institut für Informatik, Freie Universität Berlin, Germany**
{kathklost,mulzer}@inf.fu-berlin.de
3 **Bar Ilan University, Israel**
liamr@macs.biu.ac.il

Abstract

Suppose we are given a set $S \subset \mathbb{R}^2$ of n point *sites* in the plane, each with an *associated radius* $r_s > 0$, for $s \in S$. The *disk graph* $D(S)$ for S is the undirected graph with vertex set S and an edge between s and t in S if and only if $|st| \leq r_s + r_t$, i.e., if the disks with radius r_s around s and with radius r_t around t intersect. The *transmission graph* $T(S)$ for S is the directed graph with vertex set S and an edge from s to t if and only if $|st| \leq r_s$, i.e., if the disk with radius r_s around s contains the site t.

We consider two problems concerning cycles in disk graphs and transmission graphs. First, we show that the *weighted girth* of a disk graph can be found in $O(n \log n)$ expected time, almost matching the bounds for planar graphs. Second, we present an algorithm for finding a *directed triangle* in a transmission graph in $O(n \log^2 n)$ time. Thus, these problems are much easier for disk and transmission graphs than for general graphs.

1 Introduction

Despite decades of research, many seemingly simple problems on graphs continue to stump researchers. For example, given a simple graph $G = (V, E)$, the best "combinatorial" algorithm to determine whether G contains a *triangle* (i.e., a cycle of length three) requires $O(n^3 \operatorname{polyloglog}(n)/\log^4 n)$ time [13], only a slight improvement over the trivial algorithm. Using fast matrix multiplication, the problem can be solved in $O(n^\omega)$ time, where $\omega < 2.37287$ is the matrix multiplication exponent [7, 8]. For planar graphs, the problem becomes much easier: here, the unweighted *girth* (i.e., the length of the shortest cycle) can be found in linear time [5].

Two interesting graph classes that invite further study are *disk graphs* and *transmission graphs*. In both cases, we are given a set $S \subset \mathbb{R}^2$ of n point *sites* in the plane. Each site $s \in S$ has an *associated radius* $r_s > 0$ and an *associated disk* D_s centered around s with radius r_s. The *disk intersection graph* $D(S)$ for S is the undirected graph on S where two sites $s, t \in S$ are adjacent if and only if their associated disks intersect, i.e., if $D_s \cap D_t \neq \emptyset$. The edges of $D(S)$ are weighted according to the euclidean distance of their endpoints. The *directed transmission graph* $T(S)$ for S is the directed graph on S where there is an edge from a site s to a site t if and only if $t \in D_s$. Both graphs are well studied in computational geometry, since they serve as simple theoretical models for geometric sensor networks (see [9] and the

* Supported in part by grant 1367/2016 from the German-Israeli Science Foundation (GIF). W.M. supported in part by ERC StG 757609.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

references therein). Previously, Kaplan *et al.* [10] have studied the girth and triangles in disk intersection graphs. They showed that for a disk intersection graph with n sites, one can compute the unweighted girth in $O(n \log n)$ deterministic time and that one can find a shortest triangle in $O(n \log n)$ expected time. The running time for the unweighted girth is optimal in the algebraic decision tree model [12]. We extend the results of Kaplan *et al.* [10] to the weighted girth in disk graphs and to the triangle problem in transmission graphs.

2 Weighted girth of a disk graph

In this section we consider the problem of finding the *weighted* girth of a disk intersection graph. First, we describe an algorithm that, given a vertex and an abstract graph with some restrictions, finds the shortest cycle in the graph containing that vertex. This algorithm is then used as a subroutine in Section 2.2 to compute the weighted girth of a disk intersection graph.

2.1 Finding the shortest cycle containing a given vertex

Let $G = (V, E)$ be an abstract graph with nonnegative edge weights, such that all shortest paths and all cycles in G have pairwise distinct lengths and such that for all edges $uv \in E$, the shortest path from u to v is the edge uv. Let $|V| = n$ and $|E| = m$. We present an algorithm that, given G and a vertex $s \in V$, computes a shortest cycle in G containing s. A simple randomized algorithm for this problem was presented by Yuster [14]. We give a deterministic algorithm.

We run Dijkstra's algorithm to determine the shortest path tree T for s in G in $O(n \log n + m)$ time. Then, we traverse T to find for each $v \in V$ the vertex $b[v] \in V$ that comes after s on the shortest path from s to v. This takes $O(n)$ steps. Finally, we iterate over all edges $e \in E$ that do not occur in T. For each such edge $e = uv$, we check if $b[u] \neq b[v]$. If this is the case, then e closes a cycle in T that contains s. We determine the length of this cycle in $O(1)$ time, using the shortest path distances and the length of e. We return the shortest such cycle. Overall, the algorithm requires $O(n \log n + m)$ time. The following lemma shows the shortest cycle in G that contains s is of the desired form.

▶ **Lemma 2.1.** *The shortest cycle in G that contains s consists of two paths in the shortest path tree T of s, and one additional edge.*

Proof. Let $C = (v_0 = s), v_1, v_2, \ldots, v_{\ell-1}, s$ be the shortest cycle in G containing s, where all vertices v_i are pairwise distinct and $\ell \geq 3$. For $v_i \in C$, let $d_1(v_i)$ be the length of the path $s, v_1, \ldots, v_i$, and let $d_2(v_i)$ be the length of the path $v_i, v_{i+1}, \ldots, s$. Let $\pi(v_i)$ denote the shortest path from s to v_i, and let $|v_i v_{i+1}|$ be the length of the edge $v_i v_{i+1}$.

Suppose that C is not of the desired from. Let v_k, v_{k+1} be the edge on C with $d_1(v_k) < |v_k v_{k+1}| + d_2(v_{k+1})$ and $d_2(v_{k+1}) < d_1(v_k) + |v_k v_{k+1}|$. By our assumptions on G, the edge $v_k v_{k+1}$ exists and $k \neq 0, \ell - 1$. We distinguish two cases.

First, suppose that $\pi(v_k) \cap \pi(v_{k+1}) = \{s\}$. Consider the cycle C' given by $\pi(v_k)$, the edge $v_k v_{k+1}$, and $\pi(v_{k+1})$. Since $s \neq v_k, v_{k+1}$ and since the edge $v_k v_{k+1}$ does not appear on $\pi(v_k)$ and $\pi(v_{k+1})$, it follows that C' is a proper cycle. Furthermore, by assumption, C' is strictly shorter than C, because $\pi(v_k)$ is shorter than $d_1(v_k)$ or $\pi(v_{k+1})$ is shorter than $d_2(v_{k+1})$. This contradicts our choice of C.

Second, suppose that $|\pi(v_k) \cap \pi(v_{k+1})| \geq 2$. Since $\pi(v_k)$ and $\pi(v_{k+1})$ are shortest paths, their intersection is a prefix of each path. By the assumption on G, at least one of $v_1, v_{\ell-1}$ is not in $\pi(v_k) \cup \pi(v_{k+1})$. Without loss of generality, this vertex is v_1. Let $j \geq 1$ be the

smallest index such that $v_j \in \pi(v_k) \cup \pi(v_{k+1})$. We have $j \in \{2, \ldots, k\}$. Consider the cycle C' that starts at at s, follows C along $v_1, v_2, \ldots$ up to v_j, and then returns along $\pi(v_k)$ or $\pi(v_{k+1})$ to s. By construction, C' is a proper cycle. Furthermore, $C' \neq C$, because even if $j = k$, the path $\pi(v_k)$ does not use the edge $v_k v_{k+1}$ due to the choice of k. Finally, C' is strictly shorter than C, because the second part of C' from v_j to s follows a shortest path and is thus strictly shorter than $d_2(v_j)$. Again, C' contradicts our choice of C. ◀

2.2 Computing the girth

We describe an algorithm to compute the weighted girth of a disk intersection graph $D(S)$. First, we find the shortest triangle in the disk graph $D(S)$. This takes $O(n \log n)$ expected time using the algorithm of Kaplan *et al.* [10].

If $D(S)$ contains no triangle, then it is plane [6] [10, Lemma 1]. Thus, we can explicitly construct $D(S)$ with a sweep line algorithm in time $O(n \log n)$ and determine the girth of this weighted graph with an appropriate algorithm for planar graphs.

If $D(S)$ contains a triangle, its length W can serve as an upper bound for the length of the shortest cycle in $D(S)$. We use the same partition of S into *large* and *small* sites as Kaplan *et al.* [10]. Namely, we set $\ell = W/12\sqrt{2}$, and we call all sites with radius at least ℓ *large* and the remaining sites *small*. Still following Kaplan *et al.*, we cover the plane with four overlapping axis parallel grids G_1, G_2, G_3, and G_4. The *open* grid cells have side length 4ℓ, and the grids are defined such that the points $(0, 0)$, $(2\ell, 0)$, $(0, 2\ell)$ and $(2\ell, 2\ell)$ are vertices of G_1, G_2, G_3, and G_4, respectively.

We want to find the shortest cycle with at least four vertices and with length at most W. First, we consider cycles that consist only of small sites. From the choice of ℓ, it follows that there is no triangle consisting only of small sites: otherwise, there would be a triangle of length at most $3 \cdot 4\ell < W$, contradicting the choice of W. Thus, the subgraph D' of $D(S)$ induced by the small vertices is plane [6] [10, Lemma 1]. As before, we can compute D' and its girth directly, using a plane sweep and known results for planar graphs. Let Δ_1 be this girth.

Finally, we consider cycles with at least one large site. By the choice of ℓ, every triangle that is completely contained in an open grid cell has length less than W. Since there are no such triangles in $D(S)$, we can apply Lemma 6 of Kaplan *et al.* [10] to conclude that each grid cell contains $O(1)$ large sites.

By the triangle inequality, in a cycle of length less than W, the maximum distance between any two sites is less than $W/2$. Thus, any such cycle containing a given site $s \in S$ completely lies in a rectangle with side length W around s. This corresponds to a 7×7 neighborhood $N(\sigma)$ around a grid cell σ containing s. Since $N(\sigma)$ consists of $O(1)$ cells and since each cell contains $O(1)$ large sites, there are $O(1)$ large sites in $N(\sigma)$.

We iterate over all grid cells σ. For each σ, we consider all large sites $s \in \sigma$. As discussed, we must find the shortest cycle containing s in the subgraph $D(S_\sigma)$ of $D(S)$ induced by the sites $S_\sigma = S \cap N(\sigma)$. Suppose $D(S_\sigma)$ contains n'_σ small sites and n''_σ large sites. Since the graph induced by the small sites is plane and since $n''_\sigma = O(1)$, the graph $D(S_\sigma)$ has $O(n_\sigma)$ edges. This means that we can explicitly compute $D(S_\sigma)$ in time $O(n_\sigma \log n_\sigma)$ and apply the algorithm from Section 2.1 in order to compute the shortest cycle containing s in time $O(n_\sigma \log n_\sigma)$. Let Δ_2 be the length of the shortest cycle encountered in this step. If we also want to output the shortest cycle in the end, we also store a pointer to σ and s. Since each small site is involved only in a constant number of neighborhoods, we have: $\sum_{i=1}^{4} \sum_{\sigma \in G_i} n_\sigma = O(n)$, and thus the overall running time of this step is $O(n \log n)$. In the end, we return $\min\{W, \Delta_1, \Delta_2\}$. Thus, we obtain the following theorem:

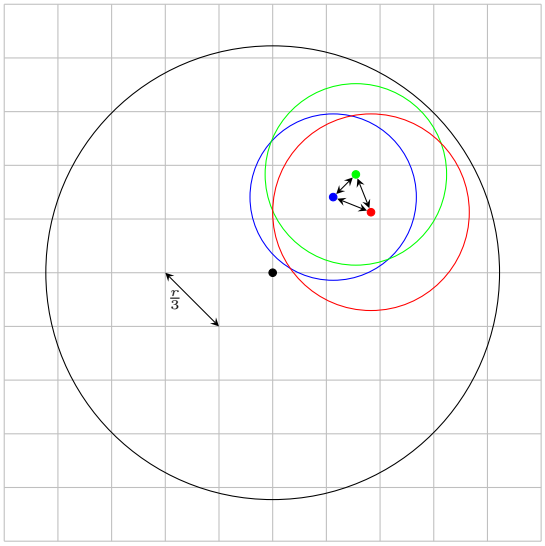


Figure 1 Three disks with associated radius at least $r/3$ are in the same grid cell form a clique

▶ **Theorem 2.2.** *Given a set S of n point sites in $\mathbb{R}^2$ with associated radii, we can compute the weighted girth of $D(S)$ in $P(n) + O(n \log n)$ expected time, where $P(n)$ is the time needed to compute the weighted girth of a planar graph with real edge weights.*

▶ **Corollary 2.3.** *Using the algorithm of Łącki and Sankowski [11], we can compute the weighted girth of a disk graph in $O(n \log n)$ expected time.*

3 Directed triangles in transmission graphs

In this section we consider directed triangles in transmission graphs. Given a disk transmission graph $T(S)$ we want to decide, if this graph contains at least one directed triangle.

First we consider the following structural lemma. It gives a condition on the disks that will help us find certain triangles.

▶ **Lemma 3.1.** *Let D be a disk of radius r. If D contains more than 152 sites with associated radius at least $r/3$, then $T(S)$ has a directed triangle.*

Proof. We cover D with a grid, where each cell has diameter $r/3$. Each grid cell has side length $\sqrt{2}r/6$, so we need at most 76 such cells (see Figure 1). By our choice of the diameter, for each site $s \in D$ with $r_s \geq r/3$, the associated disk D_s completely covers the grid cell that contains s.

If D contains more than 152 sites with associated radius at least $r/3$, the pigeonhole principle shows that one grid cell contains at least three such sites. Since the corresponding disks contain the complete grid cell, these three sites form a directed clique in $T(S)$. In particular, there is a directed triangle. ◀

Now we show how the condition of Lemma 3.1 can be checked for a given disk transmission graph. This will later be the first part of the algorithm to find a triangle.

▶ **Lemma 3.2.** *In $O(n \log^2 n)$ time, we can check whether S contains a site s such that D_s contains more than 152 sites with associated radius at least $r_s/3$. Furthermore, if every disk contains at most 152 such sites, we can find all these sites in $O(n \log^2 n)$ time.*

Proof. We use the halfspace range reporting structure by Afshani and Chan [1]. This structure allows us to preproccess a planar n-point set $P \subset \mathbb{R}^2$ in $O(n \log n)$ time so that for any query point $q \in \mathbb{R}^2$ and for any $k \in \{1, \ldots, n\}$, we can find the k nearest neighbors of q in time $O(\log n + k)$ [4]. We will actually need a semi-dynamic version of this data structure that supports insertions. For this, we apply the classic Bentley-Saxe transform to obtain a structure with $O(\log n)$ amortized insertion time and $O(\log^2 n + k \log n)$ worst-case query time [3].

We consider the sites by decreasing radius. Our range reporting data structure will always contain all sites with associated radius at least $r_s/3$, where s is the current site. When processing $s \in S$, we first insert all sites with radius at least $r_s/3$ that are not yet present in the data structure. Then, we query the 153 nearest neighbors of s in the structure, and we determine which of them lie in D_s. If all of them do, then $T(S)$ contains a triangle. Otherwise, we store this set with s. One such query takes $O(\log^2 n)$ time, for a total of $O(n \log^2 n)$ time. The total time to sort the sites by descending radius and for inserting them into the structure is $O(n \log n)$. The claim follows. ◀

With Lemma 3.2 we now know how to check if a graph contains a triangle because of the condition of Lemma 3.1. Furthermore Lemma 3.2 allows us to find for each site s all sites with radius at least $r_s/3$, contained in D_s. In the next lemma we show how, given this information, we can find a triangle in a transmission graph were no disk obeys the condition of Lemma 3.1.

▶ **Lemma 3.3.** *Suppose we are given a set S of n sites such that for each $s \in S$, the disk D_s contains at most 152 sites with associated radius at least $r_s/3$ and such that these sites are known. We can find a directed triangle in $T(S)$ in $O(n \log^2 n)$ time, if it exists.*

Proof. We need a static nearest neighbor data structure for the additively weighted euclidean distance. Using an appropriate Voronoi diagram, this can be done with $O(n \log n)$ preprocessing time and $O(\log n)$ query time [2]. We will have queries of the following form: given a query point $q \in \mathbb{R}^2$, find the nearest site to q whose radius lies in a given interval. For this, we build a perfect binary search tree on S, sorted by radius. In each inner vertex v of the tree, we store an additively weighted Voronoi diagram for all disks in the subtree of v. The weight for each site s is $-r_s$.

This tree can be constructed in $O(n \log^2 n)$ time in bottom up fashion. Given a query point q and a radius range (r, r'), we must perform $O(\log n)$ queries to the Voronoi diagrams, since we can follow the paths to r and r' and query all the diagrams of tree vertices whose intervals are completely contained in (r, r'). Thus, the query time is $O(\log^2 n)$.

We iterate over the sites by decreasing radius. We will check for each site $s \in S$ if it is the site with smallest radius in a directed triangle in $T(S)$. Suppose there is such a triangle of the form $s \to t \to u \to s$. Thus, we have $r_s \leq r_t$ and $r_s \leq r_u$. Since $t \in D_s$, there are at most 152 known candidates for t. Having fixed such a candidate t, there are two cases regarding u:

1. $r_u \geq r_t/3$: in this case, having fixed t, there are only 152 known candidates for u, and all of them can be checked in $O(1)$ time.
2. $r_u < r_t/3$: by definition, we have $s \in D_u$. From this, it follows that that $D_u \subset D_t$. Thus, to find a triangle of the desired kind, it is enough to find any site u with $r_u < r_t/3$ and

with $s \in D_u$. This can be done by finding the nearest site to s with radius in $(r_s, r_t/3)$. As explained, this takes $O(\log^2 n)$ time.

Since we iterate over all sites, this results in a total running time of $O(n \log^2 n)$. ◀

Now we can combine the Lemma 3.2 and Lemma 3.3 to get the following theorem:

▶ **Theorem 3.4.** *Given a set S of n point sites in $\mathbb{R}^2$ with associated radii, we can find a directed triangle in the associated directed transmission graph $T(S)$ in time $O(n \log^2 n)$.*

Proof. First we use the procedure described in Lemma 3.2 in time $O(n \log^2 n)$. If it finds a triangle, we return yes. Otherwise we use the resulting information, to apply the algorithm from Lemma 3.3. This results in an algorithm with $O(n \log^2 n)$ running time. ◀

References

1 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 180–186, 2009.

2 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing, 2013.

3 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

4 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comput. Geom.*, 56(4):866–881, 2016.

5 Hsien-Chih Chang and Hsueh-I Lu. Computing the girth of a planar graph in linear time. *SIAM J. Comput.*, 42(3):1077–1094, 2013.

6 William S. Evans, Mereke van Garderen, Maarten Löffler, and Valentin Polishchuk. Recognizing a DOG is hard, but not when it is thin and unit. In *Proc. 8thInternat. Conf. Fun w. Algorithms (FUN)*, pages 16:1–16:12, 2016.

7 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th Internat. Symp. Symbolic and Algebraic Comput. (ISSAC)*, pages 296–303, 2014.

8 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

9 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and reachability oracles for directed transmission graphs. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 156–170, 2015.

10 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Finding triangles and computing the girth in disk graphs. In *Proc. 33rd European Workshop Comput. Geom. (EWCG)*, pages 205–208, 2017.

11 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In *Proc. 19th Annu. European Sympos. Algorithms (ESA)*, pages 155–166, 2011.

12 Valentin Polishchuk. Personal communication. 2017.

13 Huacheng Yu. An improved combinatorial algorithm for Boolean matrix multiplication. In *Proc. 42nd Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 1094–1105, 2015.

14 Raphael Yuster. A shortest cycle for each vertex of a graph. *Inform. Process. Lett.*, 111(21-22):1057–1061, 2011.

Lower bounds for coloring of the plane

Konstanty Junosza-Szaniawski and Krzysztof Węsek

Warsaw University of Technology, Poland
{k.szaniawski,k.wesek}@mini.pw.edu.pl

Abstract

Let $G_{[1,b]}$ be the graph with the set of vertices $\mathbb{R}^2$ and adjacency between points at distance in the set $[1, b]$. We obtain new lower bounds for $\chi(G_{[1,b]})$ for certain values of b. Combined with known upper bounds, this result gives two intervals of values of b for which we exactly determine $\chi(G_{[1,b]})$ to be 7 and 9, respectively. The first interval contains and substantially enlarges the only known set of values of b with determined $\chi(G_{[1,b]})$ coming from the work of Exoo (2004).

1 Introduction

1.1 Background

Probably the most known and challenging question of geometric graph theory is the Hadwiger-Nelson problem. It asks for the minimal number of colors in a coloring of the Euclidean plane $\mathbb{R}^2$ with a restriction that any two points at distance 1 obtain distinct colors. In other words: what is the chromatic number of the unit distance graph of $\mathbb{R}^2$ (a graph defined on the set of vertices $\mathbb{R}^2$ and with edges between vertices at distance 1)? The problem was first posed by Edward Nelson in 1950 and made known to the mathematical society by Hugo Hadwiger. As soon as in 1950's, Nelson and John Isbell were first to prove that the answer is at least 4 and at most 7, respectively. Unfortunately, after several decades, these classic bounds are still the best known in general. Although far from being solved, Hadwiger-Nelson problem inspired a vast number of challenging questions, interesting results and applications in the intersection of combinatorics and geometry. For more information on the history of the problem and selected related problems, we refer to the book of Soifer [9].

One of possible directions in order to obtain more understanding of such problems is to consider a more general set of restricting distances. Let $G_{[a,b]}$ denote the graph with the set of vertices $\mathbb{R}^2$ and two vertices adjacent if they are at distance from the interval $[a, b]$. However, by scaling, we can assume that $a = 1$. In this paper, we will consider these graphs.

Some important results on coloring of such graphs were presented by Exoo [2] (using slightly different notation). In particular, he showed the following theorem.

▶ **Theorem 1.1** ([2]).
For $b \in (\sqrt{43}/5, \sqrt{7}/2] \approx (1.31149, 1.32287]$ *there holds* $\chi(G_{[1,b]}) = 7$.

To our knowledge, the interval given in Theorem 1.1 was the only known set of values of b such that $\chi(G_{[1,b]})$ was determined. We note that the real contribution of Theorem 1.1 lays in establishing the lower bound for $\chi(G_{[1,b]})$. The upper bound comes from the observation that the well known 7-coloring of $G_{[1,1]}$ based on hexagonal tiling is proper also for $G_{[1,b]}$ for any $b \leq \sqrt{7}/2$. Exoo provided also a small $\delta > 0$ such that for any $b > 1 + \delta$, it holds that $\chi(G_{[1,b]}) \geq 5$. Later, it was published in [3] that the latter statement can be strengthened to any $b > 1$. However, it appears that before the mentioned two papers, the last result was already surpassed by two independent works. In a series of papers by Brown, Dunfield, Perry [6–8], among other results, the authors gave an elegant proof by Dunfield that for any $b > 1$ we have $\chi(G_{[1,b]}) \geq 6$. The proof is based on a result by Woodall (incorrect proof [12]) and Townsend (correct proof based on similar idea, see [10, 11]). Without giving the precise

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

statement, the Woodall-Townsend theorem can be expressed in the following way: if the unit distance graph of the plane is colored with the condition that color classes are defined with Jordan curves, then at least 6 colors are necessary. The key ingredient of the proof of the Woodall-Townsend theorem, very roughly speaking, is to find a point in the plane, which has at least 3 colors in any ε-neighborhood. A related idea was used by Currie and Eggleton in their manuscript [5], where they independently prove the same result as Dunfield. Although the manuscript was not properly published, it was already mentioned by Currie in his other paper published in 1992 [1]. Currie and Eggelton consider a coloring of $G_{[1,b]}$ and then for $\varepsilon \in \big(0, (b-1)/2\big)$ find a point x for which the closed ε-ball centered at x contains at least 3 colors. Then they prove that the annulus $\{p \in \mathbb{R}^2 : 1+\varepsilon \leq dist(p,x) \leq b-\varepsilon\}$ needs at least 3 colors and observe that it cannot use any of the colors from the closed ε-ball centered at x. This ends the proof.

1.2 Our approach

It seems that the idea of a point close to at least 3 colors was not exploited for larger values of b. In our work, we use this concept to provide new lower bounds for $\chi(G_{[1,b]})$ for certain values of $b > 1$. The approach consists of two steps. First, we use the mentioned fact that any coloring of $G_{[1,b]}$ for any $b > 1$ and any sufficiently small $\varepsilon > 0$ admits a closed ε-ball centered at some point x containing at least 3 colors. We give a new proof of this statement. As the second step, we consider the annulus $A_{b,\varepsilon}$ centered at x with the inner radius $1+\varepsilon$ and the outer radius $b-\varepsilon$. Clearly, none of at least 3 colors found in the closed ε-ball centered at x can be used in $A_{b,\varepsilon}$. If for some k we are able to prove that $A_{b,\varepsilon}$ itself requires at least k colors, then we obtain $\chi(G_{[1,b]}) \geq k+3$.

In order to show a lower bound for coloring of $G_{[1,b]}$ or a subset of $G_{[1,b]}$, one may try to construct a finite subset for which finite graph coloring techniques can be applied. For example Exoo, in order to prove the lower bound in Theorem 1.1, considered coloring of a finite part P of a carefully chosen regular triangular grid. Using computer aided calculations, he showed that for the specified range of b the subgraph of $G_{[1,b]}$ induced by P requires at least 7 colors. However, it is unlikely that his choice of parameters for the grid is optimal (in terms of range of b), as it is limited by the computer computational power. In order to show a lower bound for coloring of $A_{b,\varepsilon}$, we also construct a certain finite subset of it. Our analysis suggested that it is reasonable to consider sets created by taking a number of points regularly placed on a small number of circles of radius chosen between $1+\varepsilon$ and $b-\varepsilon$.

The benefit of this approach is that we reduce the search for finite configurations to a relatively small part of the plane. On the other hand, it is likely that for many values of b the chromatic number of the subgraph of $G_{[1,b]}$ induced by $A_{b,\varepsilon}$ plus 3 is strictly smaller than $\chi(G_{[1,b]})$. However, this plan proves itself to be effective in providing a new contribution, as we were able to determine $\chi(G_{[1,b]})$ for two intervals of values of b. In particular, we improve the important part of Theorem 1.1 in the meaning that we give a more general lower bound.

Our lower bounds on the number of colors for finite configurations are obtained by computer-based computations. We used a slightly modified standard mixed integer programming formulation of graph coloring and solved the models using IBM ILOG CPLEX solver (version 12.7.1).

2 The results

We start with a key lemma already proved in [5]. However, we give a new, shorter proof of this fact.

▶ **Lemma 2.1** ([5]; a new proof)**.**
Let c be a proper coloring of $G_{[1,b]}$ for $b > 1$. Consider any $\varepsilon > 0$ satisfying $b - 1 > \varepsilon$. Then there exists a point x in $\mathbb{R}^2$ such that in the closed ε-ball centered in x there are at least 3 colors (with respect to c).

Proof. For a monochromatic set A colored with c_1 and $X \subseteq \mathbb{R}^2$ denote by $S(A, X)$ the set of all c_1-colored points in X that can be obtained from A by a sequence of c_1-colored points belonging to X with consecutive distances at most ε. For $S, X \subseteq \mathbb{R}^2$, let $H_\varepsilon(S, X) = \{p \in X : \exists_{s \in S}\ dist(p, s) \leq \varepsilon\}$. In the proof, we will use the following observation.

(∗) If S is a bounded, connected set and X contains the unbounded component of $\mathbb{R}^2 \setminus S$, then there exists a simple closed curve C in $H_\varepsilon(S, X) \setminus S$ so that all points from S are inside C.

For a simple closed curve C, let $In(C)$ be the bounded component of $\mathbb{R}^2 \setminus C$.

Suppose to the contrary to the thesis, that there is no such point x. Let $X_1 = \mathbb{R}^2$. Take any point $y \in \mathbb{R}^2$. Set $S_1 = S(\{y\}, X_1)$. Note that S_1 is a bounded set. Otherwise it would contain a sequence of points of the same color with consecutive distances less than ε and realizing arbitrarily large distance. Hence S_1 would contain a pair of points at distance in $(1, b)$, a contradiction. Since $H_{\varepsilon/2}(S_1, X_1)$ is bounded and connected, by $(*)$ we can find a simple closed curve C_1 in $H_\varepsilon(S_1, X_1) \setminus H_{\varepsilon/2}(S_1, X_1) = H_{\varepsilon/2}(H_{\varepsilon/2}(S_1, X_1), X_1) \setminus H_{\varepsilon/2}(S_1, X_1)$ so that all points from $H_{\varepsilon/2}(S_1, X_1)$ are inside C_1. Observe that all points of C_1 have the same color, as otherwise we would have a closed ε-ball with 3 colors. We continue the construction for $i > 1$ in the following way. We set $X_i = X_{i-1} \setminus In(C_{i-1})$, $S_i = S(C_{i-1}, X_i)$. Since $H_{\varepsilon/2}(S_i, X_i)$ is bounded and connected, by $(*)$ we can find a simple closed curve C_i in $H_\varepsilon(S_i, X_i) \setminus H_{\varepsilon/2}(S_i, X_i) = H_{\varepsilon/2}(H_{\varepsilon/2}(S_i, X_i), X_i) \setminus H_{\varepsilon/2}(S_i, X_i)$ so that all points from $H_{\varepsilon/2}(S_i, X_i)$ are inside C_i. Again, all points of C_i have the same color, as otherwise we would have a closed ε-ball with 3 colors.

We claim that $diam(C_i) - diam(C_{i-1}) \geq \varepsilon$ for $i > 1$. Consider two points y_1, y_2 that realize $diam(C_{i-1})$ and take the line ℓ containing y_1, y_2. Let y_1', y_2' be the points from ℓ satisfying $dist(y_1', y_1) = \varepsilon/2$, $dist(y_2', y_2) = \varepsilon/2$ and $dist(y_1', y_2') = dist(y_1, y_2) + \varepsilon$. Clearly, $y_1', y_2' \in H_{\varepsilon/2}(S_i, X_i) \subseteq In(C_i)$ and hence $diam(C_i) \geq dist(y_1', y_2')$, as claimed. Thus $diam(C_i) - diam(C_{i-1}) \geq \varepsilon$. Therefore for sufficiently large i we have $diam(C_i) > 1$ and there are two points in C_i at distance from $(1, b)$. On the other hand, all points C_i have the same color, which contradicts with the fact that c is a proper coloring of $G_{[1,b]}$. ◀

▶ **Theorem 2.2.** *The following inequalities hold:*

1. $\chi(G_{[1,b]}) \geq 7$ *for* $b > \sqrt{2 - 2\sin(\frac{18\pi}{325})} \approx 1.28599$
2. $\chi(G_{[1,b]}) \geq 8$ *for* $b > \sqrt{2 + 2\sin(\frac{\pi}{38})} \approx 1.47145$
3. $\chi(G_{[1,b]}) \geq 9$ *for* $b > \sqrt{2 + 2\sin(\frac{7\pi}{45})} \approx 1.71433$

Proof. Consider $b > 1$ and a proper coloring c of $G_{[1,b]}$ with $\chi(G_{[1,b]})$ colors, say $1, \ldots, \chi(G_{[1,b]})$. Fix $\varepsilon > 0$. Let x be a point such that in the closed ε-ball centered at x there are at least 3 colors with respect to c, say colors $1, 2, 3$. Without loss of generality we can assume that $x = (0, 0)$. Then no point in the annulus $A_{b,\varepsilon} = \{p \in \mathbb{R}^2 : 1 + \varepsilon \leq dist(p, \mathbf{0}) \leq b - \varepsilon\}$ can be colored with any of the colors $1, 2, 3$.

The general outline is that in each case for a fixed b, we will construct a finite subset of $A_{b,\varepsilon}$ (for sufficiently small ε) such that it needs at least k colors, for some k. In other words, we will find a subgraph of $G_{[1,b]}$ consisting of vertices from $A_{b,\varepsilon}$ forcing k colors. This will imply that $\chi(G_{[1,b]}) \geq 3 + k$. Denote by X_r^n the set consisting of n points evenly

distributed on the circle with the center in $(0,0)$ and radius r so that one point lays in the set $\{0\} \times (0, +\infty)$.

1. Assume that $b > \sqrt{2-2\sin(\frac{18\pi}{325})}$. Consider $Y_\varepsilon = X^{1300}_{1+\varepsilon} \cup X^{1300}_{b-\varepsilon} \subseteq A_{b,\varepsilon}$. It can be checked that for sufficiently small $\varepsilon > 0$, any proper coloring of the graph induced by Y_ε in $G_{[1,b]}$ uses at least 4 colors.
2. Assume that $b > \sqrt{2+2\sin(\frac{\pi}{38})}$. Consider $Y_\varepsilon = X^{190}_{1+\varepsilon} \cup X^{190}_{b-\varepsilon} \subseteq A_{b,\varepsilon}$. It can be checked that for sufficiently small $\varepsilon > 0$, any proper coloring of the graph induced by Y_ε in $G_{[1,b]}$ uses at least 5 colors.
3. Assume that $b > \sqrt{2+2\sin(\frac{7\pi}{45})}$. Consider $Y_\varepsilon = X^{180}_{1+\varepsilon} \cup X^{180}_{(1+b)/2} \cup X^{180}_{b-\varepsilon} \subseteq A_{b,\varepsilon}$. It can be checked that for sufficiently small $\varepsilon > 0$, any proper coloring of the graph induced by Y_ε in $G_{[1,b]}$ uses at least 6 colors.

◀

The exact right-hand side values in three inequalities on b in Theorem 2.2 are the optimal values for which the given finite configurations of points posses the desired chromatic properties. That is, in each case for any smaller value of b, the given set Y_ε can be colored with fewer colors than stated in the theorem. Nevertheless, we are far from claiming optimality of the given sets. In Theorem 2.2 we simply present the best constructions that we were able to find and verify. We expect that there exist sets of similar form which work for (possibly only slightly) smaller values of b.

By combining Theorem 2.2 with previously known bounds, we can obtain two intervals of values of b for which the chromatic number can be determined. Namely, let us use that Exoo [2] observed that $\chi(G_{[1,b]}) \leq 7$ for $b \leq \sqrt{7}/2$ and Ivanov [4] showed that $\chi(G_{[1,b]}) \leq 9$ for $b \leq \sqrt{3}$.

▶ **Corollary 2.3.** *For* $b \in \left(\sqrt{2-2\sin(\frac{18\pi}{325})}, \sqrt{7}/2\right] \approx (1.28599, 1.32287]$ *it holds* $\chi(G_{[1,b]}) = 7$.

▶ **Corollary 2.4.** *For* $b \in \left(\sqrt{2+2\sin(\frac{7\pi}{45})}, \sqrt{3}\right] \approx (1.71433, 1.73205]$ *it holds* $\chi(G_{[1,b]}) = 9$.

Note that the first interval contains and substantially enlarges the interval obtained by Exoo in Theorem 1.1. Moreover, the second interval was not known at all.

3 Conclusions

We note that our method combines a theoretical reasoning of continuous nature and constructions of finite sets for which the coloring properties are checked by computer. Therefore, the approach differs from the previously used in the literature. Similar constructions for larger values of b and larger number of colors are in preparation. However, it seems that the method should work better for relatively small values of b (and hence small number of colors), as in this case the 3 colors reserved by the ε-ball make a greater difference.

One may observe that we do not have any interval with the chromatic number determined to 8 colors. The reason is that we do not have a good 8-coloring of the plane. That is, an 8-coloring of $G_{[1,b]}$ that would work for b substantially larger than the known 7-colorings. It would be interesting to obtain an 8-coloring of $G_{[1,b]}$ even for $b > 1.4$.

References

1 J.D. Currie. Connectivity of distance graphs. *Discrete Mathematics*, (1):91 – 94, 1992.
2 G. Exoo. ε-unit distance graphs. *Discrete & Computational Geometry*, 33:117–123, 2005.

3 Jaroslaw Grytczuk, Konstanty Junosza-Szaniawski, Joanna Sokól, and Krzysztof Wesek. Fractional and j-fold coloring of the plane. *Discrete & Computational Geometry*, 55:594–609, 2016.

4 L. L. Ivanov. On the chromatic numbers of $\mathbb{R}^2$ and $\mathbb{R}^3$ with intervals of forbiden distances. *Electronic Notes in Discrete Mathematics*, 29:159–162, 2007.

5 Roger B. Eggleton James D. Currie. Chromatic properties of the euclidean plane. preprint. `arXiv:1509.03667`.

6 N. Brown N. Dunfield and G. Perry. Colorings of the plane i. *Geombinatorics*, III:24–31, 1993.

7 N. Brown N. Dunfield and G. Perry. Colorings of the plane ii. *Geombinatorics*, III:64–74, 1994.

8 N. Brown N. Dunfield and G. Perry. Colorings of the plane iii. *Geombinatorics*, III:110–114, 1994.

9 Alexander Soifer. *The Mathematical Coloring Book: Mathematics of Coloring and the Colorful Life of its Creators*. Springer-Verlag New York, 2009.

10 S. P. Townsend. Every 5-colouring map in the plane contains a monochrome unit. *Journal of Combinatorial Theory, Series A*, 30:114 – 115, 1979.

11 S. P. Townsend. Colouring the plane with no monochrome unit. *Geombinatorics*, XIV:181 – 193, 2005.

12 D. R Woodall. Distances realized by sets covering the plane. *Journal of Combinatorial Theory, Series A*, 14:187 – 200, 1973.

Bottleneck Bichromatic Non-crossing Matchings using Orbits*

Marko Savić[1] and Miloš Stojakovic[2]

1,2 Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia.
{marko.savic, milos.stojakovic}@dmi.uns.ac.rs

Abstract

Given a set of n red and n blue points in the plane, we are interested in matching red points with blue points by straight line segments so that the segments do not cross. Bottleneck matching is such a matching that minimizes the length of the longest segment. We develop tools which enable us to solve the problem of finding bottleneck matchings of points in convex position in $O(n^2)$ time. We use the same approach to design an $O(n)$-time algorithm for the case where all points lie on a circle. Previously best known results were $O(n^3)$ for points in convex position, and $O(n \log n)$ for points on a circle.

1 Introduction

Let R and B be sets of n red and n blue points in the plane, respectively, with $P = R \cup B$. Let M be a perfect matching between points from R and B, using n straight line segments to match the points, that is, each point is an endpoint of exactly one line segment, and each line segment has one red and one blue endpoint. We forbid line segments to cross. The length of a longest line segment in M is called the *value* of M. Our goal is to find a matching under given constraints with the minimum value. Any such matching is called a *bottleneck matching* of P.

1.1 Related work

Monochromatic case The monochromatic variant of the problem is the case when points are not assigned colors, and any two points are allowed to be matched. The problem of computing bottleneck monochromatic non-crossing matching of a point set is shown to be NP-complete by Abu-Affash, Carmi, Katz and Trablesi in [2]. They also proved that it does not allow a PTAS, gave a $2\sqrt{10}$ factor approximation algorithm, and showed that the case where all points are in convex position can be solved exactly in $O(n^3)$ time. We improved this result in [6] by constructing $O(n^2)$-time algorithm. In [1], Abu-Affash, Biniaz, Carmi, Maheshwari and Smid presented an algorithm for computing a bottleneck monochromatic non-crossing matching of size at least $n/5$ in $O(n \log^2 n)$ time. They extended the same approach to provide an $O(n \log n)$-time approximation algorithm which computes a plane matching of size at least $2n/5$ whose edges have length at most $\sqrt{2} + \sqrt{3}$ times the length of the longest edge in a non-crossing bottleneck matching.

Bichromatic case The problem of finding a bottleneck bichromatic non-crossing matching (BBNCM) was proved to be NP-complete by Carlson, Armbruster, Bellam and Saladi in [4].

* Partly supported by Ministry of Education and Science, Republic of Serbia, and Provincial Secretariat for Science, Province of Vojvodina.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

But for the version where crossings are allowed, Efrat, Itai and Katz showed in [5] that a bottleneck matching between two point sets can be found in $O(n^{3/2} \log n)$ time.

Biniaz, Maheshwari and Smid in [3] studied special cases of BBNCMs. They showed that the case where all points are in convex position can be solved in $O(n^3)$ time, utilizing an algorithm similar to the one for monochromatic case presented in [2]. They also considered the case where the points of one color lie on a line and all points of the other color are on the same side of that line, providing an $O(n^4)$ algorithm to solve it. The same results for these special cases are independently obtained in [4]. In [3], an even more restricted problem is studied, a case where all points lie on a circle, for which an $O(n \log n)$-time algorithm is provided.

1.2 Our results

Here, we develop tools which enable us to solve the problem of finding a BBNCM of points in convex position in $O(n^2)$ time. Also, using the same toolset we design an optimal $O(n)$ algorithm for the case when the points lie on a circle.

Some important structural properties of BBNCMs of points in convex position that we aim to exploit are captured well by the concept of (what we refer to as) *orbit*. Informally speaking, orbits form a partition of the point set that turns out to have the following property – two differently colored points can be connected by a segment in some non-crossing matching if and only if they belong to to the same orbit.

As it turns out, there is a number of additional properties of orbits that we can put to good use, and once we combine them with ideas used to efficiently solve the monochromatic case in [6], we are able to get a considerable improvement of the algorithm running time, both in the convex case and in the case where all points lie on a circle.

For detailed exposition of our results and all the proofs, please refer to [7].

2 Orbits

In what follows we consider the case where all points of P are in convex position, i.e. they are the vertices of a convex polygon. Here we only deal with matchings without crossings, so from now on, the word matching is used to refer only to pairings that are crossing-free.

Let us label the points $v_0, v_1, \ldots, v_{2n-1}$ in positive (counterclockwise) direction. To simplify the notation, we will often use only the indices when referring to the vertices. We write $\{i, \ldots, j\}$ to represent the sequence $i, i+1, i+2, \ldots, j-1, j$. All operations are calculated modulo $2n$; note that i is not necessarily less than j, and that $\{i, \ldots, j\}$ is not the same as $\{j, \ldots, i\}$.

We say that (i, j) is a *feasible* pair if there exists a matching containing (i, j). It can be shown that every set of n blue and n red points in a convex position can be perfectly matched, implying that a pair (i, j) is feasible iff i and j are of different colors and each of $\{i+1, \ldots, j-1\}$ and $\{j+1, \ldots, i-1\}$ contains the same number of red and blue points.

Let $o(i)$ be the first point j starting from i in positive direction such that (i, j) is feasible. The function o has an inverse, denoted by o^{-1}, and it is easy to see that $o^{-1}(j)$ is the first point i starting from j in the negative (clockwise) direction such that (i, j) is feasible. We define $o^0(i) := i$.

▶ **Definition 2.1.** An *orbit* of i, denoted by $\mathcal{O}(i)$, is defined by $\mathcal{O}(i) := \{o^k(i) : k \in \mathbb{Z}\}$, see Figure 1. By $\mathcal{O}(P)$ we denote the set of all orbits of a convex point set P, that is $\mathcal{O}(P) := \{\mathcal{O}(i) : i \in P\}$.

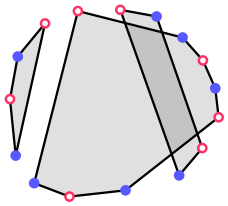


Figure 1 Orbits.

It can be seen that each point belongs to exactly one orbit. Any two neighboring points on an orbit have different colors, so each orbit has an equal number of red and blue points.

Next, we state a number of properties that can be shown to hold for orbits.

▶ **Property 2.2.** *Points i and j of different colors form a feasible pair iff $\mathcal{O}(i) = \mathcal{O}(j)$.*

If i and j are neighboring vertices of a convex polygon defined by the points of an orbit such that i precedes j in the positive direction, then $j = o(i)$. Informally, by repeatedly applying function o we visit all points of an orbit in a single turn around the polygon.

All feasible point pairs can be split into the two categories depending on their mutual position in their orbit. Pairs consisting of two neighboring vertices of the orbit are called *edges*, and all other pairs are called *diagonals*. More precisely, for $j \in \mathcal{O}(i)$, (i, j) is an edge if and only if $i = o(j)$ or $j = o(i)$, otherwise, it is a diagonal.

▶ **Lemma 2.3.** *Orbits can be computed in $O(n)$ time.*

We say that an edge $(i, o(i))$ is a *red-blue* edge if $i \in R$, and *blue-red* edge if $i \in B$. We consider edges directed from i to $o(i)$, so points right of an edge $(i, o(i))$ are points $\{i, \ldots, o(i)\} \setminus \{i, o(i)\}$.

▶ **Property 2.4** (Orbit synchronicity)**.** *Let $\mathcal{A}, \mathcal{B} \in \mathcal{O}(P)$. There are no points of $\mathcal{B}$ on the right side of red-blue edges of $\mathcal{A}$ if and only if there are no points of $\mathcal{A}$ on the right of blue-red edges of $\mathcal{B}$.*

▶ **Definition 2.5.** Let $\mathcal{A}, \mathcal{B} \in \mathcal{O}(P)$. We say that $\mathcal{A} \leq \mathcal{B}$ iff there are no points of $\mathcal{B}$ right of red-blue edges of $\mathcal{A}$ and no points of $\mathcal{A}$ right of blue-red edges of $\mathcal{B}$

It can be proven that the relation $\leq$ on orbits is transitive, which together with orbit synchronicity gives us the following important property of orbits.

▶ **Property 2.6.** *The relation $\leq$ on $\mathcal{O}(P)$ is a total order.*

2.1 Orbit graph

The orbit graph for P is a directed acyclic graph in which each vertex corresponds to an orbit in $\mathcal{O}(P)$, and there is a directed edge from $\mathcal{A}$ to $\mathcal{B}$ iff $\mathcal{A} \leq \mathcal{B}$ and orbits $\mathcal{A}$ and $\mathcal{B}$ are different and intersect each other (meaning that there is a line segment between a pair in $\mathcal{A}$ and a line segment between a pair in $\mathcal{B}$ so that those line segments intersect).

▶ **Property 2.7.** *Each weakly connected component of the orbit graph has a Hamiltonian path.*

These Hamiltonian paths are possible to calculate without constructing the full orbit graph first, as the following lemma states.

▶ **Lemma 2.8.** *All Hamiltonion paths of connected components of the orbit graph can be computed in $O(n)$ total time.*

3 Finding bottleneck matchings

3.1 Cascades

Now, what remains is to utilize the theory that we developed for orbits and the orbit graph, combining it with some parts of the approaches used in [6] to tackle the monochromatic case.

Let us consider the division of the polygon defined by points in P into regions obtained by cutting it with diagonals (but not edges) of the given matching M. Each region is bounded by some diagonals of M and by the polygon's boundary. We call a region *k-bounded* if there are exactly k diagonals bounding it. Any maximal sequence of diagonals connected by 2-bounded regions is called a *cascade*, see Figure 2. We can prove the following lemma.

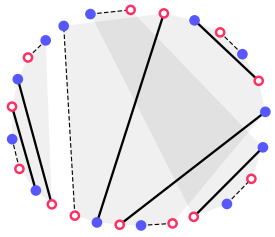


Figure 2 Matching consisting of edges (dashed lines) and diagonals (solid lines). There are three cascades in this example: one consist of the three diagonals in the left part, one consist of the two diagonals in the lower right, and one consist of the single diagonal in the upper right.

▶ **Lemma 3.1.** *There is a bottleneck matching having at most three cascades.*

It is not possible for a matching to have exactly two cascades, so we know that there is a bottleneck matching either with at most one cascade, or with exactly three cascades. We define a set of subproblems that is used to find an optimal solution in both of these cases.

3.2 Subproblems

When talking about matchings with minimal value under certain constraints, we will refer to these matchings as *optimal*.

Let (i, j) be such that $\{i, \ldots, j\}$ contains the same number of red and blue points. We define $\textsc{Matching}(i, j)$ to be the problem of finding an optimal matching $M_{i,j}$ of points $\{i, \ldots, j\}$, so that $M_{i,j}$ has at most one cascade, and pair (i, j) belongs to a region bounded by at most one diagonal from $M_{i,j}$ different from (i, j).

All these subproblems can be solved in $O(n^2)$ total time using dynamic programming. Beside the value $S_{i,j}$ of matching $M_{i,j}$, we determine if pair (i, j) is necessary for constructing $M_{i,j}$, i.e. do all solutions to $\textsc{Matching}(i, j)$ contain (i, j). If that is true then such a pair is called *necessary*. This can be easily calculated together with the solution to subproblems.

An optimal matching of the whole set P having at most one cascade can be found in linear time from calculated solutions to subproblems. We run through all subproblems of the form $\textsc{Matching}[i + 1, i]$ for all feasible pairs $(i, i + 1)$, and take the minimum.

Next, we focus on finding an optimal matching among all matchings with exactly three cascades (3-*cascade matchings*). Any three distinct points i, j and k, where (i, j), $(j + 1, k)$ and $(k + 1, i - 1)$ are feasible pairs, can be used to construct a 3-cascade matching by taking a union of $M_{i,j}$, $M_{j+1,k}$ and $M_{k+1,i-1}$. We can run through all possible triplets (i, j, k) and see which one minimizes $\max\{S[i, j], S[j + 1, k], S[k + 1, i - 1]\}$. However, that requires $O(n^3)$ time, and thus is not suitable, since our goal is to design a faster algorithm. Our approach is

to show that instead of looking at all (i, j) pairs, it is enough to select (i, j) from a set of linear size, which would reduce the search space to quadratic number of possibilities.

3.3 Candidate pairs and polarity

In 3-cascade matching, we call the three diagonals at the inner ends of the three cascades the *inner diagonals*. We take the largest region by area, such that it is bounded, but not crossed by matched pairs, and such that each two of the three cascades are separated by that region. We call this region the *inner region*. Matched pairs defining the boundary of the inner region are called the *inner pairs*.

▶ **Lemma 3.2.** *If there is no bottleneck matching with at most one cascade, then there is a bottleneck* 3*-cascade matching whose every inner pair is necessary.*

The *turning angle* of $\{i, \ldots, j\}$, denoted by $\tau(i, j)$, is the angle by which the vector $\overrightarrow{v_i v_{i+1}}$ should be rotated in positive direction to align with vector $\overrightarrow{v_{j-1} v_j}$, see Figure 3.

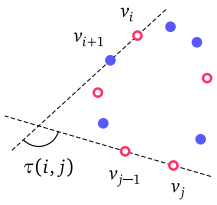


Figure 3 Turning angle.

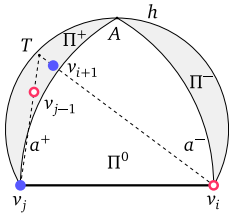


Figure 4 $\{i+1, \ldots, j-1\} \cap \mathcal{O}(i)$ all lie inside either Π^- or Π^+.

We say that (i, j) is a *candidate* pair, if it is a necessary pair and $\tau(i, j) \leq 2\pi/3$.

▶ **Lemma 3.3.** *If there is no bottleneck matching with at most one cascade, then there is a* 3*-cascade bottleneck matching M, such that at least one inner pair of M is a candidate pair.*

Let us now look at a candidate pair (i, j), and examine the position of points $\{i+1, \ldots, j-1\} \cap \mathcal{O}(i)$. We construct the circular arc h on the right side of the directed line $v_i v_j$, from which the line segment $v_i v_j$ subtends an angle of $\pi/3$, see Figure 4. Let A be the midpoint of h. Points v_i, A and v_j form an equilateral triangle, so we can construct the arc a^- between A and v_i with the center in v_j, and the arc a^+ between A and v_j with the center in v_i. These arcs define three areas: Π^-, bounded by h and a^-, Π^+, bounded by h and a^+, and Π^0, bounded by a^-, a^+ and the line segment $v_i v_j$. With $\Pi^-(i, j)$ and $\Pi^+(i, j)$ we respectively denote areas Π^- and Π^+ corresponding to the candidate pair (i, j).

▶ **Lemma 3.4.** *If (i, j) is a candidate pair, then points $\{v_{i+1}, \ldots, v_{j-1}\} \cap \mathcal{O}(i)$ either all belong to Π^- or all belong to Π^+.*

Two possibilities for a candidate pair (i, j) provided by Lemma 3.4 bring forth a concept of *polarity*. If points $\{i+1, \ldots, j-1\} \cap \mathcal{O}(i)$ lie in $\Pi^-(i, j)$ we say that candidate pair (i, j) has *negative polarity* and has i as its *pole*. Otherwise, if these points lie in $\Pi^+(i, j)$, we say that (i, j) has *positive polarity* and pole in j. The following lemma gives us the crucial observation about polarity, which enables us to limit the search space of the algorithm.

▶ **Lemma 3.5.** *No two candidate pairs of the same polarity have the same point as a pole. Hence, there are $O(n)$ candidate pairs.*

Finally, we use our findings from Lemma 3.3 and Lemma 3.5, to construct an algorithm which finds a BBNCM in $O(n^2)$ time. We first solve all subproblems and find candidate pairs. Then, we minimize $\max\{S[i,j], S[j+1,k], S[k+1,i-1]\}$ by running through all candidate pairs (i,j) and for each such pair through all $k \in \{j+1,\ldots,i-1\}$.

4 Points on a circle

Now, let us consider the special case where all points lie on a circle. The geometry of a circle provides us with the following lemma (also stated in [3]), which together with orbit properties enables us to construct an $O(n)$ time algorithm for this problem.

▶ **Lemma 4.1.** *There is a bottleneck matching in which each point i is connected either to $o(i)$ or $o^{-1}(i)$.*

This means there is a bottleneck matching M_E which can be constructed by taking alternating edges from each orbit, that is from each orbit we take either all red-blue or all blue-red edges. To find a bottleneck matching we want to search through only such matchings, and to reduce the number of possibilities, we use the properties of the orbit graph.

Consider the Hamiltonian path $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_m$ for some connected component of the orbit graph, as provided by Property 2.7. Since there is a directed edge from $\mathcal{L}_k$ to $\mathcal{L}_{k+1}$, those two orbits intersect each other, and by Property 2.4 we know that only edges from $\mathcal{L}_k$ that intersect $\mathcal{L}_{k+1}$ are blue-red edges, and only edges from $\mathcal{L}_{k+1}$ that intersect $\mathcal{L}_k$ are red-blue edges. Hence, M_E cannot have blue-red edges from $\mathcal{L}_k$ and red-blue edges from $\mathcal{L}_{k+1}$. This further implies that there is $l \in 0, 1, \ldots, m$ such that $\mathcal{L}_1, \ldots, \mathcal{L}_l$ all contribute to M_E with red-blue edges and $\mathcal{L}_{l+1}, \ldots, \mathcal{L}_m$ all contribute to M_E with blue-red edges.

For each l, we can compute the longest red-blue edge in $\mathcal{L}_1, \ldots, \mathcal{L}_l$ and the longest blue-red edge in $\mathcal{L}_{l+1}, \ldots, \mathcal{L}_m$, and computing all of this can be done in $O(n)$ total time. After we obtain these values, we can quickly get the value of a matching for each possible l, and take the one with the minimum value.

By Lemmas 2.3 and 2.8, each step in this process has time complexity not greater than $O(n)$, so we get an algorithm for points on a circle which runs in linear time.

References

1 A Karim Abu-Affash, Ahmad Biniaz, Paz Carmi, Anil Maheshwari, and Michiel Smid. Approximating the bottleneck plane perfect matching of a point set. *Computational Geometry*, 48(9):718 – 731, 2015.

2 A Karim Abu-Affash, Paz Carmi, Matthew J Katz, and Yohai Trabelsi. Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47(3):447–457, 2014.

3 Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Bottleneck bichromatic plane matching of points. Canadian Conference on Computational Geometry, 2014.

4 John Gunnar Carlsson, Benjamin Armbruster, Haritha Bellam, and Saladi Rahul. A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry and Applications*, to appear.

5 Alon Efrat, Alon Itai, and Matthew J Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

6 Marko Savić and Miloš Stojaković. Faster bottleneck non-crossing matchings of points in convex position. *Computational Geometry*, 65:27–34, 2017.

7 Marko Savić and Miloš Stojaković. Bottleneck bichromatic non-crossing matchings using orbits. *ArXiv e-prints*, 2018. `arXiv:1802.06301`.

Efficient Algorithms for Ortho-Radial Graph Drawing*

Benjamin Niedermann[1], Ignaz Rutter[2], and Matthias Wolf[3]

1 **University of Bonn**
`niedermann@uni-bonn.de`
2 **TU Eindhoven**
`i.rutter@tue.nl`
3 **Karlsruhe Institute of Technology (KIT)**
`matthias.wolf@kit.edu`

Abstract

An ortho-radial drawing is an embedding of a graph into an ortho-radial grid, whose gridlines are concentric circles around the origin and straight-line spokes emanating from the origin but excluding the origin itself. Recently, Barth et al. [1] showed that such drawings can be described combinatorially by so-called *valid ortho-radial representations*, which only specify angles at vertices and bends on the edges, but neglects any kind of geometric information. A similar representation for embeddings in orthogonal grids is a central ingredient for bend minimization in this setting [5]. However, the result of Barth et al. [1] is existential and does not provide an efficient algorithm for testing whether a given ortho-radial representation is valid, let alone actually obtaining a drawing from a valid ortho-radial representation. In this paper, we provide efficient algorithms (with quadratic running time) for these two problems.

1 Introduction

Grid drawings of graphs embed graphs into grids such that vertices map to grid points and edges map to internally disjoint curves on the grid lines that connect their endpoints. Orthogonal grids, whose grid lines are horizontal and vertical lines, are popular and widely used in graph drawing. Ortho-radial drawings are a generalization of this concept to grids that are formed by concentric circles and straight-line spokes from the center but excluding the center. Among other applications, they are used to visualize network maps; see Fig. 1. Equivalently, they can be viewed as graphs drawn in an orthogonal fashion on the surface of a standing cylinder; see Fig. 2. We will use these different points of view interchangeably.

The main objective in orthogonal graph drawing is to minimize the number of bends on the curves. The core of a large fraction of the algorithmic work on this problem is the Topology-Shape-Metrics framework (TSM) introduced by Tamassia [5], which shows that orthogonal drawings can be described purely combinatorially by giving an *orthogonal representation*, which describes (i) the angles formed by consecutive edges around each vertex and (ii) the directions of bends along the edges. Such a representation is *valid* if (I) the angles around each vertex sum to 360°, and (II) the sum of the angles around each face with k vertices is $(2k-2)\cdot 180°$ for internal faces and $(2k+2)\cdot 180°$ for the outer face.

Recently Barth et al. [1] generalized orthogonal representations to ortho-radial drawings, called *ortho-radial* representations. Previously such representations were only known to exist for cycles and Θ-graphs [4] as well as cubic graphs with rectangular faces [3].

Let $G=(V,E)$ be a planar graph with an ortho-radial drawing Δ. We distinguish two types of simple cycles in G. If the center of the grid lies in the interior of a simple cycle, the

* This research was funded in part by Humility & Conviction in Public Life, a project of the University Connecticut sponsored by the John Templeton Foundation and the Helmholtz Program Storage and Cross-linked Infrastructures, Topic 6 Superconductivity, Networks and System Integration.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

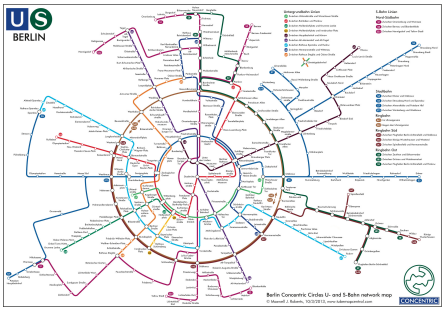
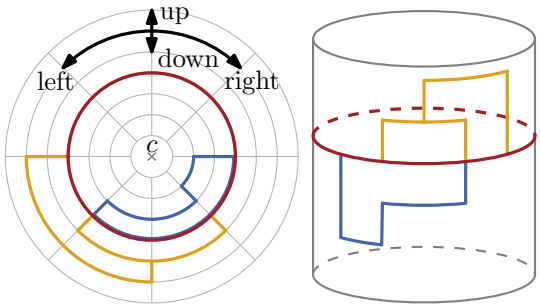


Figure 1 Metro map of Berlin using an ortho-radial layout. Image copyright by Maxwell J. Roberts. Reproduced with permission.



(a) Ortho-radial grid. **(b)** Cylinder drawing.

Figure 2 An ortho-radial drawing of a graph on a grid (a) and its equivalent interpretation as an orthogonal drawing on a cylinder (b).

cycle is *essential*; otherwise it is *non-essential*. Further, there is an unbounded face in Δ and a face that contains the center of the grid; we call the former the *outer face* and the latter the *central face*, which we mark by a small "x"; see Fig. 2a. All other faces are called *regular*.

Consider now an ortho-radial drawing Γ of G (as in [1] we restrict our attention to drawings without bends). An *ortho-radial representation* can be obtained from Γ by measuring for each incidence of a vertex v to a face f that lies to the right of the edges uv and vw the clockwise angle $a \in \{90°, 180°, 270°, 360°\}$. We call such a set of angles to vertex–face incidences an *angle assignment*. Given an angle assignment, for two edges uv and vw incident to the same vertex v, we define the *rotation* $\mathrm{rot}(uvw)$ as 1 if there is a right turn at v, 0 if uvw is straight, and -1 if there is a left turn at v. In the special case that $u = w$, we define $\mathrm{rot}(uvw) = -2$. The rotation of a path $P = v_1, \ldots, v_k$ is $\mathrm{rot}(P) = \sum_{i=1}^{k-1} \mathrm{rot}(v_{i-1}v_iv_{i+1})$ and the rotation of a cycle $C = v_1, \ldots, v_k, v_1$ is $\mathrm{rot}(C) = \sum_{i=1}^{k} \mathrm{rot}(v_{i-1}v_iv_{i+1})$, where we take $v_0 = v_k$ and $v_{k+1} = v_1$. For a face f we use $\mathrm{rot}(f)$ to denote the rotation of the facial cycle that bounds f (oriented such that f lies on the right side of the cycle).

A representation obtained from an ortho-radial drawing satisfies the following conditions.

(I) The sum of all angles around each vertex is 360°.

(II) For each face f: $\mathrm{rot}(f) = \begin{cases} 4, & f \text{ is a regular face} \\ 0, & f \text{ is the outer or the central face but not both} \\ -4, & f \text{ is both the outer and the central face.} \end{cases}$

We call a representation satisfying these properties an *ortho-radial representation*. However, not every ortho-radial representation has a corresponding ortho-radial drawing [1]. Barth et al. show that this requires a third condition, which essentially states that each essential cycle that contains an edge that points upward (on the cylinder) also has to contain an edge that points downward (and vice versa). More formally, fix a horizontal edge $e^\star = st$ on the outer face that points to the right as *reference* edge.

For a simple, essential cycle C in G and a path P from the endpoint s of the reference edge $e^\star$ to a vertex v on C the *labeling* ℓ_C^P assigns to each edge e on C the label $\ell_C^P(e) = \mathrm{rot}(e^\star + P + C[v, e])$. Note that we use $+$ to denote concatenation of paths and $C[v, e]$ to denote the subpath of C starting at v and ending with the edge e. In this paper we always assume that P is *elementary*, i.e., P intersects C only at its endpoints. For these paths the labeling is independent of the actual choice of P [1]. We therefore drop the superscript P and write $\ell_C(e)$ for the labeling of an edge e on an essential cycle C. We call an essential cycle *monotone* if either all its labels are non-negative or all its labels are non-positive. A

monotone cycle is a *decreasing* cycle if it has at least one strictly positive label, and it is an *increasing* cycle if C has at least one strictly negative label. An ortho-radial representation is *valid* if it contains neither decreasing nor increasing cycles. The validity of an orthor-radial representation ensures that on each essential cycle with at least one non-zero label there is at least one edge pointing up and one pointing down. Barth et al. [1] prove the following theorem[1].

▶ **Proposition 1** (Theorem 5 in [2]). *A 4-plane graph admits a bend-free ortho-radial drawing if and only if it admits a valid ortho-radial representation.*

To that end, Barth et al. [2] prove the following results among others. Since we use them throughout this paper, we restate them for the convenience of the reader. Both assume ortho-radial representations that are not necessarily valid.

▶ **Proposition 2** (Lemma 12 in [2]). *Let C_1 and C_2 be two essential cycles and let $H = C_1 + C_2$ be the subgraph of G formed by these two cycles. For any common edge vw of C_1 and C_2 where v lies on the central face of H, the labels of vw are equal, i.e., $\ell_{C_1}(vw) = \ell_{C_2}(vw)$.*

▶ **Proposition 3** (Lemma 16 in [2]). *Let C and C' be two essential cycles that have at least one common vertex. If all edges on C are labeled with 0, C' is neither increasing nor decreasing.*

Proposition 2 is a useful tool for comparing the labels of two interwoven essential cycles. For example, if C_1 is decreasing, we can conclude for all edges of C_2 that also lie on C_1 and that are incident to the central face of H that they are non-negative. Proposition 3 is useful in the scenario where we have an essential cycle C with non-negative labels, and a decreasing cycle C' that shares a vertex with C. We can then conclude that C is also decreasing. In particular, these two propositions together imply that the central face of the graph H formed by two decreasing cycles is bounded by a decreasing cycle.

While it is not hard to test whether a given angle assignment is an ortho-radial representation, Barth et al. left open the problem of testing whether a given ortho-radial representation is valid. Moreover, following their proof of Proposition 1, which constructs from a valid ortho-radial representation a corresponding ortho-radial drawing, requires a quadratic number of such validity tests. In this paper, we show that such a validity test can indeed be implemented to run in quadratic time. Used naively, this would result in a quartic algorithm for constructing a drawing for a valid ortho-radial drawing. We further improve on this and reduce also that running time to quadratic. Thus, our main result is that, given an ortho-radial representation of a graph, we can test in quadratic time whether it is valid, and in the positive case we can produce a corresponding drawing, whereas in the negative case, we find a monotone cycle witnessing that the representation is not valid.

2 Finding Monotone Cycles

The two conditions for ortho-radial representations are local and checking them can easily be done in linear time. We therefore assume in this section that we are given a planar graph G with an ortho-radial representation Γ. The condition for validity however references all essential cycles of which there may be exponentially many. We present an algorithm that checks whether Γ contains a monotone cycle and computes such a cycle if one exists. The main difficulty is that the labels on a decreasing cycle C depend on an elementary path P

[1] In the following we refer to the full version [2] of [1], when citing lemmas and theorems.

from the reference edge to C. However, we know neither the path P nor the cycle C in advance, and choosing a specific cycle C may rule out certain paths P and vice versa.

We only describe how to search for decreasing cycles; increasing cycles can be found by searching for decreasing cycles in a suitably defined mirrored representation. A decreasing cycle C is *outermost* if it is not contained in the interior of any other decreasing cycle. Clearly, if Γ contains a decreasing cycle, then it also has an outermost one, and in fact it can be shown that this cycle is uniquely determined.

▶ **Lemma 4.** *If Γ contains a decreasing cycle, there is a unique outermost decreasing cycle.*

The core of our algorithm is an adapted left-first DFS. Given a directed edge e, it determines the outermost decreasing cycle C in Γ such that C contains e in the given direction and e has the smallest label among all edges on C, if such a cycle exists. By running this test for each directed edge of G as the start edge, we find a decreasing cycle if one exists.

Our algorithm is based on a DFS that visits each vertex at most once. A left-first search maintains for each visited vertex v a reference edge $\mathrm{ref}(v)$, the edge of the search tree via which v was visited, and whenever it has a choice which vertex to visit next, it picks the first outgoing edge in clockwise direction after the reference edge that leads to an unvisited vertex. In addition to that, we employ a filter that ignores certain outgoing edges during the search. To that end, we define for all outgoing edges e incident to a visited vertex v a *search label* $\tilde{\ell}(e)$ by setting $\tilde{\ell}(e) = \tilde{\ell}(\mathrm{ref}(v)) + \mathrm{rot}(\mathrm{ref}(v) + e)$ for each outgoing edge e of v. In our search we ignore edges with negative search labels. For a given directed edge $e = vw$ in a graph G with ortho-radial representation Γ, we initialize the search by setting $\mathrm{ref}(w) = vw$, $\tilde{\ell}(e) = 0$ and then start searching from w.

Let T denote the directed search tree with root w constructed by the DFS in this fashion. If T contains v, then this determines a *candidate cycle* C containing the edge vw. If C is a decreasing cycle, which we can easily check by determining an elementary path from the reference edge to C, we report it. Otherwise, we show that there is no outermost decreasing cycle C such that vw is contained in C and has the smallest label among all edges on C.

▶ **Lemma 5.** *Assume Γ contains a decreasing cycle. Let C be the outermost decreasing cycle of Γ and let vw be an edge on C with the minimum label, i.e., $\ell_C(vw) \leq \ell_C(e)$ for all edges e of C. Then the left-first DFS from vw finds C.*

Proof Sketch. To prove this we assume for a contradiction that the search does not find C. Then, starting from vw there is a first edge xy of C that is not followed by the search. This can be for one of two reasons. Either it was filtered out, since the search label $\tilde{\ell}(xy)$ is less then 0, or, at the point the edge xy is considered by the search vertex y has already been visited. The first case is easily excluded using the assumption that $\ell_C(vw)$ is minimal. In the second case, we find that our search contains a search path Q that at some point deviates from C and then reaches y, before it eventually backtracks and reaches xy.

Let P be the subpath of Q from the vertex b where Q first leaves C to the first vertex p that again lies on C; see Fig. 3. The subgraph $H = P + C$ that is formed by the decreasing cycle C and the path P is a Θ-graph consisting of the three internally vertex-disjoint paths $P[b, p]$, $C[b, p]$ and $\overline{C}[b, p]$ between b and p. Due to the left-first rule, the circular ordering of the edges around the vertex b is fixed, and C is essential. Consequently the interior of C is the central face. We denote the cycle bounding the outer face but in which the edges are directed such that the outer face lies locally to the left by C'. Depending on which of the two remaining faces is the outer face, the situation is as shown in Fig. 4. One can then show that in either case the cycle bounding the outer face is a decreasing cycle, thus contradicting the assumption that C is outermost. In the remainder we sketch the proof of the easier case.

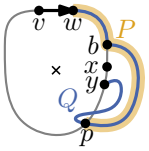


Figure 3 Path Q and its prefix P that leaves C once and ends at a vertex p of C.

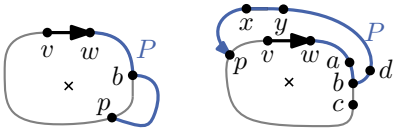


Figure 4 The two possible embeddings of the subgraph formed by the decreasing cycle C and the path P, which was found by the search.

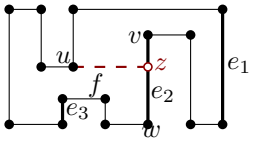


Figure 5 Candidate edges (bold) for a port u.

If $\overline{C'} = \overline{C}[b,p] + \overline{P}[p,b]$ forms the outer face of H, vw lies on C'; see Fig. 4, left. We show that C' is a decreasing cycle, which contradicts the assumption that C is the outermost decreasing cycle. Since P is simple and lies in the exterior of C, the path P is a part of C', which means $C'[w,p] = P$. The other part of C' is formed by $C[p,w]$. Since C forms the central face of H, the labels of the edges on $C[p,w]$ are the same for C and C' by Proposition 2. In particular, $\ell_C(vw) = \ell_{C'}(vw)$ and all the labels of edges on $C[p,w]$ are non-negative because C is decreasing. The label of any edge e on both C' and P is $\ell_{C'}(e) = \ell_{C'}(vw) + \mathrm{rot}(vw + P[w,e]) = \ell_C(vw) + \tilde{\ell}(e) \geq 0$. Thus, the labeling of C' is non-negative. Further, not all labels of C' are 0 since otherwise C is not a decreasing cycle by Proposition 3. Hence, C' is decreasing and contains C in its interior, a contradiction. ◀

The left-first DFS clearly runs in $O(n)$ time. We run it for each of the $O(n)$ directed edges of G. Since some edge must have the lowest label on the outermost decreasing cycle, Lemma 5 guarantees that we eventually find a decreasing cycle if one exists. Decreasing cycles can be detected symmetrically.

▶ **Theorem 6.** *Let G be a 4-planar graph on n vertices and let Γ be an ortho-radial representation of G. It can be determined in $O(n^2)$ time whether Γ is valid.*

3 Rectangulation

The core of the algorithm for drawing a valid ortho-radial representation Γ of a graph G by Barth et al. [1] is a *rectangulation procedure* that successively augments G with new vertices and edges to a graph $G^\star$ along with a valid ortho-radial representation $\Gamma^\star$ where every face of $G^\star$ is a *rectangle*, i.e., a face with no concave angles. In particular the algorithm only inserts edges that preserve the validity of the ortho-radial representation. To that end it applies $O(n^2)$ validity tests. Using the validity test from Section 2 the rectangulation algorithm runs in $O(n^4)$ time. We now sketch the procedure in more detail and briefly describe how to improve its running time to $O(n^2)$ time.

Consider a face f with a concave angle at u such that the following two turns when walking along f (in clockwise direction) are right turns; see Fig. 5. We call u a *port* of f and define a set of *candidate edges* that contains those edges e of f, for which $\mathrm{rot}(f[u,e]) = 2$. We treat this set as a sequence $e_1, \ldots, e_l$, where the edges appear in the same order as in f beginning with the first candidate after u. The *augmentation* Γ^u_{vw} with respect to a candidate edge vw is obtained by splitting vw into the edges vz and zw, where z is a new vertex, and adding the edge uz in the interior of f such that the angle formed by zu and the edge following u on f is 90°. The direction of the new edge uz in Γ^u_{vw} is the same for all candidate edges. If this direction is vertical, we call u a *vertical port* and otherwise a *horizontal port*.

The rectangulation algorithm successively removes ports by inserting edges. For a vertical

port u it inserts a vertical edge from u to its first candidate e_1, which yields a valid ortho-radial representation $\Gamma^u_{e_1}$ [2, Lemma 21]. For a horizontal port u it successively goes through candidates $e_1, \ldots, e_l$ of u and takes the first valid augmentation $\Gamma^u_{e_i}$, which always exists [2] and can be identified by the validity test of Section 2. Since there are $O(n)$ concave angles to remove and $O(n)$ candidates per augmentation, the algorithm runs in $O(n^4)$.

Exploiting structural insights we speed up each validity test to $O(n)$ time obtaining $O(n^3)$ running time in total. Since validity tests are only performed for horizontal ports, we focus on removing the concave angle at a horizontal port u in the remainder of the section. We further assume without loss of generality that the new edge uz points right. The key insight to improve the running time of a single validity tests is the following structural observation.

▶ **Lemma 7.** *If the first augmentation $\Gamma^u_{e_1}$ contains a decreasing cycle, the new edge uz in any augmentation Γ^u_e has label $\ell_C(uz) = 0$ on any decreasing cycle C.*

Hence, uz has the minimum label on all decreasing cycles and therefore one DFS from uz suffices to check for decreasing cycles. For increasing cycles a similar observation does not hold but in the rectangulation procedure only the final test for u needs to test for increasing cycles. This test can be replaced by a test for a horizontal path [2, Lemmas 25, 26]. This improves the running time of each applied validity test to $O(n)$.

Finally, we reduce the number of validity tests to $O(n)$ obtaining $O(n^2)$ running time for the rectangulation algorithm. We introduce a post-processing phase after each augmentation step, which ensures that all but a constant number of candidates that were considered for u lie on rectangular faces. Thus, these edges will not be candidates for any future port. All concave angles introduced in the post-processing phase become vertical ports, for which no validity tests are performed. Hence, the total number of validity tests is reduced to $O(n)$.

▶ **Theorem 8.** *Given a valid ortho-radial representation Γ of a graph G, a corresponding rectangulation can be computed in $O(n^2)$ time.*

In particular, using Corollary 19 from [2], given a graph G with valid ortho-radial representation Γ, a corresponding ortho-radial drawing Δ can be computed in $O(n^2)$ time. This concludes the construction of a TSM framework for ortho-radial drawings. An interesting open problem is to devise algorithms that construct valid ortho-radial representations using a small number of bend vertices. In particular, what is the complexity of testing whether a given graph with a fixed embedding has an ortho-radial drawing without bends?

Acknowledgement. We thank Lukas Barth for interesting discussions.

References

1 L. Barth, B. Niedermann, I. Rutter, and M. Wolf. Towards a Topology-Shape-Metrics framework for ortho-radial drawings. In B. Aronov and M.J. Katz, editors, *SoCG 2017*, volume 77 of *LIPIcs*, pages 14:1–14:16, 2017.

2 L. Barth, B. Niedermann, I. Rutter, and M. Wolf. Towards a Topology-Shape-Metrics framework for ortho-radial drawings. *CoRR*, arXiv:1703.06040, 2017.

3 M. Hasheminezhad, S. M. Hashemi, B. D. McKay, and M. Tahmasbi. Rectangular-radial drawings of cubic plane graphs. *Computational Geometry: Theory and Applications*, 43:767–780, 2010.

4 M. Hasheminezhad, S. M. Hashemi, and M. Tahmasbi. Ortho-radial drawings of graphs. *Australasian Journal of Combinatorics*, 44:171–182, 2009.

5 R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

On Primal-Dual Circle Representations

Stefan Felsner[1] **and Günter Rote**[2]

1 **Technische Universität Berlin, Germany**
felsner@math.tu-berlin.de
2 **Freie Universität Berlin, Germany**
rote@inf.fu-berlin.de

Abstract

The Koebe-Andreev-Thurston Circle Packing Theorem states that every triangulated planar graph has a circle-contact representation. The theorem has been generalized in various ways. The arguably most prominent generalization assures the existence of a primal-dual circle representation for every 3-connected planar graph. The aim of this note is to give a streamlined proof of this result.

1 Introduction

For a 3-connected plane graph $G = (V, E)$ with face set F, a *primal-dual circle representation* of G consists of two families of circles $(C_x : x \in V)$ and $(D_y : y \in F)$ such that:

(i) The vertex-circles C_x have pairwise disjoint interiors.

(ii) All face-circles D_y are contained in the circle D_o corresponding to the outer face o, and all other face-circles have pairwise disjoint interiors.

Moreover, for every edge $xx' \in E$ with dual edge yy' (i. e., y and y' are the two faces separated by xx'), the following holds:

(iii) Circles C_x and $C_{x'}$ are tangent at a point p with tangent line $t_{xx'}$.

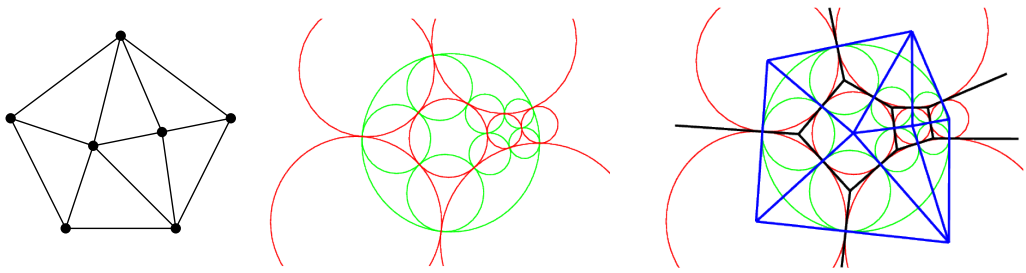
(iv) Circles D_y and $D_{y'}$ are tangent at the same point p with tangent line $t_{yy'}$.

(v) The lines $t_{xx'}$ and $t_{yy'}$ are orthogonal.

Figure 1 shows an example.

▶ **Theorem 1.** *Every 3-connected plane graph G admits a primal-dual circle representation. Moreover this representation is unique up to Möbius transformations.*

The proof presented here combines ideas from an unpublished manuscript of Pulleyblank and Rote, from Brightwell and Scheinerman [2] and from Mohar [11]. The motive for the write-up is that the amount of calculations needed for the proof has been reduced significantly.



(a) (b) (c)

Figure 1 (a) a 3-connected graph G, (b) a primal-dual circle representation of G, (c) straight-line drawings of G and the dual graph G^*, yielding a tessellation by kites.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

The proof of the theorem is given in the next section. Before getting there we give a brief account of the history of the theorem and links to applications.

In graph theory the study of circle contact representations can be traced back to the 1970's and 1980's; the term "coin representation" was used there. In a note written in 1991, Sachs [13] mentions that he found a proof of the circle packing theorem which was based on conformal mappings. This eventually lead him to the discovery that the theorem had been proved by Koebe as early as 1936 [8].

In the context of his study of 3-manifolds, Thurston [14, Sec. 13.6] proved that any triangulation of the sphere has an associated "circle packing" which is unique up to Möbius transformations. This result was already present in earlier work of Andreev [1]. Nowadays the result is commonly referred to as the *Koebe-Andreev-Thurston Circle Packing Theorem*.

In the early 1990's new proofs of the circle packing theorem where found. Colin de Verdière gave two proofs. The first is an existential proof using 'invariance of domain' [3]; the second is based on the minimization of a convex function [4]. Pulleyblank and Rote (unpublished) and Brightwell and Scheinerman [2] gave proofs of the primal-dual version (Theorem 1) based on an iterative algorithm, similar to the proof given in this note. Mohar [10] analyzed an improved iterative approach and proved its convergence in polynomial time.

Primal-dual circle representations yield *simultaneous orthogonal drawings* of G and its dual G^*, i. e., straight-line drawings of G and G^* such that the outer vertex of G^* is at infinity and each pair of dual edges is orthogonal. The existence of such drawings was conjectured by Tutte [15].

The Circle Packing Theorem has been used to prove *Separator Theorem*s. The approach was pioneered by Miller and Thurston and generalized to arbitrary dimensions by Miller, Teng, Thurston, and Vavasis [9]. The 2-dimensional case is reviewed in Pach and Agarwal [12, Chapter 8]. A slightly simpler proof was proposed by Har-Peled [7].

The theorem also has applications in Graph Drawing. Eppstein [5] used circle representations to prove the existence of *Lombardi drawing* (a drawing in which the edges are drawn as circular arcs, meeting at equal angles at each vertex) for all subcubic planar graphs. Felsner et al. [6] used circle representations to show that 3-connected planar graphs have planar *strongly monotone drawings*, i. e., straight-line drawings such that for for any two vertices u, v there is a path which is monotone with respect to the connecting line of u and v.

2 Primal-Dual Circle Representation: The Proof

Before diving into details we give a rough outline of the proof. A primal-dual circle representation of G induces a straight-line drawing of G and a straight-line drawing of the dual. Superimposing the two drawings yields a plane drawing whose faces are special quadrangles called kites, see Figures 1c and 3. After guessing radii for the circles, the shapes of the kites are determined. It is then checked whether the angles of kites meeting at a vertex sum up to 2π. If at some vertex the angle sum does not match 2π, the radii are changed to correct the situation. The process is designed to make the radii converge and to make the sum of angles meet the intended value at each vertex. The second part of the proof consists of showing that the kites corresponding to the final radii can be laid out to form a tessellation, thus giving the centers of a primal-dual circle representation of G.

Proof of Theorem 1. Given a primal-dual circle representation of G, we can use a stereographic projection to lift it to a primal-dual circle representation on the sphere. This spherical representation has the advantage that the circle D_o has no special role. On the sphere the face-circles can be viewed as a family with pairwise disjoint interiors. Rotating this representation

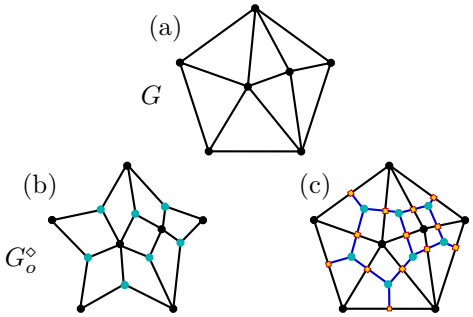


Figure 2 (a) A plane graph G. (b) Its reduced angle graph $G_o^\diamond$. (c) Its primal-dual completion (skeleton graph of kites).

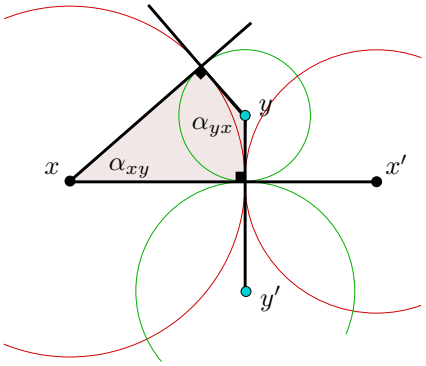


Figure 3 The kite corresponding to the incident vertex-face pair x,y.

and mapping it back to the plane, we can get a primal-dual circle representation of G or of the dual G^* where any prescribed element $z \in V \cup F$ has the role of the outer face. This process can be reverted. Therefore, we use the well-known fact that G or G^* has a triangular face, (from Euler's formula), and assume that the outer face o of the given plane graph is a triangle.

Given a primal-dual circle representation of G we can use the centers of the circles C_x for $x \in V$ to obtain a planar straight-line drawing of G. Similarly the centers of the circles D_z for $z \in F \setminus \{o\}$ yield a planar straight-line drawing of $G^* \setminus \{o\}$. Looking at the two drawings simultaneously and adding appropriate rays for the edges yo of G^* we see *kites*, i. e., quadrangular shapes with two opposite right angles, tessellating the polygon formed by the centers of the outer vertices of G, see Figure 1c.

The kites are in bijection with the incident pairs (x, y), where x is a primal vertex and y is a dual vertex. Since the involved circles intersect orthogonally, the kite of x and y (see Figure 3) is completely determined by the radii r_x of C_x and r_y of D_y. The angles at x and y are given by

$$\alpha_{xy} = 2 \arctan \frac{r_y}{r_x} \quad \text{and} \quad \alpha_{yx} = 2 \arctan \frac{r_x}{r_y}. \tag{1}$$

The *angle graph* of a plane graph $G = (V, E)$ is the graph $G^\diamond$ whose vertex set is $V \cup F$ and whose edges are the incident pairs xy with $x \in V$, $y \in F$, i. e., x is a vertex on the boundary of y. The graph $G^\diamond$ is plane, bipartite and every face is a 4-gon, i. e., $G^\diamond$ is a quadrangulation. Let $G_o^\diamond = (U, K)$ be the *reduced angle graph*, obtained by deleting the vertex corresponding to the outer face of G from $G^\diamond$, see Figure 2(b). (Note that the outer face of the graph G in this example is a pentagon, unlike in our setup, where we assume a triangular outer face.) The set K of edges of $G_o^\diamond$ is in bijection to the kites of a primal-dual circle representation of G. We will need the following property of $G_o^\diamond$.

▶ Claim 1. Every subset S of the vertices of $G_o^\diamond$ with $|S| \geq 5$ induces at most $2|S| - 5$ edges.

Proof. Since $G_o^\diamond$ is bipartite, every subset S induces at most $2|S| - 4$ edges, with equality only if S induces a quadrangulation. Since the outer face of G is incident to 3 vertices we have $|K| = 2(|U| + 1) - 4 - 3 = 2|U| - 5$. Now let $S \subsetneq U$. Since G is 3-connected, there is no separating 4-cycle in $G^\diamond$. This implies that the outer face of the induced graph $G_o^\diamond[S]$ is not a 4-cycle, whence $G_o^\diamond[S]$ has at most $2|S| - 5$ edges. ◀

We specify that the triangle formed by the three outer vertices should be equilateral. This is no loss of generality, since it can be achieved for any primal-dual circle representation by applying a Möbius transformation. After this normalization, the following equations hold:

$$\sum_{w\colon uw\in K} \alpha_{uw} = \begin{cases} \pi/3 & \text{if } u \text{ is an outer vertex of } G \\ 2\pi & \text{else.} \end{cases}$$

Define the *target angles* $\beta(u)$ for $u \in U$ such that $\beta(u) = \pi/3$ if u is an outer vertex and $\beta(u) = 2\pi$ for all other vertices and all bounded faces of G.

Given an arbitrary assignment $r : U \to \mathbb{R}_+$ of radii, we can form the corresponding kites. The angle sum at $u \in U$ is then $\alpha(u) = \sum_{w\colon uw\in K} \alpha_{uw}$. We aim at finding radii such that $\alpha(u) = \beta(u)$ for all $u \in U$. Later we will show that such radii induce a primal-dual circle representation.

We first show that $\sum_u \alpha(u) = \sum_u \beta(u)$, i. e., any choice of radii attains the correct target angles *on average*. Indeed,

$$\sum_u \alpha(u) = \sum_{xy\in K} \pi = |K|\pi \qquad \text{and}$$

$$\sum_u \beta(u) = \big((|V|-3)+(|F|-1)\big)2\pi + 3\frac{\pi}{3} = \big(|V|+|F|-1\big)2\pi - 5\pi = (2|U|-5)\pi = |K|\pi.$$

As a consequence, whenever $\alpha(u) \neq \beta(u)$ for some u, the following two sets are nonempty:

$$U_- = \{u \in U\colon \alpha(u) < \beta(u)\} \qquad \text{and} \qquad U_+ = \{u \in U\colon \alpha(u) > \beta(u)\}$$

If we increasing the radius r_u of a vertex $u \in U_+$, we observe from (1) that for every incident edge $uw \in K$, the angle α_{uw} decreases monotonically to 0 as $r_u \to \infty$. Hence, it is possible to increase r_u to the unique value where $\alpha(u) = \beta(u)$.

The core of the proof is the following infinite iteration.

repeat forever: **for all** $u \in U$: **if** $u \in U_+$ **then** increase r_u to make $\alpha(u) = \beta(u)$ (2)

We claim that the radii converge to an assignment with $\alpha(u) = \beta(u)$ for all u. The increase of r_u may cause another element $w \in U_-$ to move to U_+, but a transition from U_+ to U_- is impossible. It follows that some element u_0 must belong to U_- indefinitely unless the iteration comes to a halt with $U_- = U_+ = \emptyset$. Since radii can only increase, $u \in D$ implies that $r_u \to \infty$. We want to show that the set $D \subseteq U_+ \subsetneq U$ of elements whose corresponding radii do not converge is empty. The subset of outer vertices of V in D is denoted by D_o. If $u \in D$ and $w \in U \setminus D$, then α_{uw} converges to 0 according to (1). Thus, for given $\varepsilon > 0$, the iteration will eventually lead to vectors of radii such that $\sum_{w\in U\setminus D\colon uw\in K} \alpha_{uw} \le \frac{\varepsilon}{|U|}$ for each $u \in D$. We now consider the case $|D| \ge 5$ and use Claim 1: (The cases $1 \le |D| \le 4$ must be treated separately.)

$$\sum_{u\in D} \alpha(u) \le \varepsilon + \sum_{\text{kite with } x,y\,\in\, D} (\alpha_{xy}+\alpha_{yx}) = \varepsilon + \sum_{xy \text{ edge of } G_o^\diamond[D]} \pi \quad \le \varepsilon + (2|D|-5)\pi \qquad (3)$$

$$\sum_{u\in D} \alpha(u) = \sum_{u\in D\cap U_+} \alpha(u) \quad > \quad \sum_{u\in D} \beta(u) \quad = \quad 2\pi|D| - \frac{5|D_o|}{3}\pi.$$

By comparing these bounds, we see that $|D_o| = 3$ and the subgraph $G_o^\diamond[D]$ of $G_o^\diamond$ induced by D has $2|D|-5$ edges. This implies that $G_o^\diamond[D]$ is connected and that the outer face of $G_o^\diamond[D]$ includes the three outer vertices of G. Thus, by the edge count, $G_o^\diamond[D]$ is an internal quadrangulation, and the outer face of $G_o^\diamond[D]$ coincides with the hexagonal outer face of $G_o^\diamond$ because this face bounds the unique shortest cyclic walk through the 3 nonadjacent vertices of D_o. Since $G_o^\diamond$ has no separating 4-cycles, we conclude that $G_o^\diamond[D] = G_o^\diamond$. This contradicts $D \subsetneq U$ and shows that D must be empty.

We have shown that all radii are bounded, and hence they converge. It follows that the angle sums $\alpha(u)$ converge as well, and by the nature of the iteration (2), their limits $\hat{\alpha}(u)$ are bounded by $\hat{\alpha}(u) \le \beta(u)$. Since $\sum_u \hat{\alpha}(u) = \sum_u \beta(u)$, we must have $\hat{\alpha}(u) = \beta(u)$ for all u.

Uniqueness up to scaling. Let r and r' be two vectors of radii such that $\alpha_r(u) = \alpha_{r'}(u) = \beta(u)$ for all u and $r_{u_0} = r'_{u_0}$ for some u_0. Suppose that $S = \{u : r_u > r'_u\}$ is nonempty and observe that $u_0 \in \bar{S} = U \setminus S$.

$$0 = \sum_{u\in S} \alpha_r(u) - \sum_{u\in S} \alpha_{r'}(u) \;=\; \sum_{u\in S} \sum_{w\in U\,:\, uw\in K} \alpha_{uw}(r) \;-\; \sum_{u\in S} \sum_{w\in U\,:\, uw\in K} \alpha_{uw}(r') \qquad (4)$$

$$= \sum_{u\in S, w\in \bar{S}, uw\in K} \big(\alpha_{uw}(r) - \alpha_{uw}(r')\big) \;<\; 0 \qquad (5)$$

The equality between (4) and (5) holds because the equation $\alpha_{uw} + \alpha_{wu} = \pi$ is independent of the radii, and hence the contributions of the edges uw with $u, w \in S$ cancel. For the last inequality, note that $\alpha_{uw}(r) < \alpha_{uw}(r')$ due to (1), because $r_u > r'_u$ and $r_w \le r'_w$, and there is some pair $uw \in K$ with $u \in S$ and $w \in \bar{S}$. The contradiction proves uniqueness.

Laying out the kites. To finish the proof of Theorem 1, it remains to show that the kites defined by the limiting radii r can be laid out in the plane with the intended side-to-side contacts, and that the circles with radii given by r and centers as given by the laid-out kites have the properties (i)–(v), i. e., they form a primal-dual circle representation of G.

We first show that if the kites can be laid out without overlap, they yield a primal-dual circle representation. The kites induce a straight-line drawing of G and a straight-line drawing of the dual G^* with the outer vertex o at ∞ and edges yo being represented by rays. The point p where an edge xx' crosses its dual edge yy' is a right angle of kites. This implies (v).

For a vertex $u \in U$, consider the set of kites containing u. These kites can be put together in the cyclic order given by the rotation of u in $G^\diamond_o$ to form a polygon P_u. If u is not one of the three outer vertices V_o, P_u is a convex polygon surrounding u, because $\hat{\alpha}(u) = \beta(u) = 2\pi$. By the geometry of the kites, all edges incident to u have the same length r_u, and the circle C_u of radius r_u centered at u is inscribed in P_u and touches P_u at the common corners of neighboring kites. For $u \in V_o$, the polygon P_u has u as a corner, but the circle C_u still goes through the right-angle corners of the kites. From the incidences of the kites, and since the polygons P_u for $u \in V$ are pairwise disjoint, we obtain that the family $(C_x : x \in V)$ satisfies (i) and (iii).

The union of all kites forms an equilateral triangle T. This forces the radii of the three outer vertices to be equal, whence the touching points of the outer circles are the midpoints of the sides of T. Now, define D_o as the inscribed circle of T. Let the family of circles defined for dual vertices be $(D_y : y \in F)$. Properties (ii) and (iv) follow from the layout of kites.

The layout of kites is warranted by the following Lemma 2. When we apply this lemma, the graph H is the bipartite *primal-dual completion* of $G = (V, E)$. The vertices of H are $V \cup F \setminus \{o\} \cup E$, and the edges of H are the pairs $(z, e) \in (V \cup F \setminus \{o\}) \times E$ for which z is incident to the edge e. This graph is the skeleton of the laid-out kites, see Figure 2(c). ◀

▶ **Lemma 2.** *Let H be a 3-connected plane graph. For every inner face f of H let P_f be a simple polygon whose corners are labeled with the vertices from the boundary of f in the same cyclic order. The corner of P_f labeled with v is denoted $p(f, v)$ and $\alpha_{i,v}$ denotes the angle of P_f at $p(f, v)$. If the following conditions are satisfied:*

- *(i)* $\sum_{i=1}^{k} \alpha_{i,v} = 2\pi$ *for every inner vertex v of H with incident faces $f_1, \ldots, f_k$.*
- *(ii)* $\sum_{i=1}^{k} \alpha_{i,v} \le \pi$ *for every outer vertex v of H with incident faces $f_1, \ldots, f_k$.*
- *(iii)* $\|p(f_1, v) - p(f_1, w)\| = \|p(f_2, v) - p(f_2, w)\|$ *for every inner edge vw of H with incident faces f_1, f_2.*

Then there is a crossing-free straight-line drawing of H in which the drawing of every inner face f can be obtained from P_f by a rigid motion, i. e., translation and rotation.

Proof. Let H^* be the dual graph of H without the vertex corresponding to the outer face of H. Further let S be a spanning tree of H^*. Then by (iii) we can glue the polygons P_f of all inner faces f of H together along the edges of S. This determines a unique position for every polygon, up to a global motion. We need to show that the resulting shape has no holes or overlaps. For the edges of S we already know that the polygons of the two incident faces are touching such that corners corresponding to the same vertex coincide. For the edges of the complement $\overline{S}$ of S we need to show this. Considering $\overline{S}$ as a subset of the edges of H, the set $\overline{S}$ is a forest in H. Let v be a leaf of this forest that is an inner vertex of H, and let e be the edge of $\overline{S}$ incident to v. Then for all incident edges $e' \neq e$ of v we already know that the polygons of the two incident faces of e touch in the right way. But then also the two polygons of the two incident faces of e touch in the right way because v fulfills property (i). Since the set of edges we still need to check remains always a forest, we can iterate this process until all inner edges of H are checked. After gluing all the polygons P_f, every vertex v has a unique position, and because of property (ii), all angles at the boundary of the union are convex. An easy double-counting argument with an application of Euler's Formula shows that the sum of angles at the outer vertices equals $(d-2)\pi$ if there are d outer vertices. This is just the right value for a d-gon, whence the boundary of the union of the glued polygons P_f is a convex polygon and therefore nonintersecting. ◀

References

1 E. M. Andreev, *Convex polyhedra in Lobačevskiĭ spaces*, Mat. Sb. (N.S.), 81 (123) (1970), 445–478. English: Math. USSR, Sb. 10, 413–440 (1971).

2 G. R. Brightwell and E. R. Scheinerman, *Representations of planar graphs*, SIAM J. Discr. Math., 6 (1993), 214–229.

3 Y. Colin de Verdière, *Empilements de cercles: convergence d'une méthode de point fixe*, Forum Math., 1 (1989), 395–402.

4 Y. Colin de Verdière, *Un principe variationnel pour les empilements de cercles*, Invent. Math., 104 (1991), 655–669.

5 D. Eppstein, *A Möbius-invariant power diagram and its applications to soap bubbles and planar Lombardi drawing*, Discrete Comput. Geom., 52 (2014), 515–550.

6 S. Felsner, A. Igamberdiev, P. Kindermann, B. Klemz, T. Mchedlidze, and M. Scheucher, *Strongly monotone drawings of planar graphs*, in Proc. 32nd Intern. Symp. Comput. Geom. (SoCG 2016), vol. 51 of LIPIcs, Dagstuhl, 2016, pp. 37:1–15.

7 S. Har-Peled, *A simple proof of the existence of a planar separator*, arXiv:1105.0103,2011

8 P. Koebe, *Kontaktprobleme der konformen Abbildung*, Ber. Verh. Sächs. Akad. Leipzig, Math.-Phys. Klasse, 88 (1936), pp. 141–164.

9 G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Separators for sphere-packings and nearest neighbor graphs*, J. ACM, 44 (1997), pp. 1–29.

10 B. Mohar, *Circle packings of maps in polynomial time*, Eur. J. Comb. 18 (1997), 785–805.

11 B. Mohar, *Circle packings of maps—the Euclidean case*, Rend. Sem. Mat. Fis. Milano 1997, 67 (2000), 191–206.

12 J. Pach and P. K. Agarwal, *Combinatorial Geometry*, John Wiley & Sons, 1995.

13 H. Sachs, *Coin graphs, polyhedra, and conformal mapping*, Discr. Math., 134 (1994), 133–138.

14 W. Thurston, *Geometry and Topology of Three-Manifolds*, Princeton Univ., 1980. Lect. Notes. Electronic edition: `library.msri.org/books/gt3m/`.

15 W. T. Tutte, *How to draw a graph*, Proc. London Math. Soc. (Ser. 3), 13 (1963), 743–767.

Shape Recognition by a Finite Automaton Robot *

Robert Gmyr[1], Kristian Hinnenthal[1], Irina Kostitsyna[2], Fabian Kuhn[3], Dorian Rudolph[1], and Christian Scheideler[4]

1 **Paderborn University, Germany**
{gmyr, krijan, dorian}@mail.upb.de
2 **TU Eindhoven, the Netherlands**
i.kostitsyna@tue.nl
3 **University of Freiburg, Germany**
kuhn@cs.uni-freiburg.de
4 **Paderborn University, Germany**
scheideler@upb.de

Abstract

We investigate the problem of recognizing the ratio of the sides of a parallelogram by a finite-state automaton robot with pebbles operating on a triangular grid. Our goal is to determine the computational power of a single finite automaton robot in this setting with and without the help of pebbles. We demonstrate that a robot without pebbles can determine whether a given shape is a parallelogram whose sides have a ratio of h to $ah+b$ for constant integers a and b, but cannot detect whether the ratio is h to $f(h)$, where $f(x) = \omega(x)$ is a superlinear function. For a robot with a single pebble, we present an algorithm to decide whether the sides ratio is h to $p(h)$ for a given polynomial $p(\cdot)$ of constant degree. Finally, we present algorithms to decide more complex functions, such as exponential functions, using multiple pebbles.

1 Introduction

Motivated by the problem of shape recognition in restricted computational models, we study the problem of recognizing shapes of specific side ratios. We build upon the model introduced in [7] where finite-state automaton robots move on a triangular grid and assemble hexagonal tiles into various shapes.

In this paper, rather than exploring shape formation problems, we investigate the problem of *shape recognition* by a single robot. Specifically, we begin with testing whether a given tile formation is of a certain simple shape—in particular, a parallelogram—and then deciding for a given function f whether the longer side has length $f(h)$ where h is the shorter side's length. We further consider a variant of this problem where the robot is given a set of pebbles that can be used to mark certain positions. Our ultimate goal is to investigate the computational capabilities of a simple robot concerning shape recognition, and to what extent the robot can benefit from employing pebbles.

Model. Let a single *robot* be placed on a finite set of *hexagonal tiles*; each tile occupies exactly one node of an infinite triangular grid graph, as shown in Figure 1, such that the subgraph induced by all nodes occupied by tiles is connected. The robot is initially placed on a tile and carries a (possibly empty) set of *pebbles*. It can drop a pebble on a tile that is not already occupied by another pebble.

The robot acts as a *deterministic finite automaton* and operates in *look-compute-move* cycles. In the *look* phase the robot can observe the node it occupies and the six neighbors

* This work was partially supported by DFG grant SCHE 1592/3-1. Fabian Kuhn is supported by ERC Grant No. 336495 (ACDC).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

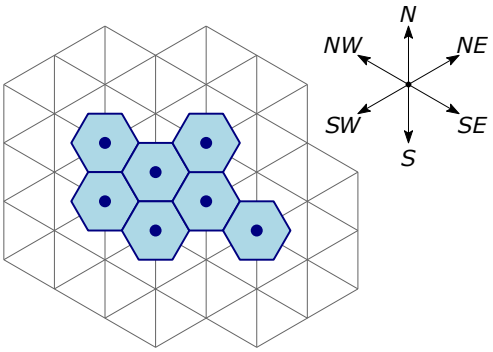


Figure 1 An exemplary tile configuration. The top right part of the figure shows the compass directions we use to describe the movement of a robot.

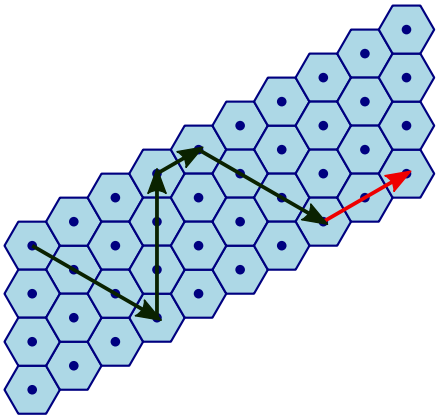


Figure 2 A N-NE-parallelogram with height 4 and length $10 = 2 \cdot 4 + 2$. The black arrows indicate the zig-zag movement of a robot as described in the proof of Theorem 3.1. The red arrow shows the final NE movement.

of that node. For each of these nodes it can determine whether it is occupied by a tile and whether a pebble is placed on that tile. In the *compute* phase the robot potentially changes its state and determines its next move according to the observed information and the number of carried pebbles. In the *move* phase the robot can either take a pebble from its current node, place a pebble it is carrying at that node, or move to an adjacent tile.

Note that even though we describe the algorithms as if the robot knew its global orientation, we do not actually require the robot to have a compass. For the algorithms presented in this paper, it is enough for the robot to be able to maintain its relative orientation with respect to its original orientation.

Related Work. To the best of our knowledge, shape recognition has never been investigated in our model. However, solving problems by traversing a tile structure with simple agents has been studied in many different areas. For instance, [14] considers the problem of deciding whether a structure is simply-connected. Other problems include Gathering and Rendezvous (e.g., [13]), Intruder Caption and Graph Searching (e.g., [2], [5]), or Black Hole Search (e.g., [12]).

For many of the above problems it has also been investigated whether pebbles can be helpful. This question is particularly well-studied for the classical Network Exploration Problem (see, e.g., [4]). For example, it is known that a finite automaton robot can neither explore all planar graphs [6] nor find its way out of a planar labyrinth [3] without any pebble. For the Labyrinth Exploration Problem (see [10, 11] for a comprehensive survey), it is known that having an additional pebble does not help the robot [8]. However, a robot with two pebbles can solve the problem [1].

2 Recognizing Simple Shapes

First, observe that a single robot can easily detect whether the initial structure is a line, a triangle, a hexagon, or a parallelogram.

Line. For example, to test if a given tile shape is a line, the robot first chooses a direction in which there is a tile (say, w.l.o.g., N), walks in that direction as far as possible (i.e., until

there is no tile in that direction anymore), and then traverses the structure into the opposite direction until no longer possible. If it ever encounters a tile to the left or right of any traversed tile, the structure is not a line.

Parallelogram. To test if a given tile shape is a (filled) parallelogram axis aligned along the directions N and NE (see Figure 2), first, the robot moves to a locally southernmost tile of the structure by moving S and SW as long as there is a tile in any of these directions. It then traverses the shape column by column in a snake-like fashion by repeating the following movements: First, the robot moves N as far as possible; it then moves one step NE; then it moves S as far as possible, and it finally moves one step NE. The above procedure is repeated until a NE movement is impossible. By performing local checks alongside the movements described above the robot can verify whether the tile shape is a parallelogram.

The other simple shapes can be easily tested in a similar fashion.

▶ Observation 2.1. A robot without any pebble can detect whether the initial tile configuration is a line, a triangle, a hexagon, or a parallelogram.

3 Recognizing Parallelograms with Specific Side Ratio

As noted in Observation 2.1, a single robot without pebbles can verify whether a given shape is a parallelogram. To investigate the computational power of a finite automaton, in this section we consider the problem of deciding whether a parallelogram has a given side ratio. Additionally, we examine how pebbles can be helpful to decide more complex side ratios.

We assume w.l.o.g. that the robot needs to detect whether the given tile configuration is a parallelogram that is axis-aligned along the north and north-east direction (we say *N-NE-parallelogram*), as indicated in Figure 2. We denote a maximal sequence of consecutive tiles from N to S as a *column* and a maximal sequence of consecutive tiles from SW to NE as a *row*. Let h be the size of each column, i.e., the parallelogram's *height*, ℓ be the size of each row, i.e., the parallelogram's *length*, and let $h \leq \ell$. We number the columns of the parallelogram from 0 to $\ell - 1$ growing in the north-eastern direction.

3.1 A Robot without any Pebble

First, we point out that a single robot can detect whether the structure is a parallelogram in which its length ℓ is a linear function of its height h.

▶ **Theorem 3.1.** *A single robot can detect whether the tile configuration is a parallelogram with $\ell = ah + b$ for any constants $a, b \in \mathbb{N}$.*

Proof. First, the robot verifies whether the structure is a parallelogram. If so, the robot moves to the northernmost tile of the column 0. It then traverses the tile structure in two stages to verify the ratio of the sides. In the first stage the robot "measures" the distance ah along the length of the parallelogram moving in a zig-zag fashion as depicted in Figure 2. In the second stage the robot measures the second term b. More specifically, in the first stage, the robot repeats the following movements in a loop: (1) move SE as far as possible, (2) move N as far as possible, and (3) make one step NE. After having performed the complete sequence of SE movements a times, the robot moves on to the second stage, in which it makes an additional b NE steps.

If the robot reaches the easternmost column before completing the above procedure, or finally halts on a tile with a neighboring tile at NE, it terminates with a negative result. Otherwise, it terminates with a positive result. It is easy to see that $\ell = ah + b$ if and only if the robot terminates with a positive result. ◀

▶ Remark. The algorithm in the previous theorem can be adapted for $b \in \mathbb{Z}$, i.e., b can also be a negative integer. To achieve that, we halt the execution of the first stage once the robot has performed $|b|$ *SE* steps, then move *SW* as far as possible, and continue the first stage from there. Then, $\ell = ah + b$ if and only if the robot eventually reaches the southernmost tile of the easternmost column.

▶ Remark. The algorithm can be further extended to apply in the case of a rational a. Let $a = p/q$ be an irreducible fraction. Instead of moving in a zig-zag fashion in the first stage, the robot alternates between moving p steps *NE* and q steps *S*. To exactly end up at the southernmost tile of the easternmost column, the robot needs to skip the very first *NE* and *S* step.

We have shown that a single robot can determine whether the length of a parallelogram is given by a certain linear function of its height. However, that is as much as one robot can hope for. Indeed, one robot is not able to decide whether the length of the parallelogram is given by a superlinear function of its height, as the following theorem states. Its proof can be found in the full version of this paper.

▶ **Theorem 3.2.** *A single robot without any pebbles cannot decide whether the tile configuration is a parallelogram with $\ell = f(h)$, where $f(x) = \omega(x)$.*

3.2 A Robot with a Single Pebble

In the following, we demonstrate that, in contrast to the negative result of Theorem 3.2, a single robot can decide any polynomial of constant degree.

▶ **Theorem 3.3.** *A single robot with a pebble can decide whether the tile configuration is a parallelogram of height h and length $\ell = p(h)$ for any given polynomial $p(\cdot)$ of constant degree n.*

Proof. Define the *falling factorial* of x as $(x)_i := x(x-1)\cdots(x-i+1)$, and transform the input polynomial into the form $p(x) = a_n \cdot (x)_n + a_{n-1} \cdot (x)_{n-1} + \ldots + a_0$. We will show that the robot can move the pebble in phases, by $|a_i \cdot (h)_i|$ steps in each phase i. Let $\mathrm{lcm}_i(x) := \mathrm{lcm}(x, \ldots, x-i+1)$, where lcm is the *least common multiple*, and $g_i(x) := (x)_i / \mathrm{lcm}_i(x)$. From [9] it follows that $\mathrm{lcm}_i(x) \mid (x)_i$, and that $g_i(x)$ is periodic with period $\mathrm{lcm}(1, \ldots, i-1)$, i.e., $g_i(x) = g_i(x + \mathrm{lcm}(1, \ldots, i-1))$. Let $p_i(x)$ be the sum of the first $n-i$ summands of $p(x)$, i.e., $p_i(x) = a_n \cdot (x)_n + a_{n-1} \cdot (x)_{n-1} + \ldots + a_{n-i+1} \cdot (x)_{n-i+1}$.

Initially, the pebble is located on the northernmost tile of column 0. To test whether $\ell = p(h)$, the robot will move the pebble along the northernmost row in phases, until eventually it is shifted $p(h) - 1$ steps to the *NE* from its original position. If upon termination the pebble is located at the northernmost tile of column $\ell - 1$, then $p(h) = \ell$.

The algorithm proceeds in phases $n, \ldots, 0$. We maintain the invariant that after phase i for all $i > 0$, the pebble is located at the northernmost tile of column $p_i(h)$. That is, in phase i, the robot moves the pebble $|a_i \cdot (h)_i|$ steps *NE*, if a_i is positive, and *SW*, otherwise. In the final phase $i = 0$, the robot moves the pebble by $|a_0 - 1|$ steps *NE*, if $a_0 \geq 1$, and *SW*, otherwise. For now, assume that each movement can be carried out without moving the pebble outside of the parallelogram. We will later describe how to lift this restriction.

We now describe how the pebble is moved by $|a_i \cdot (h)_i|$ steps. First, note that $a_i \cdot (h)_i = a_i \cdot g_i(h) \cdot \mathrm{lcm}_i(h)$. The first factor a_i is a constant. The second factor $g_i(h)$ can be determined as follows. We preliminarily encode all possible values of $g_i(\cdot)$ for all $i \in \{0, \ldots, n\}$ into the robot's memory, which can be done since n is constant and $g_i(\cdot)$ has a constant period. Before the main algorithm's execution, the robot can compute $g_i(h)$ for all i by moving

through the westernmost column from north to south: Starting with $g_i(1)$, in every step to the south the robot computes the subsequent function value until the period of $g_i(\cdot)$ is reached, in which case it restarts with $g_i(1)$. When it reaches the southernmost tile of the column, it knows $g_i(h)$ for all i.

We next show how the robot moves the pebble by $\text{lcm}_i(h)$ steps, which, by repeating the movement $|a_i \cdot g_i(h)|$ times, concludes how the complete movement by $|a_i \cdot (h)_i|$ steps is performed. Assume the pebble is in some column c and $\text{lcm}_i(h) \mid c$ (which we will prove by induction shortly). The robot alternates between the following two operations: (1) move the pebble into column c' by moving it one step *NE*, if $a_i > 0$, or *SE*, otherwise; (2) verify whether $\text{lcm}_i(h) \mid c'$ as follows. The robot first performs the zig-zag movement from the proof of Theorem 3.1 to verify whether $h \mid c'$, i.e., whether a *NE* movement moves the robot onto a tile occupied by the pebble. It continues to analogously verify whether $h-1 \mid c'$, $h-2 \mid c'$, $\ldots$, $h-i+1 \mid c'$ by performing a modified zig-zag movement an additional $i-1$ times. Here, the zig-zags of the j-th verification are adjusted accordingly by moving j steps S prior to each sequence of *SE* movements. The robot stops alternating between the two above operations once the pebble has been moved to a column c' such that $\text{lcm}_i(h) \mid c'$ for the first time. Then, the pebble must have been moved by $\text{lcm}_i(h)$ steps.

It remains to prove that when the robot wants to move the pebble, currently occupying a node of column c, by $\text{lcm}_i(h)$ steps for some i, then $\text{lcm}_i(h) \mid c$. The invariant holds initially for $c = 0$. Now assume it holds immediately after having moved the pebble by $\text{lcm}_i(h)$ steps into column $c \pm \text{lcm}_i(h)$. Afterwards, the robot can move it by either $\text{lcm}_i(h)$ steps again, in which case $\text{lcm}_i(h) \mid c \pm \text{lcm}_i(h)$ holds, or it wants to move it by $\text{lcm}_{i-1}(h)$ steps, and $\text{lcm}_{i-1}(h) \mid c \pm \text{lcm}_i(h)$ holds since $\text{lcm}_{i-1}(h) \mid \text{lcm}_i(h)$.

Finally, we show how the robot can resolve *overflows*, i.e., situations in which the above algorithm would move the pebble outside of the parallelogram. First, note that the execution of the algorithm after an overflow can, in principle, be continued by the robot by "mirroring" all movements beyond the westernmost or easternmost column, carrying them out into reverse direction. Assume that h is sufficiently large such that $\mid a_i \cdot (h)_i + \ldots + a_0 \mid \leq p(h)$ for all i. For all small $h = O(\max_i(|a_i|))$ we can encode the constantly many possible function values into the robot's state and test them prior to the algorithm's execution by traversing the two sides of the parallelogram once. If throughout the execution of the algorithm the robot ever attempts to move the pebble into a column west of column 0 or east of "virtual" column 2ℓ (while performing the mirroring method from above), it would subsequently not be able to ever move the pebble back into column ℓ (following from the assumption that h is sufficiently large), and consequently $\ell \neq p(h)$. Therefore, the robot can prematurely terminate with a negative result whenever it encounters such a situation. ◀

In contrast to the previous theorem, for a fixed k, we believe that a polynomial $p(\cdot)$ of degree $\omega(k)$ can be constructed such that a single robot with k states and a pebble cannot decide whether $\ell = p(h)$.

3.3 A Robot with Multiple Pebbles

Finally, we show that having multiple pebbles enables the robot to decide some more complex functions.

▶ **Theorem 3.4.** *A robot with two pebbles can decide whether the tile configuration is a parallelogram with $\ell = 2^h$.*

Proof. We only provide a high-level idea of the algorithm and omit some of the implementation details. We call the pebbles a and b. Pebble a will always be in the northernmost row of

the parallelogram. The robot first places pebble a at the northernmost tile of column 1, and b on the southern neighbor of a. It then repeatedly performs the following procedure: First, the robot alternatingly moves a *NE* and b *SW* until there is no tile *SW* from b. Afterwards, the robot brings b back into the current column of a by repeatedly moving b *NE* and verifying whether a is in the northernmost tile of the column. It then moves b one step *S*.

In the i-th iteration of the above procedure, pebble a gets moved 2^{i-1} times. Thus, the distance between a and the westernmost column doubles in each iteration. By moving b one step *S*, the robot counts the number of doubling operations up to h. If a has reached the easternmost column after h doubling operations, then $\ell = 2^h$. ◀

The proof of the following theorem can be found in the full version of this paper.

▶ **Theorem 3.5.** *A robot with three pebbles can decide whether a given tile configuration is a parallelogram with*

$$\ell = 2^{2^{\cdot^{\cdot^{\cdot^{2^h}}}}},$$

where the power tower is of constant height.

▶ Remark. It can be shown that a single robot can also decide a power tower of height h.

References

1 M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, 1978.
2 A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.
3 L. Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86(1):195–282, 1978.
4 S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the EATCS*, 109:54–69, 2013.
5 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, jun 2008.
6 P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.
7 R. Gmyr, I. Kostitsyna, F. Kuhn, C. Scheideler, and T. Strothmann. Forming tile shapes with a single robot. In *Abstr. European Workshop on Computational Geometry (EuroCG)*, pages 9–12, 2017.
8 F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Proc. International Fundamentals of Computation Theory Conference (FCT)*, pages 433–444, 1981.
9 S. Hong and Y. Yang. On the periodicity of an arithmetical function. *Comptes Rendus Mathematique*, 346(13):717–721, 2008.
10 G. Kilibarda, V. B. Kudryavtsev, and Š. Uščumlić. Collectives of automata in labyrinths. *Discrete Mathematics and Applications*, 13(5):429–466, 2003.
11 G. Kilibarda, V. B. Kudryavtsev, and Š. Uščumlić. Independent systems of automata in labyrinths. *Discrete Mathematics and Applications*, 13(3):221–225, 2003.
12 E. Markou. Identifying hostile nodes in networks using mobile agents. *Bulletin of the EATCS*, 108:93–129, 2012.
13 A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.
14 A. N. Shah. Pebble automata on arrays. *Computer Graphics and Image Processing*, 3(3):236–246, 1974.

Generalized kernels of polygons under rotation*

David Orden[1], Leonidas Palios[2], Carlos Seara[3], and Paweł Żyliński[4]

1 **Departamento de Física y Matemáticas, Universidad de Alcalá, Spain**
david.orden@uah.es
2 **Dept. of Computer Science and Engineering, University of Ioannina, Greece**
palios@cs.uoi.gr
3 **Mathematics Department, Universitat Politècnica de Catalunya, Spain**
carlos.seara@upc.edu
4 **Institute of Informatics, University of Gdańsk, Poland**
zylinski@inf.ug.edu.pl

Abstract

Given a set $\mathcal{O}$ of k orientations in the plane, two points inside a simple polygon P *$\mathcal{O}$-see each other* if there is an $\mathcal{O}$-staircase contained in P that connects them. The $\mathcal{O}$-kernel of P is the subset of points which $\mathcal{O}$-see all the other points in P. This work initiates the study of the computation and maintenance of the $\mathcal{O}$-Kernel of a polygon P as we rotate the set $\mathcal{O}$ by an angle θ, denoted $\mathcal{O}$-Kernel$_\theta(P)$. In particular, we design efficient algorithms for (i) computing and maintaining $\{0^\circ\}$-Kernel$_\theta(P)$ while θ varies in $[-\frac{\pi}{2}, \frac{\pi}{2})$, obtaining the angular intervals where the $\{0^\circ\}$-Kernel$_\theta(P)$ is not empty and (ii) for orthogonal polygons P, computing the orientation $\theta \in [0, \frac{\pi}{2})$ such that the area and/or the perimeter of the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ are maximum or minimum. These results extend previous works by Gewali, Palios, Rawlins, Schuierer, and Wood.

1 Introduction

The problem of computing the kernel of a polygon is a well-known visibility problem in computational geometry [2, 3, 6], closely related to the problem of guarding a polygon and to the motion problem of a robot inside a polygon with the restriction that the robot path must be *monotone* in some predefined set of orientations [5, 7, 8]. The present contribution goes a step further in the latter setting, by allowing that set of predefined orientations to rotate.

A curve $\mathcal{C}$ is $0°$-*convex* if its intersection with any $0°$-line (parallel to the x-axis) is connected or, equivalently, if the curve $\mathcal{C}$ is y-monotone. Extending this definition, a curve $\mathcal{C}$ is α-*convex* if the intersection of $\mathcal{C}$ with any line forming angle α with the x-axis is connected or, equivalently, if the curve $\mathcal{C}$ is monotone with respect to the direction perpendicular to such a line. Let us now consider a set $\mathcal{O}$ of k orientations in the plane, each of them given by an oriented line ℓ_i through the origin of the coordinate system and forming a counterclockwise angle α_i with the positive x-axis, so that $\mathcal{O} = \{\alpha_1, \ldots, \alpha_k\}$. Then, a curve is $\mathcal{O}$-*convex* if it is α_i-convex for all i, $1 \leq i \leq k$. Notice that the orientations in $\mathcal{O}$ are between 0° and 180°. From now on, an $\mathcal{O}$-convex curve will be called an $\mathcal{O}$-*staircase*.

▶ **Definition 1.1.** Let p and q be points inside a simple polygon P. We say that p and q *$\mathcal{O}$-see each other* or that they are *$\mathcal{O}$-visible from each other* if there is an $\mathcal{O}$-staircase contained in P that connects p and q. The *$\mathcal{O}$-Kernel* of P, denoted $\mathcal{O}$-Kernel(P), is the subset of points

* David Orden is supported by MINECO projects MTM2014-54207, MTM2017-83750-P, and by UE H2020-MSCA-RISE project 734922-CONNECT. Carlos Seara is supported by Gen. Cat. project DGR 2014SGR46, by MINECO project MTM2015-63791-R, and by UE H2020-MSCA-RISE project 734922-CONNECT. Paweł Żyliński is supported by the grant 2015/17/B/ST6/01887 (National Science Centre, Poland).

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

in P which $\mathcal{O}$-see all the other points in P. We denote by $\mathcal{O}$-Kernel$_\theta(P)$ the $\mathcal{O}$-kernel when the set $\mathcal{O}$ is rotated by an angle θ.

Schuierer, Rawlins, and Wood [7] defined the restricted orientation visibility or *$\mathcal{O}$-visibility* in a simple polygon P with n vertices, giving an algorithm to compute the $\mathcal{O}$-Kernel(P) in time $O(k + n \log k)$ with $O(k \log k)$ preprocessing time to sort the set $\mathcal{O}$ of k orientations. In order to do so, they used the following observation:

▶ **Observation 1.2.** *For any simple polygon P, the $\mathcal{O}$-*Kernel(P) *is $\mathcal{O}$-convex and connected. Furthermore, $\mathcal{O}$-*Kernel$(P) = \bigcap_{\alpha_i \in \mathcal{O}} \alpha_i$-Kernel$(P)$.

The computation of the $\mathcal{O}$-Kernel has been considered by Gewali [1] as well, who described an $O(n)$-time algorithm for orthogonal polygons without holes and an $O(n^2)$-time algorithm for orthogonal polygons with holes. The problem is a special case of the one considered by Schuierer and Wood [9] whose work implies an $O(n)$-time algorithm for orthogonal polygons without holes and an $O(n \log n + m^2)$-time algorithm for orthogonal polygons with $m \geq 1$ holes. More recently, Palios [5] gave an output-sensitive algorithm for computing the $\mathcal{O}$-Kernel of an n-vertex orthogonal polygon P with m holes, for $\mathcal{O} = \{0^\circ, 90^\circ\}$. This algorithm runs in $O(n + m \log m + \ell)$ time where $\ell \in O(1 + m^2)$ is the number of connected components of the $\{0^\circ, 90^\circ\}$-Kernel(P). Additionally, a modified version of this algorithm computes the number ℓ of connected components of the $\{0^\circ, 90^\circ\}$-Kernel in $O(n + m \log m)$ time.

2 The rotated $\{0^\circ\}$-Kernel$_\theta(P)$

In this section, for a given n-vertex simple polygon P, we deal with the rotation by $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2})$ of the $\mathcal{O}$-Kernel(P) in the particular case of $\mathcal{O} = \{0^\circ\}$. For the steady case $\theta = 0$, when the kernel is composed by the points which see every point in P via a y-monotone curve, Schuierer, Rawlins, and Wood [7] presented the following results.

▶ **Definition 2.1.** A reflex vertex p_i in P where p_{i-1} and p_{i+1} are both below (respectively, above) p_i is a *reflex maximum* (respectively, *reflex minimum*). A horizontal edge with two reflex vertices may also form a reflex maximum or minimum.

Let $C(P)$ be the (infinite) strip defined by the horizontal lines h_N and h_S passing through the lowest reflex minimum, p_N, and the highest reflex maximum, p_S.

▶ **Lemma 2.2** ([7])**.** *The* $\{0^\circ\}$-Kernel(P) *is the intersection $C(P) \cap P$.*

▶ **Observation 2.3.** *Notice that P may not have a reflex maximum, and in that case we have to take as the highest reflex maximum the lowest vertex of P. Analogously, if P has no reflex minimum, then we take as the lowest reflex minimum the highest vertex of P. Notice also that a horizontal edge of P may also form a reflex maximum or minimum. If P is a convex polygon then the* $\{0^\circ\}$-Kernel(P) *is the whole P.*

▶ **Corollary 2.4** ([7])**.** *The* $\{0^\circ\}$-Kernel(P) *can be computed in $O(n)$ time.*

It is clear that there are neither reflex minima nor reflex maxima within $C(P)$. Moreover, the lines h_N and h_S contain the segments of the *north* and *south* boundary of the $\{0^\circ\}$-Kernel(P). See Figure 1, left.

Let c^l and c^r denote the left and the right polygonal chains defined by the respective left and right parts of the boundary of P inside $C(P)$. Because of Lemma 2.2, both chains are 0°-convex curves, i.e., y-monotone chains.

▶ **Corollary 2.5.** *The area and perimeter of* $\{0^\circ\}$-Kernel(P) *can be computed in* $O(n)$ *time.*

Proof. To compute the area of the $\{0^\circ\}$-Kernel$(P) = C(P) \cap P$ we proceed as follows. The lines h_N and h_S contain the segments of the north and south boundary of the $\{0^\circ\}$-Kernel(P). The area can be computed by decomposing it into (finite) horizontal strips, which in fact are trapezoids defined by the edges of P inside $C(P)$ in such a way that correspond to consecutive vertices from the merging of the sorted list of the vertices of either c^l or c^r. Each strip is computed in constant time, and therefore, the area of $\{0^\circ\}$-Kernel$(P) = C(P) \cap P$ can be computed in $O(|c^l| + |c^r|)$ time.

Computing the perimeter is simpler, because we only need to add the lengths of both chains c^l and c^r and the lengths of the sides of the north and south boundary of the $\{0^\circ\}$-Kernel(P); this requires computing the intersection of the lines h_N and h_S with the boundary of P and can be done in $O(|c^l| + |c^r|)$ time. ◀

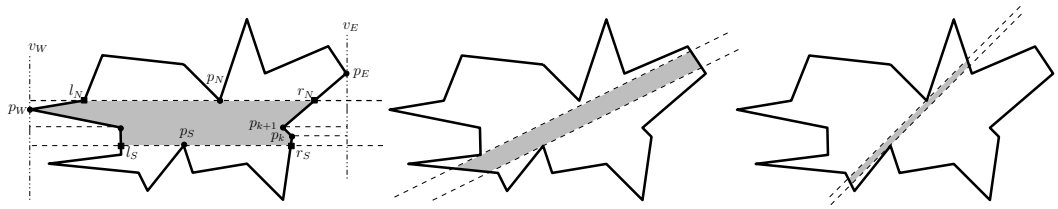


Figure 1 A rotating $\{0°\}$-Kernel$_\theta(P)$ for $\theta = 0$ (left), $\theta = \frac{\pi}{8}$ (middle), and $\theta = \frac{\pi}{4}$ (right).

Next, we will compute the $\{0^\circ\}$-Kernel$_\theta(P)$ as θ varies from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. To do that, first we need to maintain the strip boundary $C_\theta(P)$ during the variation of θ, i.e., the equations of the two horizontal lines $h_N(\theta)$ and $h_S(\theta)$ which contain the current horizontal sides of $\{0^\circ\}$-Kernel$_\theta(P)$, in such a way that for a given value of θ the boundary of $C_\theta(P)$ can be computed from these equations. Second, we also need to maintain the set of vertices of the left and right chains c^l_θ and c^r_θ of $\{0^\circ\}$-Kernel$_\theta(P)$.

First, we observe that Definition 2.1 of a reflex maximum and a reflex minimum, with respect to the horizontal orientation, extends easily for any given orientation $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2})$. Then, for every reflex vertex $p_i \in P$ we compute in constant time the angular interval $(\theta^i_1, \theta^i_2) \subset [-\frac{\pi}{2}, \frac{\pi}{2})$ of the orientations such that $p_i \in P$ is a candidate reflex maximum or reflex minimum. Finally, in constant time we compute the corresponding *slope intervals* $(m^i_1, m^i_2) \subset (-\infty, \infty)$ for the lines with these orientations. Using these to sweep in the dual plane [4], we get the following result, where $\alpha(n)$ is the extremely slowly-growing inverse of Ackermann's function and $O(n\,\alpha(n))$ comes from the complexity of the upper and lower envelopes of a set of n straight line segments in the plane:

▶ **Theorem 2.6.** *For an* n*-vertex simple polygon* P*, there are* $O(n\,\alpha(n))$ *angular intervals* $(\theta_1, \theta_2) \subset [-\frac{\pi}{2}, \frac{\pi}{2})$ *such that* $\{0^\circ\}$-Kernel$_\theta(P) \neq \emptyset$ *for all the values of* $\theta \in (\theta_1, \theta_2)$*, and the set of such intervals can be computed in* $O(n \log n)$ *time.*

▶ **Observation 2.7.** *To maintain the polygonal chains* c^l_θ *and* c^r_θ *of* $\{0^\circ\}$-Kernel$_\theta(P)$ *we compute the intersections of the lines* $h_N(\theta)$ *and* $h_S(\theta)$ *with the boundary of* P*, maintaining the information of the first and the last vertices of* c^l_θ *and* c^r_θ *in the current interval* (θ_1, θ_2)*.*

3 The rotated $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ of simple orthogonal polygons

We now extend our study to $\mathcal{O} = \{0^\circ, 90^\circ\}$ for the particular case of orthogonal polygons, where it suffices to consider $\theta \in [0, \frac{\pi}{2})$.

Each edge of an orthogonal polygon is a N-*edge*, S-*edge*, E-*edge*, or W-*edge* if it bounds the polygon from the north, south, east, or west, respectively. For $D \in \{\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}\}$, a D-edge is a D-*dent* (resp., D-*extremity*) if both of its endpoints are reflex (resp., convex) vertices of the polygon. Next, a NE-*staircase* is a sequence of alternating N- and E-edges and similarly we can define the NW-*staircase*, SE-*staircase*, and SW-*staircase*; clearly, each such staircase is both x- and y-monotone. Finally, depending on the type of incident edges, a *reflex* vertex of the given orthogonal polygon P is called a NE-*reflex vertex* if it is incident to a N-edge and an E-edge, and analogously for NW-, SE- and SW-*reflex vertices*. See Fig. 2(a).

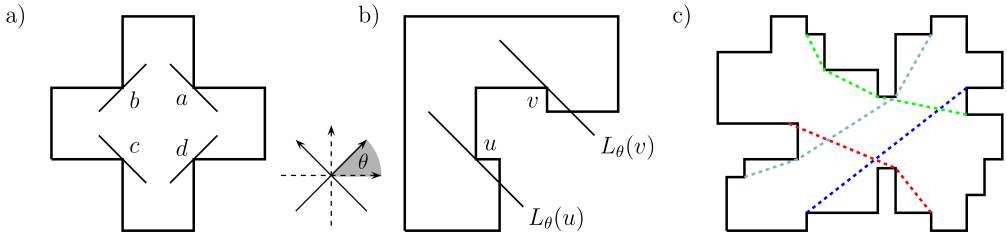


Figure 2 (a) Clipping lines through the reflex vertices of the polygon, where a, b, c, d are respectively a NE-, NW-, SW-, and SE-reflex vertex. (b) Illustration for Lemma 3.2. (c) An orthogonal polygon belonging to the family $\mathcal{Q}$.

The size of the kernel. For $\theta = 0$, in accordance with Observation 1.2 and Lemma 2.2, the $\{0^\circ, 90^\circ\}$-Kernel$_0(P)$ equals the intersection of P with the four *closed* halfplanes defined below the lowermost N-dent, above the topmost S-dent, right of the rightmost W-dent, and left of the leftmost E-dent, if such dents exist; thus, the $\{0^\circ, 90^\circ\}$-Kernel$_0(P)$ can be computed in $O(n)$ time. For $\theta \in (0, \frac{\pi}{2})$, the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ can be obtained by clipping P about appropriate lines; it turns out that *every* reflex vertex contributes such a clipping line. In particular, each NW- or SE-reflex vertex contributes a clipping line parallel to 0°_θ whereas each NE- or SW-reflex vertex contributes a clipping line parallel to 90°_θ, where α_θ denotes the rotation of the orientation α by the angle θ; see Fig. 2(a). Thus, if there exist reflex vertices of all four types, the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ has two pairs of sides, with one pair parallel to each of the rotated orientations. Moreover, an extremity may contribute an edge to the kernel; however, we can prove that only one extremity from each of the four kinds may do so [4]. On the other hand, if the polygon has no NE-reflex vertices then there is a unique N-extremity and a unique E-extremity, which share a common endpoint, and then the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ has no additional edges in such a case. Similar results hold in the cases of absence of NW-reflex, SW-reflex, or SE-reflex vertices. Therefore, we obtain:

▶ **Lemma 3.1.** *The rotated* $\{0°, 90°\}$*-Kernel*$_\theta(P)$ *has at most eight edges.*

In order to compute the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ for each value of θ, we need to collect the sets of the four types of reflex vertices of P and compute the convex hull of each of them, which will give us the corresponding minimum or maximum. We can show the following ($v.x$ and $v.y$ denote the x- and y-coordinates of a vertex v, respectively):

▶ **Lemma 3.2.** *Let P be an orthogonal polygon. If u and v are a* NE*-reflex and a* SW*-reflex vertex of P, respectively, such that $v.x \geq u.x$ and $v.y \geq u.y$ (see Fig.* 2(b)*), then for each $\theta \in (0, \frac{\pi}{2})$, the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ is empty. Similarly, if u', v' are a* NW*-reflex and a* SE*-reflex vertex of P, respectively, such that $v'.x \leq u'.x$ and $v'.y \geq u'.y$, then for each $\theta \in (0, \frac{\pi}{2})$, the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ is empty.*

Proof. For each $\theta \in (0, \frac{\pi}{2})$, let $L_\theta(u), L_\theta(v)$ be the lines forming angle $90^\circ + \theta$ with the positive x-axis that are tangent at u, v, respectively (see Fig. 2(b)). Clearly, $L_\theta(u)$ is below and to the left of $L_\theta(v)$. This leads to a contradiction since the kernel can neither be higher than $L_\theta(u)$ nor lower than $L_\theta(v)$. ◀

Next, let $\mathcal{Q}$ be the family of simple orthogonal polygons whose boundary (in ccw traversal) consists of the concatenation of (i) a y-monotone chain from the lowermost E-extremity to the topmost E-extremity, (ii) a NE-staircase to the rightmost N-extremity, (iii) an x-monotone chain to the leftmost N-extremity, (iv) a NW-staircase to the topmost W-extremity, (v) a y-monotone chain to the lowermost W-extremity, (vi) a SW-staircase to the leftmost S-extremity, (vii) an x-monotone chain to the rightmost S-extremity, and (viii) a SE-staircase to the lowermost E-extremity. See Fig. 2(c). Note that each pair of extremities and the monotone chain between them may degenerate into a single extremity, whereas each staircase may degenerate into a 2-edge chain.

▶ **Corollary 3.3.** *If $P \notin \mathcal{Q}$, then the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ is empty for each $\theta \in (0, \frac{\pi}{2})$.*

Proof. Let $\mathcal{E}$ (resp., $\mathcal{N}$) be the subset of E-extremities (resp., N-extremities) such that $e \in \mathcal{E}$ (resp., $e \in \mathcal{N}$) iff for every point p of $e \in \mathcal{E}$ (resp., $e \in \mathcal{N}$), the rightward horizontal ray (resp., upward vertical ray) from p does not intersect the interior of P; clearly, $\mathcal{E} \neq \emptyset$ and $\mathcal{N} \neq \emptyset$.

If the ccw chain of edges from the topmost E-extremity e in $\mathcal{E}$ to the rightmost N-extremity e' in $\mathcal{N}$ contains a S-edge, and let e'' be the first encountered S-edge from e to e', then the rightmost edge in the ccw boundary chain from e'' to e' is an E-extremity in $\mathcal{E}$ higher than e, whereas if it contains an W-edge then the edge preceding the first-encountered W-edge in the ccw boundary chain from e to e' is a N-extremity in $\mathcal{N}$ to the right of e'; a contradiction. Hence, the ccw boundary chain from e to e' is a NE-staircase. Next, we show that if the ccw chain of edges from the lowermost E-extremity e_1 in $\mathcal{E}$ to the topmost E-extremity e_2 in $\mathcal{E}$ is not y-monotone, then the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ is empty for each $\theta \in (0, \frac{\pi}{2})$. For contradiction, suppose that this chain is not y-monotone. Then, it contains an W-edge; let e be the first encountered such edge. Two cases are possible; see Figure 3.

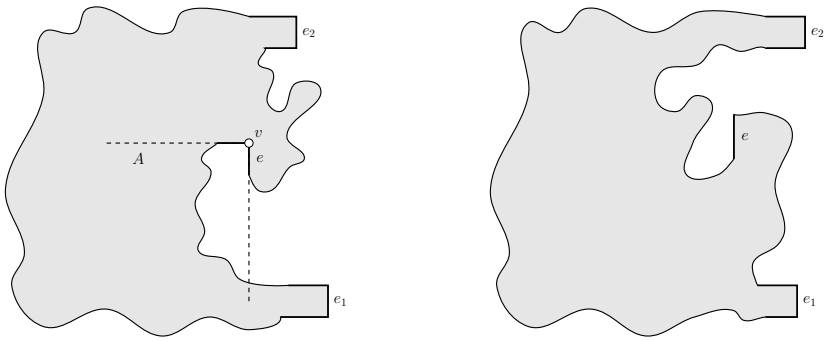


Figure 3 The two possible cases for the proof of Corollary 3.3.

In the case shown on the left in Figure 3, the edge preceding e is a S-edge and their common endpoint v is a SW-reflex vertex. Let A be the bottom-left quadrant defined by the lines supporting the edges incident on v and let ρ be the part of the boundary chain from e_1 to e that belongs to A. Then, the bottom vertex of the leftmost edge in ρ is a NE-reflex vertex which is to the left and below v, and Lemma 3.2 implies that the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ is empty for each $\theta \in (0, \frac{\pi}{2})$, as desired. The case shown on the right in Figure 3 is a top-down mirror image of the case shown on the left; hence, the same result holds also for this case. ◀

Algorithm. Corollary 3.3 shows that the $\{0^\circ, 90^\circ\}$-Kernel$_\theta$ of orthogonal polygons not in $\mathcal{Q}$ is empty for all $\theta \in (0, \frac{\pi}{2})$. For those in $\mathcal{Q}$, their particular form allows the efficient computation of the kernel. In particular, for any polygon $P \in \mathcal{Q}$, a single traversal of the vertices of P allows us to compute the maxima of the SW-reflex vertices of P in linear time. Indeed, the SW-reflex vertices of such a polygon P belong to the ccw boundary chain from the topmost W-extremity to the rightmost S-extremity; then, both the maxima in the y-monotone chain from the topmost W-extremity to the leftmost S-extremity and the maxima in the x-monotone chain from the leftmost S-extremity to the rightmost S-extremity can be computed in linear time, and the two maxima sequences can be merged in linear time as well. Then, the cw part of the boundary of their convex hull from its topmost to its rightmost vertex (the red dashed line in Fig. 2(c)), which is the useful one, can also be computed in linear time by using the iterative step of the Graham scan algorithm. In a similar fashion, we can compute in linear time the useful boundary parts of the convex hulls of the other three types of reflex vertices; see the dashed lines in Fig. 2(c).

Since the $\{0°, 90°\}$-Kernel$_\theta(P)$ for each angle $\theta \in [0, \frac{\pi}{2})$ is mainly determined by the, at most, 4 reflex vertices through which clipping lines are passing (Lemma 3.1), the algorithm needs to maintain these vertices. Therefore, for each edge of each of the at most 4 convex hulls, we compute the corresponding angle $\theta \in [0, \frac{\pi}{2})$ and we associate it with the edge's endpoints, and these angles are sorted by merging the 4 ordered lists of angles. Then, for each pair of consecutive angles $\theta_{prev}, \theta_{cur}$, we have the reflex vertices through which the clipping lines pass for any $\theta \in [\theta_{prev}, \theta_{cur})$, and we compute the at most 8 vertices of the kernel parameterized on θ. Next, we maximize the kernel's area/perimeter for $\theta \in [\theta_{prev}, \theta_{cur})$, and if needed, we update the overall maximum and store the corresponding angle. Thus, we have:

▶ **Theorem 3.4.** *Given a simple orthogonal polygon P on n vertices, the value of the angle $\theta \in [0, \frac{\pi}{2})$ such that the $\{0^\circ, 90^\circ\}$-Kernel$_\theta(P)$ has maximum (or minimum) area/perimeter can be computed in $O(n)$ time and space.*

References

1 L.P. Gewali. Recognizing s-Star Polygons. *Pattern Recognition*, 28(7), 1995, 1019–1032.
2 C. Icking and R. Klein. Searching for the kernel of a polygon—A competitive strategy. *Proceedings of the 11th Annual Symposium on Computational Geometry*, 1995, 258–266.
3 D. T. Lee and F. P. Preparata. An Optimal Algorithm for Finding the Kernel of a Polygon. *Journal of the ACM*, 26(3), 1979, 415–421.
4 D. Orden, L. Palios, C. Seara, and P. Żyliński. Generalized kernels of polygons under rotation. Preprint, arXiv:1802.05995 [cs.CG]
5 L. Palios. An output-sensitive algorithm for computing the s-kernel. *27th Canadian Conference on Computational Geometry*, Kingston, 2015, 199–204.
6 L. Palios. A New Competitive Strategy for Reaching the Kernel of an Unknown Polygon. *7th Scandinavian Workshop on Algorithm Theory*, 2000, 367–382.
7 S. Schuierer, G. J. E. Rawlins, and D. Wood. A generalization of staircase visibility. *3rd Canadian Conference on Computational Geometry*, Vancouver, 1991, 96–99.
8 S. Schuierer and D. Wood. Generalized kernels of polygons with holes. *5th Canadian Conference on Computational Geometry*, Waterloo, 1993, 222–227.
9 S. Schuierer and D. Wood. Multiple-guards kernels of simple polygons. *Theoretical Computer Science Center Research Report HKUST-TCSC-98-06*, 1998.

Generalized Visibility Kernel

Eyüp Serdar Ayaz[1] and Alper Üngör[1]

1 **CISE Department, University of Florida, Gainesville**
[ayaz,ungor]@cise.ufl.edu

Abstract

We propose a generalization of the concept visibility kernel which finds use in art gallery problems. For a point p in a simple polygon P and a subset X of P, we refer the set of points visible by every point in X that is seen by p as the generalized visibility kernel of p with respect to X. We present an $O(n + m \log m)$ time algorithm for computing the generalized visibility kernel, where n, m are the complexities of P and X respectively. As essential components of our approach, we also propose two efficient algorithms for computing the relative convex hull and the complete visibility polygon of a set of points.

1 Introduction

We consider a generalized version of the well known Art Gallery Problem. For a polygon P, and two subsets of it, $X, Y \subseteq P$, AGP(X, Y) asks to find the minimum subset of X as guard locations, such that all the points of Y are guarded [4]. A recent approach for solving art gallery problems involves use of witness sets [2–4]. A *witness set* W of a polygon P is defined as a set such that any set G that guards W also guards P. In [1], we extended the notion of witness sets for the parametrized version AGP(X, Y) and outlined an iterative refinement scheme for the art gallery problem. Formulation and computation of visibility kernels turns out to be an essential step of this refinement scheme. For a point $p \in P$, the set of points visible by every point in P that is seen by p is called the *visibility kernel* of p [3], denoted as $\mathcal{VK}(p)$. We present a generalization of this concept. For a point $p \in P$ and $X \subseteq P$, the set of points visible by every point in X that is seen by p is called the *generalized visibility kernel* of p with respect to X, denoted as $\mathcal{GVK}(p, X)$. Note that $\mathcal{VK}(p) = \mathcal{GVK}(p, P)$. While this generalization is seemingly simple, several algorithmic challenges arise.

Contribution. In Section 3, we present a $O(n+m \log m)$ time algorithm for computing generalized visibility kernel, where n, m are the complexities of P and X, respectively. Our algorithm for computing generalized visibility kernel involves computation of relative convex hulls. *Relative convex hull* of a set of points S is defined as the region with minimum circumference that includes S within a polygon P [10]. In Section 3.2, we present an efficient algorithm for computing the relative convex hull of a set of points in a visibility polygon of a point. Our algorithm works in $O(n + m)$ time when the result of Section 3.1 is given in $O(n+m \log m)$ time cumulatively. Previously, an algorithm is given to find a single viewpoint that sees a given relative convex hull in $O(n + m)$ time [6]. We present an algorithm that calculates the set of all viewpoints of a set of points in the same time complexity.

2 Preliminaries and definitions

The input for $AGP(X, Y)$ is a simple (non-convex) polygon P with n vertices and two sets $X, Y \subseteq P$. We assume that X is given as a union of non-intersecting sets of points, line segments, and convex polygonal regions within P. Let m denote the complexity of X. For any polygon Q, ∂Q denotes the boundary of Q and $ref(Q)$ denotes the reflex vertices of Q.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Two points $p, q \in P$ *see* each other if the whole line segment $\overline{pq}$ is in P. The set of points in P that can be seen from a point $p \in P$ is called the *visibility polygon* of p, denoted as $\mathcal{V}(p)$ (See Figure 1). The set of points that can see every point in $\mathcal{V}(p)$ is called the *visibility kernel* of p, denoted as $\mathcal{VK}(p)$. For a set of points $S \subseteq P$, the set of points that are visible from all points in S is called its *complete visibility polygon* [7]. It is denoted as $\mathcal{CV}(S)$ and can be formulated as $\bigcap_{p \in S} \mathcal{V}(p)$.

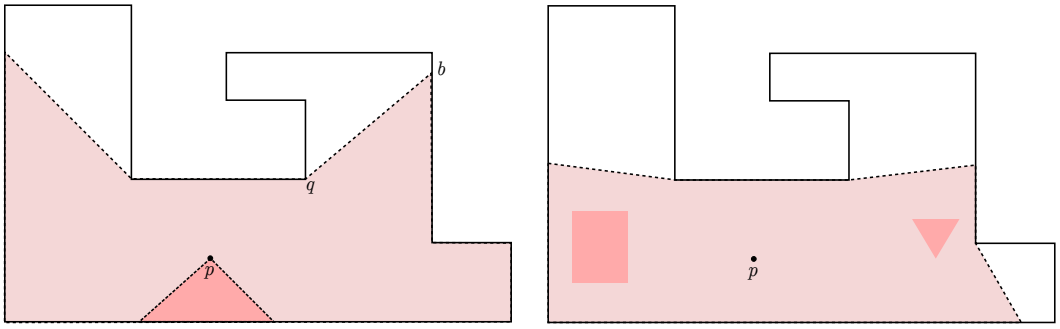


Figure 1 (Left) Visibility polygon $\mathcal{V}(p)$ (shaded area) and visibility kernel $\mathcal{VK}(p)$ (dark shaded area) of a point p. (Right) For a given set X (dark shaded area), $\mathcal{GVK}(p, X)$ is the shaded area.

The counter-clockwise (CCW) angle defined by ordered three points x, p, y is denoted as $\widehat{xpy}$. The shortest path within a polygon P between two points $p, q \in P$ is denoted as $SP(P, p, q)$. A *chain* is an ordered list of j line segments $s_1 ... s_j$ between $j + 1$ points $p_1 ... p_{j+1}$ such that $s_i = \overline{p_i p_{i+1}}$ for $1 \leq i \leq j$. A chain on a boundary of a polygon P between two points $q_1, q_2 \in \partial P$, is defined as the CCW chain on ∂P from q_1 to q_2, denoted as $chain(P, q_1, q_2)$. The half plane on the left of the ray $\overrightarrow{pq}$ is denoted as $HP(\overrightarrow{pq})$. The closure of the half plane is denoted as $CHP(\overrightarrow{pq})$. A *wedge* is determined by three points $l, p, r \in P$ and denoted as $wedge(l, p, r)$. If $\widehat{lpr} \leq \pi$, then $wedge(l, p, r) = \mathcal{V}(p) \cap HP(\overrightarrow{lp}) \cap HP(\overrightarrow{pr})$. Otherwise, $wedge(l, p, r) = \mathcal{V}(p) \cap (HP(\overrightarrow{lp}) \cup HP(\overrightarrow{pr}))$.

2.1 Generalized visibility kernel and its properties

From the definitions, we simply have $\mathcal{GVK}(p, X) = \mathcal{CV}(\mathcal{V}(p) \cap X)$. Also note that if a guard $g \in X$ sees p, then g is guaranteed to see all points in $\mathcal{GVK}(p, X)$. Indeed this very observation has been the motivation for us to propose a generalization of visibility kernels. Below, we list some more properties.

▶ **Lemma 2.1.** *Given two sets $X_1, X_2 \subseteq P$ such that $X_1 \subseteq X_2$. For any point $p \in P$ we have $\mathcal{GVK}(p, X_2) \subseteq \mathcal{GVK}(p, X_1)$.*

▶ **Lemma 2.2.** *For $p \in P$ and $X \subseteq P$, we have $\mathcal{VK}(p) \subseteq \mathcal{GVK}(p, X)$.*

▶ **Lemma 2.3.** *For $p \in P$ and $X \subseteq P$, we have $p \in \mathcal{GVK}(p, X)$.*

▶ **Lemma 2.4.** *The following statements are equivalent for $p, q \in P$ and $X \subseteq P$: (i) Any point $r \in X$ that sees p also sees q; (ii) $q \in \mathcal{GVK}(p, X)$; (iii) $\mathcal{GVK}(q, X) \subseteq \mathcal{GVK}(p, X)$.*

▶ **Theorem 2.5.** *A point set $W \subseteq P$ is a witness set for Y with respect to X if and only if $Y \subseteq \bigcup_{p \in W} \mathcal{GVK}(p, X)$.*

3 Algorithms

Our algorithm for computing $\mathcal{GVK}(p, X) = \mathcal{CV}(\mathcal{V}(p) \cap X)$ consists of four steps. Due to the space limitations, we present the summary of the algorithms in this extended abstract. The details of the algorithms can be found in the full version of the paper.

1. First, we use the linear-time algorithm by Joe and Simpson [8] to compute $\mathcal{V}(p)$.
2. Second, we present a angular plane sweep algorithm to find $\mathcal{V}(p) \cap X$ in Section 3.1.
3. Third, we calculate the relative convex hull of $\mathcal{V}(p) \cap X$ in Section 3.2.
4. Finally, we compute the complete visibility polygon of $\mathcal{V}(p) \cap X$ in Section 3.3 relying on the relative convex hull computed in Step 3.

3.1 Calculating $\mathcal{V}(p) \cap X$

After computing $\mathcal{V}(p)$, we divide $\mathcal{V}(p)$ into wedges $W_i = wedge(r_i, p, r_{i+1})$, where r_i is the ith reflex vertex of $\mathcal{V}(p)$ in CCW order. Assume that $r_1 = r_{j+1}$ where j is the number of reflex vertices. Without loss of generality, we assume that $\widehat{r_j p r_1} \geq \widehat{r_i p r_{i+1}}$ for $1 \leq i < j$. Note that each wedge, except W_j in some cases, is convex. (See Figure 2).

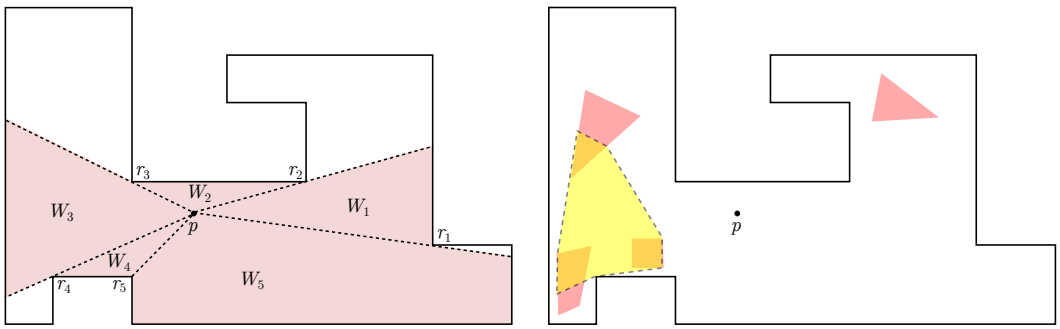


Figure 2 (Left) $\mathcal{V}(p)$ is partitioned into wedges with apices of p. (Right) For a given X (pink and orange shaded areas), $\mathcal{V}(p) \cap X$ is the orange shaded area. Yellow and orange shaded areas together shows $\mathcal{RCH}(\mathcal{V}(p) \cap X)$

.

Our sweepline is a ray anchored at p. We record the geometric objects in X that intersects the sweepline in a balanced binary search tree ($BBST$) in the order of the distance from p to the object. Since the objects in X are convex, the order of the nodes do not change. A node is active if p is closer to the corresponding object than ∂P in the direction of the sweepline. A node of the BBST records the geometric object in X, whether the object is active, and the event points where the node is activated and deactivated.

The event points in our angular plane sweep algorithm are the vertices of X and reflex vertices of $\mathcal{V}(p)$ in the angular order from p. For this, we sort the elements of X in CCW order with respect to the viewpoint p. The node corresponding to an object $x \in X$ is inserted to (deleted from) the BBST at the rightmost (leftmost) vertex of x with respect to p. The object x can be activated or deactivated at a reflex vertex $r \in ref(\mathcal{V}(p))$. We prune x through the sweepline when it is activated or deactivated. The output $\mathcal{V}(p) \cap X$ is recorded as a sequence of nodes sorted in the angular order of the appearance from p.

Each element of $x \in X$ is added to and removed from BBST once, and activated and/or inactivated at most once. Add, remove operates in $O(\log m)$ time, activation and inactivation takes $O(1)$ time per element. Calculating the cutting points of x takes an amortized

$O(1)$ time. Data structure operations in total takes $O(n + m \log m)$ time and sorting takes $O(m \log m)$. Therefore the total complexity of calculating $\mathcal{V}(p) \cap Y$ is $O(n + m \log m)$.

3.2 Calculating the relative convex hull of $\mathcal{V}(p) \cap X$

To calculate $\mathcal{CV}(\mathcal{V}(p) \cap X)$, we use the *relative convex hull* for a set of points S in a polygon P, denoted as $\mathcal{RCH}(S)$ [10]. $\mathcal{RCH}(S)$ is defined as the polygon that has the minimum circumference such that $S \subseteq \mathcal{RCH}(S) \subseteq P$. Note that $\mathcal{RCH}(S)$ can be a degenerate polygon (See Figure 3).

Ghosh [6] states that if a point p in a simple polygon P sees a set of points $S \subseteq P$ if and only if p sees $\mathcal{RCH}(S)$. Based on this, we can conclude that $\mathcal{CV}(\mathcal{V}(p) \cap X) = \mathcal{CV}(\mathcal{RCH}(\mathcal{V}(p) \cap X))$. Let us call $\mathcal{RCH}(\mathcal{V}(p) \cap X)$ as Q for the rest of the paper for brevity.

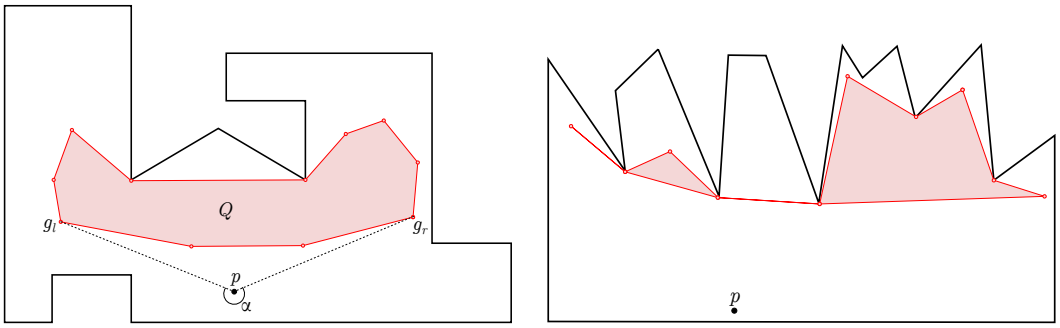


Figure 3 (Left) Q is the relative convex hull of $\mathcal{V}(p) \cap X$ for a given X. The near chain is $chain(Q, g_l, g_r)$, the far chain is $chain(Q, g_r, g_l)$, and α is the outer angle between the tangents from p to Q. (Right) A degenerate relative convex hull.

▶ **Lemma 3.1.** *For a point $p \in P$ and a set $X \subseteq P$, we have $\mathcal{RCH}(\mathcal{V}(p) \cap X) \subseteq \mathcal{V}(p)$.*

Let $g_l, g_r \in Q$ be the points that make the angle $\alpha = \widehat{g_l p g_r}$ maximal such that $wedge(g_l, p, g_r) \cap X = \emptyset$. (See Figure 3). In other words, if $p \notin Q$, then g_l and g_r are the tangent points from p to Q. We decide whether p is in Q based on α.

▶ **Lemma 3.2.** *The point p is in $\mathcal{RCH}(\mathcal{V}(p) \cap X)$, if and only if $\alpha \leq \pi$ or $p \in X$.*

To calculate Q, we define two chains based on whether p is in Q or not. If $p \notin Q$, then $chain(Q, g_r, g_l)$ is the *near chain* and $chain(Q, g_l, g_r)$ is the *far chain*. If $p \in Q$, then ∂Q is the *far chain*. Based on Lemma 3.2, if $\alpha > \pi$ and $p \notin X$, we calculate both the near chain and the far chain. Otherwise, we calculate only the far chain.

For each wedge W_i, we calculate the convex hull of $X \cap W_i$ by calculating the far and (if necessary) the near chains. From the previous calculation, we have $\mathcal{V}(p) \cap X$ in CCW order from the viewpoint of p. We traverse the points in the wedge by updating the far chain in CCW from p. While traversing, if the current point creates a right turn in the far chain, then the previous points are popped from the far chain until there is no right turn in the chain. The process is similar for the near chain.

After calculating the the far and (if necessary) the near chains for each wedge, we merge each of them to calculate Q. The merge is also done in CCW order from p, using the reflex vertices of $\mathcal{V}(p)$ as pivot points and draw tangents to the near and the far chains of the wedges. If $\alpha > \pi$ and $p \in X$, then we concatenate line segments $\overline{g_l p}$ and $\overline{g_r p}$ to the far chain to yield Q. If $\alpha > \pi$ and $p \notin X$, we concatenate the near chain and the far chain. If $\alpha \leq \pi$, only the far chain gives us Q.

The time complexity of calculating Q is $O(n+m)$, since there are $O(n+m)$ points to be considered in the convex hull and each vertex in $\mathcal{V}(p) \cup X$ and $ref(P)$ is traversed once and inserted to and/or removed from a chain at most once.

3.3 Calculating the complete visibility polygon of $\mathcal{RCH}(\mathcal{V}(p) \cap X)$

Here, we present an algorithm to calculate the complete visibility polygon of Q as a sub-method of calculating the $\mathcal{GVK}(p, X)$. However, p is not particularly relevant in our algorithm other than the assumption of the existence of at least one viewpoint for the relative convex hull.

Let k be the number of reflex vertices of Q. Let us define $r_i \in ref(Q)$ as the reflex vertices of Q in the CCW order, where $1 \leq i \leq k$. Without loss of generality, let us assume that r_1 is the reflex vertex that maximizes the angle $\widehat{r_k p r_1}$ and i is an integer such that $1 \leq i \leq k$. We also assume that $r_{k+1} = r_1$. Now, we calculate $\mathcal{CV}(Q)$ by handling different cases depending on some properties of Q.

Case 1: $k = 0$, or $k = 1$. Ghosh [7] gave an algorithm to find the complete visibility polygon of a given convex chain within a simple polygon in linear time. When $k = 0$ i.e., Q is convex, $\mathcal{CV}(Q)$ can be calculated directly using his algorithm. If Q has only one reflex vertex, then we can represent ∂Q as a chain starting and ending at r_1 that has only left turns. This allows us to use the same algorithm to calculate $\mathcal{CV}(Q)$.

Case 2: $k \geq 2$. We have the following property:

▶ **Lemma 3.3.** *If $\widehat{r_i p r_{i+1}} \leq \pi$, then r_i and r_{i+1}, see each other.*

From Lemma 3.3, we yield that there can be at most one consecutive reflex vertex pair in Q that cannot see each other. Then, we have the following sub-cases:

Case 2.1: r_1 and r_k see each other. For each pair of consecutive reflex vertices $r_i, r_{i+1} \in ref(Q)$, we denote the polygon bounded by $chain(Q, r_i, r_{i+1})$ and $\overline{r_i r_{i+1}}$ as Q_i. We call the rest of Q after removing all Q_i's as Q_{core} (See Figure 4).

Case 2.1.1: Q_k is convex. For each Q_i, we define a superset P_i, which is bounded by $\overline{r_i r_{i+1}}$ and $chain(P, r_i, r_{i+1})$. We calculate the complete visibility polygon of Q_i inside P_i using [7] and call it C_i. Then, we create a polygon that is bounded by $chain(C_i, r_i, r_{i+1})$ for all i and call it as G. For each reflex vertex r_i, two incident vertices to r_i on ∂Q are denoted as t_i and u_i, such that t_i, r_i, u_i is the CCW order of these vertices on ∂Q. Then we have $\mathcal{CV}(Q) = G \cap \bigcap_{i=1}^{k} (CHP(\overrightarrow{t_i r_i}) \cap CHP(\overrightarrow{r_i u_i}))$.

We use a similar approach as Lee and Preparata's algorithm [9] to calculate the half plane intersections. Our algorithm starts with initializing a partially bounded intersection region K as $CHP(\overrightarrow{t_1 r_1}) \cap CHP(\overrightarrow{r_1 u_1})$. We initially assign the left and right tangents from r_1 to K as $\overrightarrow{t_1 r_1}$ and $\overrightarrow{u_1 r_1}$ respectively. Then, we traverse the reflex vertices of Q in CCW order starting from r_1. In the ith step, we trim K with the rays $\overrightarrow{r_i u_i}$, $\overrightarrow{t_{i+1} r_{i+1}}$ and $chain(G, r_i, r_{i+1})$ using [5] and update the tangents from r_i to K. After k steps, K yields us $\mathcal{CV}(Q)$.

Case 2.1.2: Q_k is not convex. In this case G will not be a simple polygon. However, we have $\mathcal{GVK}(Q) \subseteq C_k \subseteq CHP(\overrightarrow{r_1 t_1})$ assuming that r_1 is a reflex vertex of Q_k without loss of generality. We initialize C_k to K and iteratively trim K through the reflex vertices of Q using the half plane intersections.

Case 2.2: r_1 and r_k do not see each other. This case is similar to Case 2.1.2, except in this case, Q_k is defined as the polygon bounded by $chain(Q, r_k, r_1)$ and $SP(r_1, r_k)$. Similarly, P_k is bounded by $chain(P, r_k, r_1)$ and $SP(r_1, r_k)$. We calculate C_k as the intersection of the complete visibility polygon of $chain(Q, r_k, r_1)$ and P_k. Then, we use the same algorithm initializing $C_k \cap CHP(\overrightarrow{t_1 r_1}) \cap CHP(\overrightarrow{r_1 u_1})$ to K using [5].

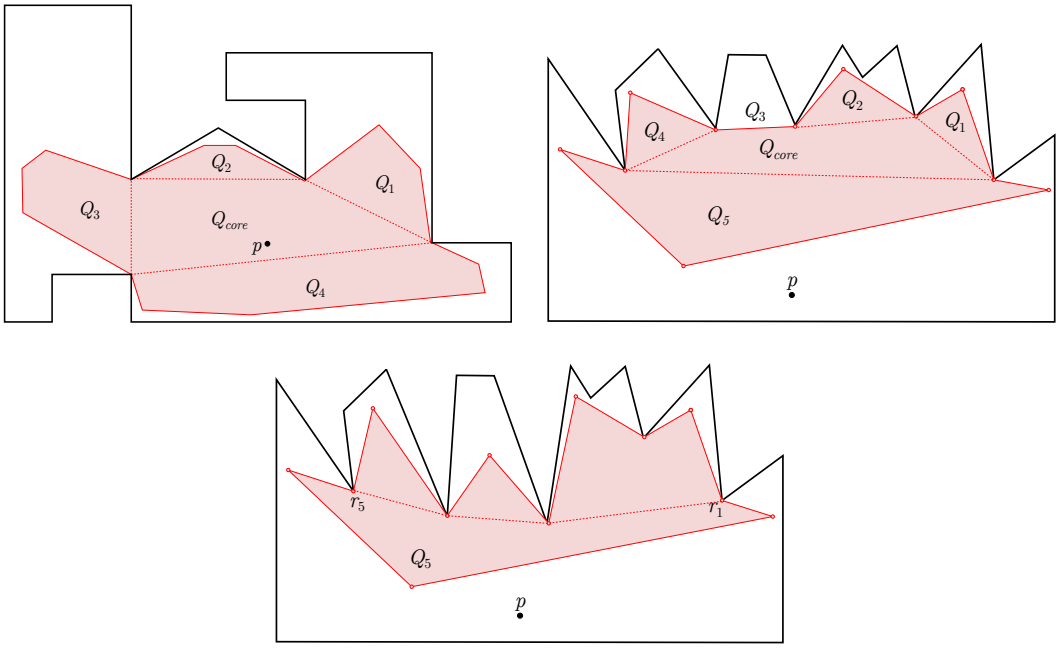


Figure 4 Partition of Q for Cases (Upper Left) 2.1.1, (Upper Right) 2.1.2, (Lower) 2.2.

These cases cover all the possible situations assuming that there exists at least one point in P that sees every point in Q. We can also spot whether $\mathcal{CV}(Q)$ is empty if there are more than one r_i, r_{i+1} pairs that do not see each other or K becomes empty at some step in the half plane intersections. All the submethods can be calculated in linear time with respect to the number of vertices in P, Q, Q_{core} and G all of which are in $O(n+m)$.

References

1 E. S. Ayaz and A. Üngör. An iterative refinement scheme of dominating guards and witnesses for art gallery problems. In *Canadian Conf. on Comp. Geom.*, pages 168–174, 2016.

2 E. S. Ayaz and A. Üngör. Minimal witness sets for art gallery problems. In *European Workshop on Computational Geometry (EuroCG)*, pages 195–198, 2016.

3 K. Chwa, B. Jo, C. Knauer, E. Moet, R. van Oostrum, and C. Shin. Guarding art galleries by guarding witnesses. *Int. J. Comput. Geometry Appl.*, 16(2-3):205–226, 2006.

4 P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering art galleries. *CoRR*, abs/1410.8720, 2014.

5 S. K. Ghosh. A linear-time algorithm for determining the intersection type of two star polygons. In *Foundations of Software Technology and Theoretical Computer Science*, pages 317–330. Springer, 1984.

6 S. K. Ghosh. Computing a viewpoint of a set of points inside a polygon. In *Foundations of Software Technology and Theoretical Computer Science*, pages 18–29. Springer, 1988.

7 S. K. Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12(1):75 – 95, 1991.

8 B. Joe and R. B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987.

9 D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *J. ACM*, 26(3):415–421, July 1979.

10 G. T. Toussaint. An optimal algorithm for computing the relative convex hull of a set of points in a polygon. In *Signal Processing: Theories and Applications*. North-Holland, 1986.

Index of Abstracts

Index of Authors